

Sacha Chua's Emacs configuration

Contents

1	About this file	15
1.1	Debugging tips	16
1.2	Starting up	16
1.3	Emacs initialization	16
1.3.1	Add package sources	16
1.3.2	Review packages when upgrading	17
1.3.3	Add my elisp directory and other files	17
1.4	Personal information	17
1.5	System information	17
1.6	Reload	17
1.7	Backups	17
1.7.1	Obscure Emacs package appreciation: backup-walker	18
1.8	History	21
1.9	Disabling the toolbar	22
1.10	Change "yes or no" to "y or n"	22
1.11	Minibuffer editing - more space!	22
1.12	Killing text	22
2	Keybindings	23
2.1	Embark	23
2.1.1	Using Embark and qrencode to show a QR code for the Org Mode link at point	25
2.1.2	TODO Using Embark to act on video	26
2.1.3	Using Embark to act on audio	26
2.1.4	Using Embark to insert files as Org INCLUDEs	27
2.1.5	Using Embark to offer context-sensitive actions for Org elements	28
2.1.6	Whichkey and Embark	29
	DONE Changing the which-key labels for shortcuts	29
2.1.7	Embark and images	31
2.1.8	Converting handwriting to text	32
2.1.9	Renaming and storing	33
2.1.10	Embark and subed	36
2.1.11	Embark, symbols, and casual-symbol-overlay	37
2.1.12	Embark and erefactor-rename-symbol-in-buffer	38
2.1.13	Embark and Org Mode timestamps	38
2.2	Extended command list	38
2.3	Menus	41
2.4	Context menus	41
2.5	Repeatable commands	41
2.6	Hydra keyboard shortcuts	41
2.6.1	Emacs Hydra: Allow completion when I can't remember the command name	44
2.7	which-key and which-key-posframe	45
2.8	Casual	45
2.9	Foot pedal	45

3	Completion	46
3.0.1	Emacs completion and handling accented characters with orderless	46
3.1	Consult	47
3.1.1	Completing blog posts	48
3.1.2	Completing sketches	49
3.1.3	Consult directory navigation	51
3.1.4	Using projects as a source for consult-buffer	52
3.1.5	consult-omni	52
	SOMEDAY Using web searches and bookmarks to quickly link placeholders in Org Mode	54
	Bookmarks	57
	DONE Finding my blog posts with consult-omni	57
3.1.6	STARTED Searching my blog, notes, and sketches with consult-ripgrep and consult-omni	59
3.2	Marginalia	61
3.2.1	Marginalia and annotating journal entries	62
3.3	Cargo-culted stuff	63
4	Appearance	63
4.1	color-theme sometimes comes across lists. Odd!	63
4.2	Display	63
4.3	Color theme	64
4.3.1	Set up a color scheme	64
4.3.2	Making highlight-sexp follow modus-themes-toggle	64
4.4	Modeline	65
4.4.1	Time in the modeline	65
4.4.2	Diminish mode names in modeline	65
4.4.3	Highlight the active modeline using colours from modus-themes	65
4.5	Quickly adding face properties to regions	66
5	Navigation	67
5.1	Going to the last change	68
5.2	Substitution	68
5.3	Expand region	68
5.4	selected.el: Special keymap when the region is active	68
5.5	Copy and append string	69
5.6	PDF view	69
5.6.1	TOBLOG Copy text from current PDFview page in other window	69
5.7	Links	70
5.8	Manage windows	70
5.8.1	Rotate windows	70
5.8.2	Jump between windows	70
5.8.3	Focus on the current window	70
5.8.4	Winner mode - undo and redo window configuration	70
5.9	TODO Get the hang of using vundo	71
5.10	Get scroll-other-window to work with PDFs	71
5.11	Manage positions	71
5.11.1	Save place between sessions	71
5.11.2	Quickly jump to positions	71
5.12	TODO Sort files in read-file-name	72
5.13	Downloaded files	73
5.13.1	Replace with latest download	74
5.14	Searching	74
5.14.1	Deleting things	75

5.14.2	Transient for isearch	75
5.14.3	Search invisible text	77
5.14.4	Occur	77
5.15	Ediff	77
5.16	Hideshow	77
5.17	Pop to mark	78
5.18	Helm-swoop - quickly finding lines	78
5.19	Highlight the current line while still being able to easily customize/describe under- lying faces	79
5.20	Windmove - switching between windows	79
5.21	Frequently-accessed files	79
5.22	C-g improvement	81
5.23	Smartscan	82
5.24	Dired	82
5.24.1	Using image-dired to browse the latest screenshots from multiple directories	82
5.24.2	peep-dired	84
5.24.3	Saving photos	85
5.25	Move to beginning of line	86
5.26	Recent files	86
5.27	Open files externally	86
5.28	Emacs: Toggle a buffer for quick reference	87
5.29	link-hint	88
5.30	Bookmarks	88
5.31	Dogears	89
5.32	Randomness for serendipity	89
5.32.1	Building a today-I-learned habit, and displaying the documentation for ran- dom Emacs commands	89
5.32.2	Shuffling lines	90
5.33	Network: TRAMP and editing files over SSH	90
5.34	Touch gestures	90
5.35	Applications	91
6	Reading	91
7	Writing and editing	91
7.1	Learning French	92
7.1.1	Practice pronunciation	93
7.1.2	Start the process for transcribing the latest recording	96
7.1.3	Emacs Lisp and NodeJS: Getting the bolded words from a section of a Google Document	96
7.1.4	Emacs and French: Focus flycheck-grammalecte on the narrowed part of the buffer	102
7.1.5	Speech synthesis	103
	gtts-cli	103
	DONE Azure	103
7.1.6	Add shadowing with tts to subed-record	103
7.1.7	Make it easy to add reference links	104
7.1.8	Word timestamps	104
7.1.9	AI feedback	105
7.1.10	Using flycheck to display AI grammar feedback from gptel as I learn French	106
7.1.11	Process audio files	106
7.1.12	Save journal entries for analysis	107
7.1.13	TODO Load en-fr dictionary	108
7.1.14	Conjugation	109

7.1.15	Looking up words via wordreference	110
7.1.16	Lexique	112
7.1.17	STARTED Write a completion-at-point function for French	114
7.1.18	Highlight and count new words in journal entries	114
7.1.19	Doublecheck with Google Translate	116
7.1.20	Correct encoding errors	117
7.2	Spellcheck	117
7.3	gif-screencast	117
7.4	Sentences end with a single space	119
7.5	Writeroom	119
7.6	Try redacting	119
7.7	DONE Audio braindump workflow tweaks: Adding Org Mode hyperlinks to recordings based on keywords	121
7.7.1	Recognizing keyword phrases	122
7.7.2	Splitting the lines based on keywords and oopses	126
7.7.3	Preparing the VTT subtitles	128
7.7.4	Formatting the subtitles into Org Mode subtrees	128
7.7.5	Process a single transcript from the raw text file	131
7.7.6	Process multiple files	132
7.7.7	Ideas for next steps	134
7.8	DONE Updating my audio braindump workflow to take advantage of WhisperX	134
7.9	Markdown	136
7.10	Screenshot	137
7.11	Avoiding weasel words	138
7.12	Emacs: Cycle through different paragraph formats: all on one line, wrapped, max one sentence per line, one sentence per line	138
7.13	Visual line	141
7.14	Unicode	141
7.15	Clean up spaces	141
7.16	Expand	141
7.17	Write about keybindings	142
7.18	Transcripts from my phone	142
7.19	Speech recognition	143
7.19.1	Map lang-gptel feedback from the logbook to KwizIQ topics	143
	Fréquence des erreurs par catégorie	144
	Other	144
	Quiz de révision (10 questions)	144
7.19.2	Using whisper.el to convert speech to text and save it to the currently clocked task in Org Mode or elsewhere	144
7.19.3	Emacs and whisper.el: Trying out different speech-to-text backends and models	152
7.19.4	Queuing multiple transcriptions with whisper.el speech recognition	156
7.19.5	Using Silero voice activity detection to automatically queue multiple transcriptions with natrys/whisper.el	160
7.19.6	Slowly building speech-based commands for Emacs	161
7.19.7	Okay, track...	162
7.19.8	Using speech recognition for on-the-fly translations in Emacs and faking in-buffer completion for the results	163
7.19.9	Expanding yasnippets by voice in Emacs and other applications	164
7.19.10	Streaming speech recognition into Emacs using Google Chrome Web Speech API	166
	speech and subed-record	172
7.19.11	Try speeches with realtime	173
7.19.12	TOBLOG Using Emacs Lisp to send audio files to Deepgram and format VTTs	174
7.19.13	TOBLOG Rerecognize this audio and reprocess it	178

7.19.14	Gladia	178
7.19.15	DONE Getting live speech into Emacs with Deepgram's streaming API	181
	General code	181
	Display in speech buffer	183
	Display words per minute	184
	Append to EmacsConf Etherpad	185
7.20	UTF-8	185
7.21	Wdiff	185
7.22	Denote	187
8	Org Mode	187
8.1	Time zones	187
8.2	Set file property	188
8.3	Automatically continue lists	189
8.4	PDFs	189
8.5	After I jump to a task from org-clock-goto, narrow to it automatically	189
8.6	Find first common Org Mode heading	189
8.7	Writing about sketches and including their text	190
8.8	Remove open Org Mode clock entries	191
8.9	Insert a link to an Org Mode heading from an org-refile prompt	191
8.10	Org - mapping blog posts and image URLs from bulk exports	192
8.11	Move Org Mode properties from subtree to parent	193
8.12	My files	194
8.13	Modules	194
8.14	Keyboard shortcuts	194
8.14.1	Speed commands	195
	Org Mode: Cutting the current list item (including nested lists) with a speed command	195
8.14.2	Other speed commands	196
8.15	Org navigation	198
8.15.1	Link Org subtrees and navigate between them	198
8.15.2	Viewing, navigating, and editing the Org tree	198
8.15.3	Finding my place on a small mobile screen with org-back-to-heading	199
8.15.4	Dealing with big tables	199
8.16	Taking notes	199
8.16.1	Date trees	199
8.16.2	Templates	199
	Allow refile in the middle(ish) of a capture	203
8.16.3	Try out this capture command	203
8.16.4	Estimating WPM	204
8.17	Logbook	204
8.18	Tasks	205
8.18.1	Managing tasks	205
	Get things to be set to TODO when they repeat	205
	Track TODO state	205
	TODO Change Org Mode TODO keyword color based on the state and the current Modus theme	205
	Projects	206
	Tag tasks with GTD-ish contexts	207
	Enable filtering by effort estimates	207
	Track time	208
	Habits	208
8.18.2	Estimating tasks	208
8.18.3	Flexible scheduling of tasks	208

8.18.4	Task dependencies	209
8.18.5	Quick way to archive all DONE from inbox	209
8.18.6	Strike through DONE headlines	209
8.18.7	Checklists	209
8.19	Templates	210
8.19.1	Structure templates	210
	Demarcate, but for all blocks	210
8.19.2	Emacs chats, Emacs hangouts	212
8.20	Org agenda	213
8.20.1	Basic configuration	213
8.20.2	Starting my weeks on Saturday	214
8.20.3	Display projects with associated subtasks	214
8.20.4	Org agenda custom commands	216
8.20.5	Shuffling my Org Mode unscheduled tasks	221
8.20.6	Making it easier to tag inbox items	221
8.20.7	Make it easy to mark a task as done	221
8.20.8	Make it easy to mark a task as done and create a follow-up task	222
8.20.9	Capture something based on the agenda	222
8.20.10	Sorting by date and priority	222
8.20.11	Preventing things from falling through the cracks	224
8.20.12	Synchronizing with Google Calendar	224
8.20.13	Projects	224
8.21	Reviews	225
8.21.1	Weekly review	225
	Sketch list	231
	Link-related convenience functions	231
8.21.2	Monthly reviews	232
8.21.3	Emoji summaries	235
8.22	Filing	236
8.22.1	DONE Org Mode: Prompt for a heading and then refile it to point	236
8.22.2	TODO Bounce to another file	237
8.22.3	Basic refiling configuration	237
	TEACH Jump to Org location by substring	238
	Quick way to jump	239
8.22.4	TODO Refile inbox entries to a smaller set of org-refile-targets	239
8.22.5	DONE Automatically refiling Org Mode headings based on tags	240
8.22.6	Moving lines around	242
8.22.7	Organizing my blog index	243
8.22.8	Refiling Org Mode notes to headings in the same file	244
8.23	Contacts	244
8.24	Inserting code	245
8.25	Bookmarks	246
8.26	Org Babel	248
8.26.1	Make it easier to split my literate config into files	248
8.26.2	Linking to Org Babel source in a comment, and making that always use file links	249
8.26.3	DONE Org Mode: Tangle Emacs config snippets to different files and add boilerplate	250
8.26.4	Org Babel: Detangle just the current block	255
8.26.5	Format source	256
8.26.6	Run source blocks in an Org Mode subtree by custom ID	256
8.26.7	Execute named babel block	257
8.26.8	JSON	257
	JQ	258

8.26.9	Fix block indentation	258
8.26.10	Let's try literate-elisp	258
8.27	Publishing	258
8.27.1	HTML	258
	Changing Org Mode underlines to the HTML mark element	259
	Include inline SVGs in Org Mode HTML and Markdown exports	260
	Org Mode: Export HTML, copy files, and serve the results via simple-httpd so that media files work	262
8.27.2	11ty static site generation	265
	DONE Linking to blog topics	270
	Linking to blog posts	271
	embark-11ty	281
	Moving my Org post subtree to the 11ty directory	281
	Listing exported Org posts	285
	Include Mastodon, HN, Reddit fields in front matter	285
	Comments	286
8.27.3	L ^A T _E X	286
8.27.4	TODO Copy Tasker task	287
8.27.5	DONE Remove heading from TOC	287
8.27.6	Counting words without blocks	288
8.27.7	Org Mode: Including portions of files between two regular expressions	289
8.27.8	ox-epub	295
8.27.9	DONE Add a note to the bottom of blog posts exported from my config file	295
8.27.10	Copy linked file and change link	295
8.27.11	Org Mode: Create a quick timestamped note and capture a screenshot	297
8.27.12	Cleaning up export	299
8.27.13	Publish without prompting	300
8.27.14	Special blocks	301
8.27.15	Abbreviations	303
8.27.16	Adding a custom header argument to Org Mode source blocks and using that argument during export	304
8.27.17	Stylesheet / header	305
8.27.18	Footer	306
8.27.19	Copy region	306
8.27.20	UTF-8 checkboxes	307
8.27.21	Beamer	307
8.27.22	PlantUML	307
8.27.23	ox-hugo	307
8.27.24	Org Mode: Asynchronous export and tangle of a large file	308
8.27.25	Plain text	310
8.27.26	PDF	311
8.28	Org roam	313
8.29	Links	314
8.29.1	Convert an inline link into a side note/footnote	314
8.29.2	Adding Org Mode link awesomeness elsewhere: sach-org-insert-link-dwim	314
8.29.3	IDs	320
8.29.4	Quick links	320
8.29.5	Tag links	320
8.29.6	Links to my config	320
	TODO add dotemacs completion	321
8.29.7	YouTube	322
8.29.8	Videos	324
8.29.9	Linking to a specific time in a video	327
8.29.10	Audio	328

8.29.11	Captions	330
8.29.12	Using an Emacs Lisp macro to define quick custom Org Mode links to project files; plus URLs and search	332
	Copy web link	334
	Quickly search my code	337
8.29.13	Linking to headings that match a tag	339
8.29.14	Links from org-protocol	339
8.29.15	Fix elisp links	341
8.29.16	IRC	341
8.29.17	Dired	341
8.30	Org protocol: following Org links from outside Emacs	342
8.31	TODO Speed command for adding a custom ID to Org Mode posts	343
8.32	Journal	345
8.32.1	Working with journal entries	352
8.32.2	Tagging journal entries	353
8.32.3	Photos	355
8.32.4	Moments	356
8.32.5	Slicing and dicing the journal entries	356
8.33	Attachments	357
8.34	HTTP	357
8.35	Lilypond	357
8.36	Diagrams and graphics	358
8.36.1	Treemap visualization	359
8.37	Org Babel, Mermaid JS, and fixing "Failed to launch the browser process" on Ubuntu 24	360
8.38	Counting	362
8.39	Spreadsheets	363
8.40	Literate programming	363
8.40.1	Editing source code	363
8.40.2	Copying and sharing code	363
8.40.3	Tables	363
8.41	Invoices	363
8.42	Presentations	366
8.42.1	Counting words	366
8.43	Allow dashes in tags	367
8.44	Convert from Markdown	367
8.45	Copying information from my phone	368
8.46	Emacs packages, other settings for easy Emacs News generation	368
8.46.1	ASCII export	368
8.46.2	Reddit	368
8.46.3	Sorting Org Mode lists using a sequence of regular expressions	369
8.46.4	Package links	370
8.47	Save when Emacs loses focus	371
8.48	Clipboard	372
8.49	Setting properties	372
8.50	Linking to and exporting function definitions in Org Mode	373
8.50.1	Still allow linking to the file	376
8.51	Including variables	376
8.52	Org - send things to the bottom of the list	378
8.53	Org Mode: Format Libby book highlights exported as JSON	379
8.54	STARTED Validation	381
8.54.1	Keep only unique headings	381
8.54.2	TODO No blank titles, no duplicate paths	381
8.55	Sorting completion candidates, such as sorting Org headings by level	382

8.56	Vector search	385
8.56.1	Indexing-related code	385
8.56.2	Consult-based interface for searching blog posts	386
	Index automatically	388
	Handle sketches too	388
	Multiple sources	389
8.56.3	emacs-rag-search?	390
9	Multimedia	391
9.1	Emacs.tv	391
9.2	Timestamps	391
9.3	Images	392
9.3.1	Save edited text for sketch post	392
9.3.2	TODO Make header for Mastodon	392
9.3.3	Imagemagick	392
	Rotate clockwise or counterclockwise	392
	TOBLOG Emacs: Extract part of an image to another file	393
	TODO Make an image square	396
	TODO Animate highlighting part of an image	396
9.3.4	Artrage	397
9.3.5	SVG	397
	Animating SVGs	397
9.3.6	Finding sketches	422
9.3.7	Renaming and recoloring sketches	424
9.3.8	TODO Interactively recolor a sketch	424
9.3.9	Org Mode sketch: links	427
9.3.10	Org Mode custom link: copy to clipboard	430
9.3.11	Helm completion with sachahelm-org-sketches	431
9.3.12	Button-based interface	432
9.3.13	Templates	434
9.3.14	Dot-grid box templates	435
9.3.15	Easily backfill my journal	440
9.3.16	Rename scanned index cards	441
9.3.17	Automatically resize images	443
9.3.18	Get information for sketched books	443
9.3.19	Make it easy to follow up on a sketch	444
9.3.20	Digital index piles with Emacs	444
9.3.21	Xournalpp and Krita	448
9.3.22	Sketched books	449
9.3.23	Other sketches	450
9.3.24	Other sketch-related functions	452
9.3.25	SOMEDAY Write about half-page scans	453
9.3.26	Doodles	453
9.3.27	Supernote	454
	Using Emacs Lisp to export TXT/EPUB/PDF from Org Mode to the Super-	
	note via Browse and Access	458
	org-attaching the latest image from my Supernote via Browse and Access	459
	DONE Using Puppeteer to grab an image from the SuperNote's screen mirror	462
9.4	Audio	463
9.5	FFmpeg	463
9.6	Subtitles with Subed	463
9.6.1	Reformat speaker in a two-speaker transcript	463
9.6.2	Interleave images with transcript	464
9.6.3	TOBLOG Split a transcript into phrases for subtitles	465

9.6.4	Transcript editing	466
9.6.5	Remove underlining from WhisperX VTT	467
9.6.6	Adjust subtitles	467
9.6.7	Extract part of a video	468
9.6.8	Hide IDs and times	470
9.6.9	Other subtitle code	471
9.6.10	Simplify inserting audio links	472
9.6.11	Using Emacs to fix automatically generated subtitle timestamps	475
9.6.12	Using word-level timing information when editing subtitles or captions in Emacs	476
9.6.13	Using WhisperX to get word-level timestamps for audio editing with Emacs and subed-record	479
9.6.14	Testing subtitle start times by skimming the first second	483
9.6.15	DONE Remove filler words at the start and upcase the next word	484
9.6.16	Showing captions	485
9.6.17	Edit text	485
9.6.18	Working with media	488
9.6.19	DONE Working with sections defined by NOTE comments	488
9.6.20	Split up oops better	490
9.6.21	TODO Org Mode: Insert YouTube video with separate captions	493
9.6.22	Export transcript as list	494
9.6.23	Removing gaps and merging subtitles	495
9.6.24	Using scripts to correct transcripts	496
9.7	Creating	497
9.7.1	Prepare for EmacsConf screenshots or recordings	497
9.7.2	Preparing to record YouTube shorts	497
9.8	Working with or renaming a set of files	498
9.9	Elfeed	500
10	Coding	501
10.1	Comments	501
10.2	DONE Scan ~/bin and turn the scripts into interactive commands	501
10.3	CSVs	502
10.4	Whitespace	502
10.5	Python	502
10.6	Web development	503
10.7	LSP	504
10.8	Turbo log	505
10.9	Tab width of 2 is compact and readable	505
10.10	More indentation things	506
10.11	Alignment	506
10.12	YAML	506
10.13	Expand region with expreg	506
10.14	Compilation	506
10.15	Emacs Lisp	507
10.15.1	Prefix for writing functions	507
10.15.2	Easily override existing functions	507
10.15.3	Lispy	508
	SOMEDAY Keep track of the number of times specified commands have been called	508
	Emacs: Making a hydra cheatsheet for Lispy	509
10.15.4	Smartparens mode	510
10.15.5	Edit list	512
10.15.6	General-purpose Emacs Lisp libraries	512

10.15.7	Let's try this setup	512
10.15.8	Edebug	513
10.15.9	Testing	513
	ERT	513
	Buttercup	515
	Undercover	519
10.15.10	Eldoc	520
10.15.11	Refactoring	521
10.15.12	Jumping to code	521
10.15.13	YE11: Fix find-function for Emacs Lisp from org-babel or scratch	522
10.15.14	Sorting	525
10.15.15	Evaluation	525
10.15.16	Auto insert	526
10.15.17	Stubbing	527
10.15.18	Helpful	527
10.15.19	Looking up help	527
10.15.20	elisp-demos	527
10.15.21	Democratize	528
10.15.22	JSON	528
10.15.23	Useful libraries	528
10.15.24	Other useful functions	528
10.15.25	Try getting the differences between two lists of strings in terms of edit operations	528
10.16	Snippets	529
10.17	Show column number	531
10.18	Don't show whitespace in diff, but show context	531
10.19	Javascript	531
10.19.1	Node	533
10.19.2	Indium	533
10.19.3	React	533
10.20	Typescript	533
10.21	HTML	533
10.22	Shell	534
10.22.1	Shellcheck	535
10.22.2	dwim-shell-command	535
10.22.3	Exec path from shell	537
10.23	Magit - nice git interface	537
10.23.1	TODO Make this better by adding a post command options variable	539
10.23.2	Finding repos with uncommitted changes	539
10.23.3	Use difftastic	540
10.24	Forge	541
10.25	Checking things out	542
10.26	git-messenger - shows commit message	542
10.27	Tag files	542
10.28	Projects and projectile	543
10.28.1	Capturing notes to per-project files	543
10.29	Exploring MELPA recipes	544
10.30	Ruby	544
10.31	Skewer	545
10.32	Autocomplete	545
10.33	Tern - for Javascript	545
10.34	Docker	546
10.35	Automation	546
10.35.1	Multiple cursors mode	546

10.36	Eshell	546
10.36.1	Eshell completion	547
10.36.2	Correctly complete commands in subdirectories	547
10.37	SQLite	549
10.38	Crontab	549
10.39	TODO Write a function to help with crontab entries	549
10.40	Documentation	550
11	Internet Relay Chat	550
11.1	Search logs	551
12	Mastodon	551
12.1	Adding Mastodon toots as comments in my 11ty static blog	553
12.2	mastodon.el: Copy toot content as Org Mode	555
12.3	mastodon.el: Mention people based on regexp	555
12.4	mastodon.el: Collect handles in clipboard (Emacs kill ring)	558
12.5	mastodon.el: Copy toot URL after posting; also, copying just this post with 11ty	560
12.6	Storing Mastodon links in Org mode	564
12.7	Collecting Emacs News from Mastodon	564
12.8	Copy Mastodon link for Emacs News	567
12.9	Combining Mastodon timelines using mastodon.el	569
12.10	Following people	574
12.11	Tooting a link to the current post	575
12.12	Compose a Mastodon toot with the current Org subtree	577
12.13	Posting the latest screenshot with mastodon.el	577
12.14	Mastodon keyboard shortcuts via Hydra	578
12.15	Making it easier to toot my config	580
12.16	Org contacts	580
12.16.1	Capture	580
12.16.2	Completion	580
12.17	Copy Mastodon toot URL as author link	582
12.18	Collect my recent toots in an Org file so that I can refile them	582
12.19	Archive toots on my blog	583
13	Web	584
13.1	Emacs: Open URLs or search the web, plus browse-url-handlers	584
13.2	Checking URLs	587
13.3	Search	587
13.4	Parsing RSS and Atom feeds	588
13.5	Spookfox	589
13.5.1	Link to current webpage from Spookfox	589
13.5.2	STARTED Link to currently-selected text using Spookfox	590
13.5.3	Running the current Org Mode Babel Javascript block from Emacs using Spookfox	590
13.5.4	Using Spookfox to scroll Firefox up and down from Emacs	592
13.5.5	Emacs and Spookfox: org-capture blue the current tab from Firefox or a link from the page	594
14	Self-tracking, statistics, and other data transformations	594
14.1	Quantified Awesome	594
14.1.1	Child time!	596
14.1.2	Make a tablist of my time entries	596
14.2	Compare times and effort estimates	598
14.2.1	Using the calendar-date-echo-text variable to help plot a heatmap on a year-long calendar in Emacs	598

14.3	Workrave	600
14.4	Blog	600
14.5	Time tracking, previous weekly review	601
14.5.1	List upcoming tasks so that I can see if I'm overloaded	602
14.5.2	Compare time use	603
15	Emacs and my phone	604
15.1	Syncthing	605
16	Clipboard	607
17	Mail and news	607
17.1	TOBLOG Send mail asynchronously	607
17.2	Notmuch	608
17.2.1	Act on current message with Embark	608
17.2.2	Add comment to blog post	609
17.3	Gnus	609
17.3.1	Windows	611
17.4	Approve or discard Mailman messages	611
18	Collaboration	612
18.1	Bike Brigade	612
18.1.1	Transforming HTML clipboard contents with Emacs to smooth out Mailchimp annoyances: dates, images, comments, colours	612
	Automating buttons	615
	Transforming HTML	616
	Saving images	617
	Cleaning up	621
	Removing sections	621
	Removing unneeded styles	622
	Formatting calls to action	623
	Changing link colours	624
	Just the headings	626
	Wrapping it up	626
	Next steps	627
18.1.2	Getting a Google Docs draft ready for Mailchimp via Emacs and Org Mode	628
18.1.3	Bike Brigade: working with Mailchimp images	632
18.1.4	Emacs: Updating a Mailchimp campaign using a template, sending test e- mails, and scheduling it	638
19	Streaming	641
19.1	Mode for streaming	641
19.2	Display large text (and maybe QR code?)	645
19.3	OBS websockets	647
19.4	Comment overlay via socialstream.ninja	649
19.5	Custom Org link type for hints (and sound effects)	649
19.6	Chat	650
19.7	DONE Send currently-clocked task title to file, include in stream	651
19.8	Stream agenda	652
19.9	Simple streaming with FFmpeg	652
19.10	Controlling my stream audio from Emacs: background music, typing sounds, and push to talk	654
19.11	More background music	657
19.12	Show Emacs-related tasks	657
19.13	Stream message	657

19.14	Playing recordings	657
19.15	Stream notes	659
19.16	Chapters	660
19.17	STARTED Download and post-process	660
19.18	CANCELLED Try continuous streaming and the Google Speech Recognition API	661
19.19	Make chapter markers and video time hyperlinks easier to note while I livestream	663
19.19.1	Capture timestamps	663
19.19.2	Calculate an Org timestamp's offset into a YouTube stream	664
19.20	Create a Google Calendar event from an Org Mode timestamp	667
19.21	Create YouTube livestream broadcasts from Emacs Lisp	668
19.22	Insert an Org Mode link to an upcoming YouTube broadcast	672
20	Miscellaneous	674
20.1	Ledger	674
20.2	SSH and <code>-daemon</code>	676
20.3	Encryption	676
20.4	Stardew Valley	676
21	Other cool configs you may want to check out	677
22	Inactive/infrequent things	677
22.1	Old Flickr/Evernote export	677
22.2	Animation for Emacs chats	678
22.3	Oddmuse	678
22.4	Beeminder	679
22.5	Plover	679
22.5.1	Looking things up	679
22.5.2	Adding steno hints as I type	681
22.5.3	Running Plover drills from Emacs	685
22.5.4	Making it easier to execute commands	686
22.5.5	Suggesting briefs	688
22.5.6	Practising within Emacs	690
22.5.7	Editing subtitles	694
22.5.8	Using inotify to add Plover Clippy suggestions into Emacs	695
22.5.9	Stenoing interface	697
22.5.10	Cheat sheets	698
22.5.11	Coding with Plover	699
22.5.12	Measuring WPM	700
22.5.13	Displaying frequency-sorted completions with stroke hints	700
22.6	Key chords	702
22.7	Exwm	703
22.8	Emacspeak	704
22.9	TOBLOG Manage photos with geeqie	704
22.10	EmacsConf	708
22.11	ChatGPT, AI, and large-language models	709
22.11.1	agent-shell	711
22.11.2	Interact with Google Gemini web interface through Spookfox	711
22.12	Chronos timers	711
22.13	Idle timer	712
22.14	Completion at point?	712
22.15	Enable minibuffer completion	713
22.16	Encryption	713
22.17	Tools for organizing	713
22.18	Fun and games	714

22.18.1	TODO Make memes from Emacs	714
22.18.2	Cubing	714
	Rubik's Cube	714
	Diagrams	714
22.18.3	Minecraft	718
22.18.4	Typing of Emacs	718
22.19	Speech synthesis (experimental)	719
22.20	Comparison-shopping with Org Mode	719
23	Style stuff and other exports	725
24	Index of sections by module	725

1 About this file

This is my personal config. It's really long, but that's partly because I sometimes leave blog posts in it as commentary, and also because I've got a lot of little customizations that I might not even remember. =) If you want to see a table of contents and other useful niceties, go to <https://sachachua.com/dotemacs>. Other links for this page: [Org Mode version](#), [Codeberg repository](#), [Github repository](#)

I've installed a lot of packages. See the [package sources](#) (p. 16) section to add the repositories to your configuration. When you see `use-package` and a package name you might like, you can use `M-x package-install` to install the package of that name.

If you're new to Emacs Lisp, you probably don't want to copy and paste large chunks of this code. Instead, copy small parts of it (always making sure to copy a complete set of parentheses) into your `*scratch*` buffer or some other buffer in `emacs-lisp-mode`. Use `M-x eval-buffer` to evaluate the code and see if you like the way that Emacs behaves. See [An Introduction to Programming in Emacs Lisp](#) for more details on Emacs Lisp. You can also find the manual by using `C-h i` (`info`) and choosing "Emacs Lisp Intro".

If you're viewing the Org file, you can open source code blocks (those are the ones in `begin_src`) in a separate buffer by moving your point inside them and typing `C-c ' (org-edit-special)`. This opens another buffer in `emacs-lisp-mode`, so you can use `M-x eval-buffer` to load the changes. If you want to explore how functions work, use `M-x edebug-defun` to set up debugging for that function, and then call it. You can learn more about edebug in the [Emacs Lisp](#) manual.

I like using `(setq ...)` more than `Customize` because I can neatly organize my configuration that way. Ditto for `use-package` - I mostly use it to group together package-related config without lots of `with-eval-after-load` calls, and it also makes declaring keybindings easier.

Nudged by Prot's configuration, I think I'm going to slowly start splitting off my code into modules that define functions and files that set behaviours.

[Index of sections by module](#) (p. 725)

Here's my `early-init.el`:

```
(setq load-path (cl-remove-if (lambda (p) (string-match-p "lisp/org$" p)) load-path))
(add-to-list 'load-path "~/vendor/org-mode/lisp")
(add-to-list 'load-path "~/vendor/org-mode/contrib/lisp")
(load "~/vendor/org-mode/lisp/org-loaddefs.el" nil t)
(setq user-lisp-directory "~/sync/emacs/lisp")
```

Here's my `init.el`:

```
(load-file "~/sync/emacs/Sacha.el")
(load-file "~/sync/cloud/.emacs.secrets")
(load custom-file t)
(put 'narrow-to-region 'disabled nil)
(put 'list-timers 'disabled nil)
(server-mode 1)
```

Sacha.el is what `M-x org-babel-tangle` (`C-c C-v t`) produces.

A note about Org updates: I like running Org Mode from checked-out source code instead of package.el. I add the Lisp directories to my `load-path`, and I also use the `:load-path` option in my first `use-package org` call to set the load path. One of those is probably doing the trick and the other one is redundant, but maybe it's a belt-and-suspenders sort of thing. Using the git checkout also makes upgrading Org easy. All I have to do is `git pull; make`, and stuff happens in an external Emacs process. Since I create `Sacha.el` via `org-babel-tangle`, my Emacs config can load `Sacha.el` without loading Org first.

1.1 Debugging tips

If things break, I can use:

- `check-parens` to look for mismatched parentheses
- `bug-hunter` to bisect my configuration
- `trace-function-background` to get information printed to a buffer
- `profiler-start` to find out more about slow functions

1.2 Starting up

Here's how we start:

```
;; -*- lexical-binding: t -*-
;; This sets up the load path so that we can override it
(setq warning-suppress-log-types '(((package reinitialization))) (package-initialize)
(add-to-list 'load-path "/usr/local/share/emacs/site-lisp")
(setq custom-file "~/config/emacs/custom-settings.el")
(setq use-package-always-ensure t)
```

Memoize is handy for improving the performance when I use slow functions multiple times.

```
(use-package memoize)
```

1.3 Emacs initialization

1.3.1 Add package sources

```
(unless (assoc-default "melpa" package-archives)
  (add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t))
(unless (assoc-default "nongnu" package-archives)
  (add-to-list 'package-archives '("nongnu" . "https://elpa.nongnu.org/nongnu/") t))
```

Use `M-x package-refresh-contents` to reload the list of packages after adding these for the first time.

1.3.2 Review packages when upgrading

Emacs 31 onwards:

```
(setq package-review-policy t
      package-review-diff-command '("git" "diff" "--no-index" "--color=never" ↵
      ↵ "--diff-filter=d"))
(add-to-list 'display-buffer-alist
  '("\\`\\`*Package Review Diff;"
    (display-buffer-full-frame)))
```

1.3.3 Add my elisp directory and other files

Sometimes I load files outside the package system. As long as they're in a directory in my `load-path`, Emacs can find them.

```
(add-to-list 'load-path "~/elisp")
(setq use-package-verbose t)
(setq use-package-always-ensure t)
(require 'use-package)
(use-package quelpa)
(use-package quelpa-use-package)
(quelpa-use-package-activate-advice)
(setq load-prefer-newer t)
```

1.4 Personal information

```
(setq user-full-name "Sacha Chua"
      user-mail-address "sacha@sachachua.com")
```

1.5 System information

```
(defvar sacha-laptop-p (or (equal (system-name) "sacha-x230") (equal (system-name) ↵
  ↵ "sacha-p52")))
(defvar sacha-server-p (and (equal (system-name) "localhost") (equal user-login-name "sacha")))
(defvar sacha-phone-p (not (null (getenv "ANDROID_ROOT")))
  "If non-nil, GNU Emacs is running on Termux.")
(when sacha-phone-p (setg gnutls-algorithm-priority "NORMAL:-VERS-TLS1.3"))
(global-auto-revert-mode) ; simplifies syncing
```

1.6 Reload

```
;; ###autoload
(defun sacha-reload-emacs-configuration ()
  (interactive)
  (load-file "~/proj/.emacs.d/Sacha.el"))
```

1.7 Backups

This is one of the things people usually want to change right away. By default, Emacs saves backup files in the current directory. These are the files ending in `~` that are cluttering up your directory

lists. The following code stashes them all in `~/.config/emacs/backups`, where I can find them with `C-x C-f` (`find-file`) if I really need to.

```
(setq backup-directory-alist '(("\\env$" . nil)
↔
↔      ("." . "~/config/emacs/backups")))
(with-eval-after-load 'tramp
  (setq tramp-backup-directory-alist nil))
```

Disk space is cheap. Save lots.

```
(setq delete-old-versions -1)
(setq version-control t)
(setq vc-make-backup-files t)
(setq auto-save-file-name-transforms '(("*" "~/config/emacs/auto-save-list/" t)))
```

1.7.1 Obscure Emacs package appreciation: backup-walker

The [Emacs Carnival](#) theme for September is [obscure packages](#), which made me think of how the [backup-walker](#) package saved me from having to write some code all over again. Something went wrong when I was editing my config in Org Mode. I probably accidentally deleted a subtree due to over-enthusiastic speed commands. (... Maybe I should make my `k` shortcut for [sacha-org-cut-subtree-or-list-item](#) (p. 195) only work in my Inbox.org, posts.org, and news.org files.) Chunks of my [literate Emacs configuration](#) were gone, including the code that defined [sacha-org-insert-link-dwim](#) (p. 314). Before I noticed, I'd already exported my (now slightly shorter) Emacs configuration file with [org-babel-tangle](#) and restarted Emacs. I couldn't recover the definition from memory using [symbol-function](#) (p. 245). I couldn't use [vundo](#) to browse the Emacs undo tree. As usual, I'd been neglecting to commit my config changes to Git, so I couldn't restore a previous version. Oops.

Well, not the first time I've needed to rewrite code from scratch because of a brain hiccup. I started to reimplement the function. Then I remembered that I had other backups. I have a 2 TB SSD in my laptop, and I had configured Emacs to neatly save numbered backups in a separate directory, keeping all the versions without deleting any of the old ones.

```
(setq backup-directory-alist '(("\\env$" . nil)
↔
↔      ("." . "~/config/emacs/backups")))
(with-eval-after-load 'tramp
  (setq tramp-backup-directory-alist nil))
(setq delete-old-versions -1)
(setq version-control t)
(setq auto-save-file-name-transforms '(("*" "~/config/emacs/auto-save-list/" t)))
```

At the moment, there are about 12,633 files adding up to 3 GB. Totally worth it for peace of mind. I could probably use `grep` to search for the function, but it wasn't easy to see what changed between versions.

I had learned about [backup-walker](#) in the process of writing about [Thinking about time travel with the Emacs text editor, Org Mode, and backups](#). So I used [backup-walker](#) to flip through my file's numbered backups in much the same way that [git-timemachine](#) lets you flip through Git versions of a file. After `M-x backup-walker-start`, I tapped `p` to go through the previous backups. The diff it showed me made it easy to check with `C-s` (`isearch-forward`) if this was the version I was looking for. When I found the change, I pressed RET to load the version with the function in it. Once I found it, it was easy to restore that section. I also restored a couple of other sections that I'd accidentally

deleted too, like the custom [plain text publishing backend](#) (p. 310) I use to export Emacs News with less punctuation. It took maybe 5 minutes to figure this out. Hooray for backup-walker!

Note that the backup-walker diff was the other way around from what I expected. It goes "diff new old" instead of "diff old new", so the green regions marked with + indicate stuff that was removed by the newer version (compared to the one a little older than it) and the red regions marked with - indicate stuff that was added. This could be useful if you think backwards in time, kind of like the [Emacs Antinews](#) file, but my mind doesn't quite work that way. I wanted it to look like a regular diff, with the additions in newer versions marked with +. Emacs being Emacs, I changed it. Here's an example showing what it looks like now:

```
{ { -307~ - -306~ } } <p> -305~ <n> -307~ <return> open -306~
diff -u /home/sacha/.config/emacs/backups/!home!sacha!sync!emacs!Sacha.org.\-306~ /home/sacha/.c
onfig/emacs/backups/!home!sacha!sync!emacs!Sacha.org.\-307~
--- /home/sacha/.config/emacs/backups/!home!sacha!sync!emacs!Sacha.org.\-306~ 2025-09-10 12:47:15.8506
56284 -0400
+++ /home/sacha/.config/emacs/backups/!home!sacha!sync!emacs!Sacha.org.\-307~ 2025-09-15 12:26:54.2806
53047 -0400
@@ -5933,6 +5933,7 @@
      ("C-M-<return>" . org-insert-subheading))
      :custom
      (org-export-with-sub-superscripts nil)
+ (org-footnote-section nil)
      (org-fold-catch-invisible-edits 'smart))
    #+end_src
    ** Writing about sketches and including their text
@@ -9124,7 +9125,27 @@
    #+begin_src emacs-lisp
    (use-package org-contacts
      :config
- (setf org-contacts-files '("~/sync/orgzly/people.org")))
+ (setf org-contacts-files '("~/sync/orgzly/people.org")))
+ :hook
+ (message-setup-hook . 'my-message-greet-contacts))
+
+ (defun my-message-greet-contacts ()
+   (interactive)
+   (let* ((emails
+          (mapcar 'car
+                  (append
+                    (mail-header-parse-addresses (message-fetch-field "To"))
+                    (mail-header-parse-addresses (message-fetch-field "Cc")))))
+          (people
+            (seq-keep
+              (lambda (email)
+                (cdr (assoc-string "NAME_SHORT"
+                                  (caddr (car (org-contacts-filter nil nil (cons "EMAIL" email)))))))
+            emails)))
+   (when people
+     (message-goto-body)
+     (insert "Hi, " (string-join people ",") "!\\n\\n"))))
    #+end_src
    ** Inserting code
    :PROPERTIES:
@@ -9149,7 +9170,6 @@
      (if (org-in-src-block-p)
          (insert definition)
          (insert "#+begin_src emacs-lisp\\n" definition "#+end_src\\n"))))
-
- (defun my-org-insert-function-and-key (keys)
-   (interactive (caar (help--read-key-sequence)))
-   (insert (format "%s= (%s)" (symbol-name (key-binding keys) t))))
@@ -10100,0 +10100,17 @@
U:%%- *Walking: Sacha.org* Top (1,0) ({Diff backup walker} wb WK) 1:46PM 1.58
```

Figure 1: backup-walker diffs going the direction I want them to: additions (+) marked in green, deletions (-) in red

The following code makes it behave the way I expect:

```
(defvar backup-walker-data-alist)
(declare-function diff-no-select "diff" (old new &optional switches noasync bufname))
(declare-function backup-walker-get-version "backup-walker")
;; ###autoload
```

```

(defun sacha-backup-walker-refresh ()
  (let* ((index (cdr (assq :index backup-walker-data-alist)))
         (suffixes (cdr (assq :backup-suffix-list backup-walker-data-alist)))
         (prefix (cdr (assq :backup-prefix backup-walker-data-alist)))
         (right-file (concat prefix (nth index suffixes)))
         (right-version (format "%i" (backup-walker-get-version right-file)))
         diff-buf left-file left-version)
    (if (eq index 0)
        (setq left-file (cdr (assq :original-file backup-walker-data-alist))
              left-version "orig")
        (setq left-file (concat prefix (nth (1- index) suffixes))
              left-version (format "%i" (backup-walker-get-version left-file))))
    ;; we change this to go the other way here
    (setq diff-buf (diff-no-select right-file left-file nil 'noasync))
    (setq buffer-read-only nil)
    (delete-region (point-min) (point-max))
    (insert-buffer-substring diff-buf)
    (set-buffer-modified-p nil)
    (setq buffer-read-only t)
    (force-mode-line-update)
    (setq header-line-format
      (concat (format "{{ ~%S~ → ~%S~ }}"
                    (propertize left-version 'face 'font-lock-variable-name-face)
                    (propertize right-version 'face 'font-lock-variable-name-face))
              (if (nth (1+ index) suffixes)
                  (concat (propertize "<p>" 'face 'italic)
                          " ~"
                          (propertize (int-to-string
                                        (backup-walker-get-version (nth (1+ index) suffixes)))
                                        'face 'font-lock-keyword-face)
                          "~ ")
                  "")
              (if (eq index 0)
                  ""
                  (concat (propertize "<n>" 'face 'italic)
                          " ~"
                          (propertize (int-to-string (backup-walker-get-version (nth (1- ↵
↵ index) suffixes)))
                          'face 'font-lock-keyword-face)
                          "~ ")
                  (propertize "<return>" 'face 'italic)
                  " open ~"
                  (propertize (propertize (int-to-string (backup-walker-get-version right-file))
                                        'face 'font-lock-keyword-face)
                              "~"))
              (kill-buffer diff-buf)))

```

```

(with-eval-after-load 'backup-walker
  (advice-add 'backup-walker-refresh :override #'sacha-backup-walker-refresh))

```

`backup-walker` is not actually a real package in the sense of `M-x package-install`, but fortunately, recent Emacs makes it easier to install from a repository. I needed to install it from <https://github.com/lewang/backup-walker>. It was written so long ago that I needed to `defalias` some functions that were removed in Emacs 26.1. Here's the use-package snippet from my configuration:

```

(use-package backup-walker
  :vc (:url "https://github.com/lewang/backup-walker")

```



```

:commands backup-walker-start
:init
(defalias 'string-to-int 'string-to-number) ; removed in 26.1
(defalias 'display-buffer-other-window 'display-buffer))

```

So there's an obscure package recommendation: [backup-walker](#). It hasn't been updated for more than a decade, and it's not even installable the regular way, but it's still handy.

I can imagine all sorts of ways this workflow could be even better. It might be nice to dust off backup-walker off, switch out the obsolete functions, add an option for the diff direction, and maybe sort things out so that you can reverse the diff, split hunks, and apply hunks to your original file. And maybe a way to walk the backup history for changes in a specific region? I suppose someone could make a spiffy [Transient](#)-based user interface to modernize it. But it's fine, it works. Maybe there's a more modern equivalent, but I didn't see anything in a quick search of `M-x list-packages / N (package-menu-filter-by-name-or-description)` for "backup~", except maybe [vc-backup](#).¹ Is there a general-purpose VC equivalent to git-timemachine? That might be useful.

I should really be saving things in proper version control, but this was a good backup. That reminds me: I should backup my backup backups. I had initially excluded my `~/.config` directory from [borgbackup](#) because of the extra bits and bobs that I wouldn't need when restoring from backup (like all the Emacs packages I'd just re-download). But my file backups... Yeah, that's worth it. I changed my `--exclude-from` to `--patterns-from` and changing my [borg-patterns](#) file to look like this:

```

+ /home/sacha/.config/emacs/backups
- /home/sacha/.config/*
# ... other rules

```

May backup-walker save you from a future oops!

1.8 History

From <http://www.wisdomandwonder.com/wp-content/uploads/2014/03/C3F.html> and <https://emacsredux.com/blog/2026/04/07/stealing-from-the-best-emacs-configs/>

```

(setq savehist-file "~/.config/emacs/savehist")
(savehist-mode 1)
(setq history-length t)
(setq history-delete-duplicates t)
(setq savehist-save-minibuffer-history 1)
(setq savehist-additional-variables
  '(kill-ring
    search-ring
    sacha-stream-number
    regexp-search-ring))
(add-hook 'savehist-save-hook
  (lambda ()
    (setq kill-ring
      (mapcar #'substring-no-properties
        (cl-remove-if-not #'stringp kill-ring)))))

```

¹vc-backup: The [original repo](#) is missing, but you can read it via [ELPA's copy](#). Update: It's over on [Codeberg](#) now, and presumably the info on ELPA will be updated soon.

1.9 Disabling the toolbar

When you're starting out, the tool bar can be very helpful. ([Emacs Basics: Using the Mouse](#)). Eventually, you may want to reclaim that extra little bit of screenspace. The following code turns that thing off. (Although I changed my mind about the menu - I want that again.)

```
(tool-bar-mode -1)
```

1.10 Change "yes or no" to "y or n"

Lazy people like me never want to type "yes" when "y" will suffice.

<https://emacsredux.com/blog/2026/03/15/use-short-answers/>

```
(setopt use-short-answers t)
```

1.11 Minibuffer editing - more space!

Sometimes you want to be able to do fancy things with the text that you're entering into the minibuffer. Sometimes you just want to be able to read it, especially when it comes to lots of text. This binds **C-M-e** in a minibuffer) so that you can edit the contents of the minibuffer before submitting it.

```
(use-package miniedit
  :commands minibuffer-edit
  :init (miniedit-install))
```

1.12 Killing text

```
(setq kill-ring-max 1000)
```

From <https://github.com/itsjeyd/emacs-config/blob/emacs24/init.el>

```
;;;###autoload
(defun sacha-kill-single-line-if-region-is-inactive (beg end &optional region)
  "Wrap around `kill-region' so that we kill a single line."
  (interactive (progn
    (let ((beg (mark kill-region-dwim))
          (end (point)))
      (cond
        ((and kill-region-dwim (not (use-region-p)))
         (list beg end kill-region-dwim))
        ((not (and beg end))
         (user-error "The mark is not set now, so there is no region"))
        ((list beg end 'region))))))
  (if (or (region-active-p)
          (derived-mode-p 'minibuffer-mode))
      (kill-region beg end region)
      (kill-region
       (line-beginning-position)
       (line-beginning-position 2)))
  (ert-deftest sacha-kill-single-line-if-region-is-inactive (
```

```
"Tests `sacha-kill-single-line-if-region-is-inactive'."
(should
 (equal
  (with-temp-buffer
   (insert "Hello there\nWorld\n")
   (goto-char (point-min))
   (sacha-kill-single-line-if-region-is-inactive nil nil)
   (setq text (buffer-string)))
  "World\n")))
```

If I use `M-w` (`kill-ring-save`) without a region, I usually mean the current symbol.

```
;;###autoload
(defun sacha-copy-symbol-if-region-is-inactive (beg end &optional region)
  "Wrap around `kill-ring-save' so that we kill a single line."
  (interactive (list (mark) (point) 'region))
  (if (region-active-p)
      (kill-ring-save beg end region)
      (let ((bounds (or (bounds-of-thing-at-point 'symbol)
                        (bounds-of-thing-at-point 'word))))
        (kill-new (filter-buffer-substring (car bounds) (cdr bounds))))))
```

```
(keymap-global-set "M-w" #'sacha-copy-symbol-if-region-is-inactive)
```

2 Keybindings

```
(repeat-mode 1)
```

2.1 Embark

embark

```
(use-package embark
  :after org
  :load-path "~/vendor/embark"
  :config
  (setq embark-prompter 'embark-keymap-prompter)
  (add-to-list 'embark-target-finders 'sacha-embark-org-element)
  (add-to-list 'embark-target-finders 'sacha-embark-subed-timestamp)
  (add-to-list 'embark-target-injection-hooks '(sacha-journal-post embark--allow-edit))
  (with-eval-after-load 'subed
    (defvar-keymap embark-subed-timestamp-actions
      :doc "Subed timestamp actions"
      :parent subed-mode-map
      "." #'sacha-subed-set-timestamp-to-mpv-position
      "w" #'sacha-subed-copy-timestamp-dwim
      "<up>" #'sacha-subed-adjust-timestamp/sacha-subed-adjust-timestamp-up
      "f" #'sacha-waveform-subed-show-after-time
      "<down>" #'sacha-subed-adjust-timestamp/sacha-subed-adjust-timestamp-down))
    (defvar-keymap embark-sketch-actions
      :doc "Org Mode sketch-related actions"
      :parent org-mode-map
      "o" #'sacha-sketch-insert-file-as-link
      "i" #'sacha-sketch-insert-file-as-link
      "v" #'sacha-geeqie-view))
```

```

(defvar-keymap embark-journal-actions
  :doc "Journal"
  "e" #'sacha-journal-edit)
(add-to-list 'embark-keymap-alist '(sketch . embark-sketch-actions))
(add-to-list 'embark-keymap-alist '(subed-timestamp . embark-subed-timestamp-actions))
(add-to-list 'embark-keymap-alist '(journal . embark-journal-actions))
:bind
(("C-." . embark-act)
 :map vertico-map
(("M-e" . embark-export))
:map minibuffer-local-map
(("C-c e" . embark-act)
 ("M-e" . embark-export)
 ("C-;" . embark-act)
 ("C-<tab>" . embark-select)
 ("C-S-<tab>" . (lambda () (interactive) (embark-select) (vertico-next))))
:map embark-collect-mode-map
(("C-c e" . embark-act)
 ("C-;" . embark-act)
 ("C-<tab>" . embark-select))
:map embark-general-map
(("j" . sacha-journal-post)
 ("m" . sacha-stream-message)
 ("M-w" . (lambda (s) (interactive "MString: ") (kill-new s))))
:map embark-symbol-map
("r" . erefactor-rename-symbol-in-buffer)
:map embark-url-map
("c" . sacha-caption-show)
))
(with-eval-after-load 'embark-org
  (define-key embark-org-src-block-map
    "i" #'sacha-org-fix-block-indentation))

```

This allows me to use C-h with completion in general.

```
(setq prefix-help-command 'embark-prefix-help-command)
```

Things I'm getting used to using:

- C-. c on an Org Mode source block to copy the contents

In addition to the entries below, I also have Embark-related code in other places in my config:

- [Insert a link to an Org Mode heading from an org-refile prompt](#) (p. 191)
- [consult-omni](#) (p. 52)
- [Quickly jump to positions](#) (p. 71)
- [Making it easier to add a category to a blog post](#) (p. 274)
- [embark-11ty](#) (p. 281)
- [Using an Emacs Lisp macro to define quick custom Org Mode links to project files; plus URLs and search](#) (p. 332) (2024)
- [Copy web link](#) (p. 334)
- [Quickly search my code](#) (p. 337)
- [Tip from Omar: embark-around-action-hooks](#) (p. 338)

- [Edit list](#) (p. 512)
- [Running the current Org Mode Babel Javascript block from Emacs using Spookfox](#) (p. 590) (2024)
- [Act on current message with Embark](#) (p. 608)

2.1.1 Using Embark and qrencode to show a QR code for the Org Mode link at point emacs:org

[2024-01-12 Fri]: Added some code to display the QR code on the right side.

John Kitchen includes [little QR codes in his videos](#). I thought that was a neat touch that makes it easier for people to jump to a link while they're watching. I'd like to make it easier to show QR codes too. The following code lets me show a QR code for the Org link at point. Since many of my links use custom Org link types that aren't that useful for people to scan, the code reuses the link resolution code from [web-link](#) (p. 334) so that I can get the regular `https:` link.

```
;; ##autoload
(defun sacha-org-link-qr (url)
  "Display a QR code for URL in a buffer."
  (let ((buf (save-window-excursion (qrencode--encode-to-buffer ←
    ↪ (sacha-org-stored-link-as-url url))))
    (if (> (frame-width) 80)
        (display-buffer-in-side-window buf '((side . right)))
        (display-buffer buf))))
```

```
(use-package qrencode
  :defer t
  :commands qrencode--encode-to-buffer
  :config
  (with-eval-after-load 'embark-org
    (define-key embark-org-link-map (kbd "q") #'sacha-org-link-qr)))
```

while they're watching. I'd like to make it easier to show QR codes too. The following code lets me show a QR code for the Org link at point. Since many of my links use custom Org link types that aren't that useful for people to scan, the code reuses the link resolution code from [dotemacs:web-link](#) so that I can get the regular `https:` link.

```
#+begin_src emacs-lisp
(defun my-org-link-qr (url)
  "Display a QR code for URL in a buffer."
```

U : *- Sacha.org



Encoded Text:
https://sachachua.com/dotemacs#web-link

U : %- *QRCode* <left> left-char

Figure 2: Screenshot of QR code for the link at point

2.1.2 TODO Using Embark to act on video

```
;;;###autoload
(defun sacha-embark-video ()
  "Match video."
  (let ((extensions "youtu\\.?.be\\|\\(webm\\|mp4\\|flv\\)$"))
    (or (if-let* ((link (and (derived-mode-p 'org-mode)
                              (org-element-context)))
                  (when (eq (org-element-type link) 'link)
                        (cond
                         ((string-match extensions
                                          (org-element-property :path link))
                          (cons 'video
                                (string-match extensions (url-get-url-at-point))
                                (cons 'video (url-get-url-at-point)))
                         (and (derived-mode-p 'dired-mode)
                              (string-match extensions (dired-get-filename))
                              (cons 'video (dired-get-filename))))))
        (org-element-property :raw-link link))))))

;;;###autoload
(defun sacha-embark-video-edit-youtube (url)
  "Go to the YouTube edit page for VIDEO."
  (interactive "MURL: ")
  (when (or (string-match "youtu\\.?.be\\|\\(?:\\.com\\)?/\\(?:live/\\|watch/\\)?\\([-A-Za-z0-9_]+\\)" url)
            (string-match "watch\\?v=\\([-A-Za-z0-9_]+\\)" url))
    (browse-url (format "https://studio.youtube.com/video/%s/edit" (match-string 1 url)))))
```

```
(with-eval-after-load 'embark
  (add-to-list 'embark-target-finders 'sacha-embark-video)
  (defvar-keymap sacha-embark-video-actions
    :doc "video"
    "d" #'sacha-deepgram-recognize-audio
    "$" #'sacha-deepgram-cost
    "m" #'mpv-play
    "w" #'embark-copy-as-kill
    "c" #'sacha-caption-show
    "e" ↵
    "t" #'sacha-audio-text
    "W" #'waveform-show)
  (add-to-list 'embark-keymap-alist '(video . sacha-embark-video-actions)))
```

2.1.3 Using Embark to act on audio

```
;;;###autoload
(defun sacha-embark-audio ()
  "Match audio."
  (let ((extensions "m4a\\|mp3\\|wav\\|ogg\\|opus"))
    (if-let* ((link (and (derived-mode-p 'org-mode)
                          (org-element-context)))
              (when (eq (org-element-type link) 'link)
                    (cond
```

```

        ((string-match extensions (org-element-property :path link))
         (cons 'audio (org-element-property :path link))))
      (when (and (derived-mode-p 'dired-mode)
                 (string-match extensions (dired-get-filename)))
        (cons 'audio (dired-get-filename)))))

;;;###autoload
(defun sacha-audio-text (file &optional insert)
  "Get the text for FILE audio.
  If called interactively, copy to the kill ring."
  (interactive (list (read-file-name "Audio: ")))
  (let (text)
    (cond
     ((file-exists-p (concat (file-name-sans-extension file) ".txt"))
      (with-temp-buffer
        (insert-file-contents (concat (file-name-sans-extension file) ".txt"))
        (setq text (buffer-string))))
     ;; no txt yet, is there a vtt?
     ((file-exists-p (concat (file-name-sans-extension file) ".vtt"))
      (setq text (subed-subtitle-list-text
                  (subed-parse-file (concat (file-name-sans-extension file) ".vtt")))))
     ;; no VTT, let's recognize it
     (t
      (sacha-deepgram-recognize-audio file)
      (when (file-exists-p (concat (file-name-sans-extension file) ".vtt"))
        (setq text (subed-subtitle-list-text
                    (subed-parse-file (concat (file-name-sans-extension file) ".vtt"))))))))
    (when text
      (when (called-interactively-p 'any)
        (if insert
          (insert text "\n")
          (kill-new text)))
      text)))

;;;###autoload
(defun sacha-open-in-audacity (file)
  (interactive "FFile: ")
  (start-process "audacity" nil "audacity" file))

```

```

(with-eval-after-load 'embark
  (add-to-list 'embark-target-finders 'sacha-embark-audio)
  (defvar-keymap sacha-embark-audio-actions
    :doc "audio"
    "a" #'sacha-open-in-audacity
    "d" #'sacha-deepgram-recognize-audio
    "$" #'sacha-deepgram-cost
    "D" #'sacha-audio-braindump-reprocess
    "m" #'mpv-play
    "w" #'sacha-audio-text
    "W" #'waveform-show)
  (add-to-list 'embark-keymap-alist '(audio . sacha-embark-audio-actions)))

```

2.1.4 Using Embark to insert files as Org INCLUDEs

```

;;;###autoload
(defun sacha-insert-file-as-org-include (file)

```

```

(interactive "fFile: ")
(set-text-properties 0 (length file) nil file)
(let ((mode (assoc-default file auto-mode-alist 'string-match)))
  (insert
   (org-link-make-string (concat "file:" file) (concat "Download " (file-name-nondirectory file)
   ↪ file))) "\n"
   "#+begin_my_details " (file-name-nondirectory file) "\n"
   (format "#+INCLUDE: %s" (prin1-to-string file))
   (if mode
    (concat " src " (replace-regexp-in-string "-mode$" "" (symbol-name mode)))
    "")
   "\n"
   "#+end_my_details\n"))

;;;###autoload
(defun sacha-transform-org-link-to-include ()
  (interactive)
  (let ((link (org-element-lineage (org-element-context) '(link) t))
        (mode (assoc-default (org-element-property :path link) auto-mode-alist 'string-match)))
    (when link
      (delete-region (org-element-property :begin link)
                     (org-element-property :end link))
      (sacha-insert-file-as-org-include (org-element-property :path link)))))

(with-eval-after-load 'embark
  (define-key embark-file-map "O" #'sacha-insert-file-as-org-include))

```

2.15 Using Embark to offer context-sensitive actions for Org elements

```

;;;###autoload
(defun sacha-embark-org-element ()
  "Target an Org Mode element at point."
  (save-window-excursion
    (save-excursion
      (save-restriction
        (when (derived-mode-p 'org-agenda-mode)
          (org-goto-marker-or-bmk (org-get-at-bol 'org-marker))
          (org-back-to-heading))
        (when (derived-mode-p 'org-mode)
          (let* ((context ;; Borrowed from org-open-at-point
                  ;; Only consider supported types, even if they are not the
                  ;; closest one.
                  (org-element-lineage (org-element-context)
                                       '(headline src-block link) t))
                 (type (org-element-type context))
                 (value (org-element-property :value context)))
            (cond ((eq type 'headline)
                   (cons 'org-heading (org-element-property :title context)))
                  ;; src-block and link can be handled by embark-org
                  (t)
                  )))
          )))

;;;###autoload
(defun sacha-embark-org-src-block-copy-noweb-reference (element)
  (kill-new (if (org-element-property element :parameters)
                (format "<<%s(%s)>>" (org-element-property element :name)
                        (org-element-property element :parameters))
                (org-element-property element :text))))

```



```
(format "<<%s>>" (org-element-property element :parameters))))))
```

```
(with-eval-after-load 'embark-org  
  (keymap-set embark-org-src-block-map "N" #'sacha-embark-org-src-block-copy-noweb-reference))
```

2.1.6 Whichkey and Embark

From <https://github.com/oantolin/embark/wiki/Additional-Configuration#use-which-key-like-a-key-menu-prompt>

```
;; ###autoload  
(defun embark-which-key-indicator ()  
  "An embark indicator that displays keymaps using which-key.  
  The which-key help message will show the type and value of the  
  current target followed by an ellipsis if there are further  
  targets."  
  (lambda (&optional keymap targets prefix)  
    (if (null keymap)  
        (which-key--hide-popup-ignore-command)  
        (which-key--show-keymap  
         (if (eq (plist-get (car targets) :type) 'embark-become)  
             "Become"  
             (format "Act on %s '%s'%s"  
                     (plist-get (car targets) :type)  
                     (embark--truncate-target (plist-get (car targets) :target))  
                     (if (cdr targets) "... " "")))  
         (if prefix  
             (pcase (lookup-key keymap prefix 'accept-default)  
               ((and (pred keymap) km) km)  
               (_ (key-binding prefix 'accept-default)))  
             keymap)  
        nil nil t (lambda (binding)  
                    (not (string-suffix-p "-argument" (cdr binding))))))))  
  
;; ###autoload  
(defun embark-hide-which-key-indicator (fn &rest args)  
  "Hide the which-key indicator immediately when using the completing-read prompter."  
  (which-key--hide-popup-ignore-command)  
  (let ((embark-indicators  
        (remq #'embark-which-key-indicator embark-indicators)))  
    (apply fn args)))
```

```
(setq embark-indicators  
      '(embark-which-key-indicator  
        embark-highlight-indicator  
        embark-isearch-highlight-indicator))  
  
(with-eval-after-load 'embark  
  (advice-add #'embark-completing-read-prompter  
    :around #'embark-hide-which-key-indicator))
```

DONE Changing the which-key labels for shortcuts

EMBARK It's hard to remember keyboard shortcuts, so I'm glad that Emacs 30 now includes which-key by default. You still need to enable `which-key-mode`, but once you do, if you pause in the middle of a keyboard combination like `C-c ...`, you'll see a menu of shortcuts.

I've been exploring more context menus with [Embank](#). Using the [suggested configuration from Embark's wiki](#), I have it set up so that I can use `C-.` (`embark-act`) to open or cycle through different context-sensitive keymaps, with a `which-key` display when I pause, and completion on the commands in the keymap if I use `C-h`.

I want to make the `which-key` menu a little easier to scan: shorter function names, more logical grouping, and less visual noise. I think I'm on the right track with these `which-key` tweaks.

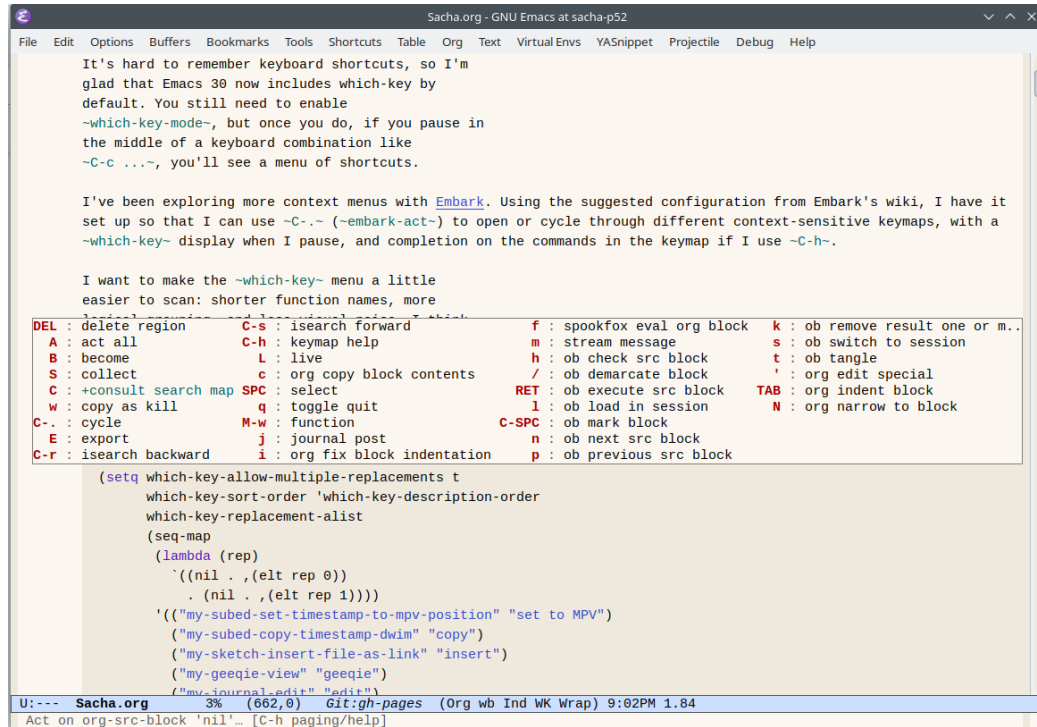


Figure 3: Screenshot of `which-key` with some tweaks

```
(with-eval-after-load 'which-key
  (setopt which-key-allow-multiple-replacements t
          which-key-sort-order 'which-key-description-order
          which-key-replacement-alist
          (seq-map
            (lambda (rep)
              `((nil . ,(elt rep 0))
                . (nil . ,(elt rep 1))))
            '(("sacha-subed-set-timestamp-to-mpv-position" "set to MPV")
              ("sacha-subed-copy-timestamp-dwim" "copy")
              ("my-sketch-insert-file-as-link" "insert")
              ("my-geeie-view" "geeie")
              ("my-journal-edit" "edit")
              ("sacha-org-link-qr" "qr")
              ("sacha-image-open-in-" "")
              ("org-babel-" "ob-")
              ("next" "")
              ("previous" "")
              ("sacha-image-" "")
              ("sacha-embark-org-blog-" ""))
            (↪ exported URL"))
```

```
( "embark-collect" " " collect")
( "sacha-embark-org- " " ")
( "sacha- " " ")
( "embark- " " ")
( "embark-act-all" "all")
( "embark-become" "become")
( "embark-collect" "collect")
( "- " " " ))))
```

Changing `which-key-sort-order` to sort by description seems to do a reasonable job of grouping things somewhat logically. The sorting happens before replacement, so the result might not look sorted once my replacements are done. If I want to tweak it further, I could `defalias` commands and then put them into the keymaps with replacements, or maybe look into overriding `which-key--get-bindings`. Also, these replacements are not used by `embark-completing-read-prompter`, but maybe I can wrap some advice around `embark--formatted-bindings` in order to change that.

2.1.7 Embark and images image

image

[2024-09-25 Wed]: added attachment handling

```
;;###autoload
(defun sacha-sketch-insert-file-as-link (f)
  (interactive (list (sacha-complete-sketch-filename)))
  (cond
   ((derived-mode-p 'org-mode)
    (insert (org-link-make-string (concat "sketch:" (file-name-nondirectory f))) "\n"))
   ((or (derived-mode-p 'html-mode) (derived-mode-p 'web-mode))
    (insert "{% sketchFull \"" (file-name-base f) "\" %}"))
   (t (insert f))))
```

```
;;###autoload
(defun sach-embark-image ()
  "Match images."
  (let ((extensions "\\(png\\|jpg\\|svg\\|gif\\|jpeg\\|\\)"))
    (cond
      ((derived-mode-p 'org-mode)
       (when-let* ((link (org-element-context)))
         (when (eq (org-element-type link) 'link)
          (cond
            ((string= "attachment" (org-element-property :type link))
             (cons 'image (expand-file-name (org-element-property :path link)
                                              (org-attach-dir))))
            ((string-match "sketch" (org-element-property :type link))
             (cons 'image (sach-get-sketch-filename (org-element-property :path link))))
            ((string-match extensions (org-element-property :path link))
             (cons 'image (org-element-property :path link))))))
      ((and (derived-mode-p 'dired-mode)
            (string-match extensions (dired-get-filename)))
       (cons 'image (dired-get-filename)))
      ((derived-mode-p 'subed-mode)
       (when-let* ((filename (thing-at-point 'filename)))
         (when (string-match (concat "file:\\(\\.+\\.\\)" extensions "\\)") filename)
          (cons 'image (match-string 1 filename))))
      ((and (buffer-file-name)
```

```

      (string-match extensions (buffer-file-name)))
    (cons 'image (buffer-file-name))))))
  (with-eval-after-load 'embark
    (add-to-list 'embark-target-finders 'sacha-embark-image))

```

I want to:

- open images in an annotation program, maybe [com.github.phase1geo.annotator](https://github.com/phase1geo/annotator)
- open images in [Krita](https://krita.org/)
- replace with latest screenshot
- copy text to kill ring
- insert text as details block

```

;;;###autoload
(defun sacha-image-open-in-annotator (file)
  (interactive "FImage: ")
  (start-process "annotator" nil "com.github.phase1geo.annotator" (expand-file-name file)))

;;;###autoload
(defun sacha-image-open-in-krita (file)
  (interactive "FImage: ")
  (start-process "krita" nil "krita" "--nosplash" (expand-file-name file)))

;;;###autoload
(defun sacha-image-open-in-inkscape (file)
  (interactive "FImage: ")
  (start-process "inkscape" nil "inkscape" (expand-file-name file)))

;;;###autoload
(defun sacha-image-open-in-gimp (file)
  (interactive "FImage: ")
  (start-process "gimp" nil "gimp" (expand-file-name file)))

;;;###autoload
(defun sacha-open-in-firefox (file)
  (interactive "FItem: ")
  (start-process "firefox" nil "firefox" (if (string-match "^http" file) file
    ↪ (expand-file-name file))))

(defvar sacha-image-autocrop-border 10)

;;;###autoload
(defun sacha-image-autocrop (filename &optional border)
  (interactive "FFile: ")
  (setq border (or border sacha-image-autocrop-border))
  (let ((args (append '("-trim")
    (if border `("-bordercolor" "#FFFFFF" "-border" ,(number-to-string ↪
    ↪ border))))
    (list "+repage" (expand-file-name filename)))))
  (apply #'call-process "mogrify" nil sacha-debug-buffer nil args)
  filename))

```

2.1.8 Converting handwriting to text

```

;;;###autoload
(defun sacha-image-recognize (file)
  "Returns the text."
  (interactive "FFile: ")
  (if (file-exists-p (concat (file-name-sans-extension file) ".txt"))
      (with-temp-buffer
        (insert-file-contents (concat (file-name-sans-extension file) ".txt"))
        (buffer-string))
      (let* ((data
              (json-parse-string
               (if (file-exists-p (concat (file-name-sans-extension file) ".json"))
                   (with-temp-buffer
                     (insert-file-contents (concat (file-name-sans-extension file) ".json"))
                     (buffer-string))
                   (when (string= (file-name-extension file) "pdf")
                     (setq file (sacha-sketch-convert-pdf file)))
                   (when (string= (file-name-extension file) "svg")
                     (call-process "inkscape" nil sacha-debug-buffer nil "--export-type=png" ↵
                                   "--export-dpi=96" "--export-background-opacity=1" "--pdf-poppler" (expand-file-name ↵
                                   file))))
              (catch 'done
                (dolist (ext '(".png" ".jpg" ".jpeg"))
                  (when (file-exists-p (concat (file-name-sans-extension file) ext))
                    (with-temp-file (concat (file-name-sans-extension file) ".json")
                      (call-process "gcloud" nil t nil "ml" "vision" "detect-document"
                                    (expand-file-name (concat (file-name-sans-extension file) ↵
                                    ext))))
                    (throw 'done (buffer-string))))))
              :object-type 'alist))
        (text
         (if (assoc-default 'responses data)
             (assoc-default 'text (assoc-default 'fullTextAnnotation (elt (assoc-default ↵
             'responses data) 0)))
             (assoc-default 'description (elt (assoc-default 'textAnnotations data) 0))))
         (with-temp-file (concat (file-name-sans-extension file) ".txt")
           (insert text))
         text)))

```

2.1.9 Renaming and storing

```

;;;###autoload
(defun sacha-image-rename-current-image-based-on-id (id)
  (interactive
   (let ((filename (if (derived-mode-p 'image-mode)
                       (buffer-file-name)
                       (dired-get-filename))))
     (list
      (if (string-match "[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]" filename)
          (match-string 0 filename)
          (read-string "ID: "))))
   (let* ((data (sacha-journal-get-by-zidstring id))
          (old-file (if (derived-mode-p 'image-mode)
                       (buffer-file-name)
                       (dired-get-filename))))
     (ext (file-name-extension old-file))
     (new-prefix (concat id " " (plist-get data :Note)))
     (text (plist-get data :Other))

```

```

        new-file)
    (when (and text (not (string= (string-trim text) "")))
      (with-temp-file (concat (file-name-sans-extension old-file) ".txt")
        (insert text)))
    (when (derived-mode-p 'image-mode)
      (kill-buffer))
    (setq new-file
      (sacha-image-store (sacha-rename-file-set old-file
        new-prefix t)
        t))
    (when (derived-mode-p 'image-mode)
      (find-file new-file))
    (find-file (concat (file-name-sans-extension new-file) ".txt"))))

;;;###autoload
(defun sacha-image-recognize-get-new-filename (file)
  (interactive "FFile: ")
  (if-let* ((text (sacha-image-recognize file))
    (id (and (string-match "[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]" text)
      (match-string 0 text)))
    (data (and id (sacha-journal-get-by-zidstring id))))
    (expand-file-name
      (concat id " " (plist-get data :Note) "." (file-name-extension file))
      (file-name-directory file))
    file))

;;;###autoload
(defun sacha-image-recognize-and-rename (file)
  (interactive "FFile: ")
  (let ((new-name (expand-file-name (sacha-image-recognize-get-new-filename file)
    (file-name-directory file))))
    (rename-file file new-name t)
    new-name))

;;;###autoload
(defun sacha-image-tags (file)
  (setq file (file-name-base file))
  (cond
    ((string-match "#\\([^\ ]+\\)" file)
      (let ((start-pos 0))
        (cl-loop for match-pos = (string-match "#\\([^\ \\.]+\\)" file start-pos)
          while match-pos
            collect (match-string 1 file)
            do (setf start-pos (1+ match-pos))))
      ((string-match " -- \\(.+\\)" file)
        (split-string (match-string 1 file) " "))))

;;;###autoload
(defun sacha-image-name-without-tags (file)
  (replace-regexp-in-string " -- \\([^\ \\.]+\\)" (replace-regexp-in-string "#\\([^\ \\.]+\\)" ""
    ↵ file)))

;;;###autoload
(defun sacha-image-rename-set (old-name new-name &optional tags do-copy)
  (when (or (not (string= old-name new-name)) tags)
    (when tags
      (setq new-name (concat (sacha-image-name-without-tags new-name)
        " -- "
        (string-join tags " "))))

```

```

        (file-name-extension new-name))))
  (dolist (file (sacha-file-set old-name))
    (funcall
      (if do-copy
        'copy-file
        'rename-file)
      file
      (concat (file-name-sans-extension new-name) "." (file-name-extension file))
      t)))
  new-name)

;;;###autoload
(defun sacha-image-store (file &optional do-move)
  "Copy or move this image into public or private sketches as needed."
  (interactive (list (if (derived-mode-p 'image-mode)
                        (buffer-file-name)
                        (dired-get-filename))
                    t))
  (unless (string-match "[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]" file)
    (setq file (sacha-image-recognize-and-rename file)))
  (if (string-match "[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]" file)
    (let ((private (member "private" (sacha-image-tags file))))
      (sacha-image-rename-set
        file
        (expand-file-name
          (file-name-nondirectory file)
          (if private
              sacha-private-sketches-directory
              (car sacha-sketch-directories)))))
      file))

;;;###autoload
(defun sacha-image-copy-text (file)
  (interactive "FImage: ")
  (kill-new (sacha-image-recognize file)))

;;;###autoload
(defun sacha-image-insert-text-as-details (file)
  (interactive "FImage: ")
  (when (and (derived-mode-p 'org-mode)
             (eq (org-element-type (org-element-context)) 'link))
    (goto-char (org-element-end (org-element-context))))
  (insert "\n#+begin_my_details\n" (sacha-image-recognize file) "\n#+end_my_details\n"))

;;;###autoload
(defun sacha-image-thumbnail (path)
  (interactive "FImage: ")
  (let* ((filename (expand-file-name (concat "thumb-" (file-name-nondirectory path)
                                           (file-name-directory path))))
        (call-process "convert" nil nil nil path "-thumbnail" "500x" filename)
        (kill-new filename)
        filename))

;;;###autoload
(defun sacha-org-svg-copy-links (filename)
  (interactive (list (read-file-name "SVG: " nil

```

```

nil
(lambda (f)
  (or (string-match "\\.svg$" f)
      (file-directory-p f))))
(let ((dom (car (xml-parse-file filename))))
  (kill-new
   (mapconcat
    (lambda (elem)
      (concat "- " (org-link-make-string
                    (dom-attr elem 'href)
                    (or (dom-attr elem 'title)
                        (dom-text (dom-by-tag elem 'title))))))
    (dom-by-tag dom 'a)
    "\n"))))

```

```

(with-eval-after-load 'embark
  (defvar-keymap sachas-embark-image-actions
    :doc "Images"
    "k" #'sachas-image-open-in-krita
    "a" #'sachas-image-open-in-annotator
    "i" #'sachas-image-open-in-inkscape
    "w" #'sachas-image-copy-text
    "c" #'sachas-image-autocrop
    "]" #'sachas-image-rotate-clockwise
    "[" #'sachas-image-rotate-counterclockwise
    "g" #'sachas-image-open-in-gimp
    "f" #'sachas-open-in-firefox
    "s" #'sachas-image-store
    "r" #'sachas-image-recognize-and-rename
    "t" #'sachas-org-sketch-open-text-file
    "T" #'sachas-image-thumbnail
    "L" #'sachas-org-svg-copy-links
    "C" #'sachas-image-recolor
    "d" #'sachas-image-insert-text-as-details)
    (add-to-list 'embark-keymap-alist '(image . sachas-embark-image-actions)))

```

2.1.10 Embark and subed

```

;; ##autoload
(defun sachas-subed-set-timestamp-to-mpv-position (&optional rest)
  (interactive)
  (skip-chars-backward "0-9:,.")
  (when (looking-at "\\([0-9]+\\):\\([0-9]+\\):\\([0-9]+\\)\\.\\([0-9]+\\)")
    (replace-match (save-match-data (subed-msecs-to-timestamp subed-mpv-playback-position)) t <=> t)))

;; ##autoload
(defun sachas-subed-adjust-timestamp (offset)
  (interactive (list -100))
  (save-excursion
    (skip-chars-backward "0-9:,.")
    (when (looking-at subed-vtt--regexp-timestamp)
      (let ((new-ts (+ (subed-vtt--timestamp-to-msecs (match-string 0)) offset)))
        (replace-match (save-match-data
                        (subed-vtt--msecs-to-timestamp new-ts)))
        (sachas-waveform-subed-show-after-time)))

```



```

new-ts))))

;;;###autoload
(defun sacha-subed-adjust-timestamp-up (offset)
  (interactive (list 100))
  (subed-mpv-jump (sacha-subed-adjust-timestamp (- offset))))

;;;###autoload
(defun sacha-subed-adjust-timestamp-down (offset)
  (interactive (list -100))
  (subed-mpv-jump (sacha-subed-adjust-timestamp (- offset))))

;;;###autoload
(defun sacha-subed-copy-timestamp-from-previous ()
  (interactive)
  (let ((ms (save-excursion (subed-backward-subtitle-time-stop) (subed-subtitle-msecs-stop))))
    (subed-set-subtitle-time-start ms)))

;;;###autoload
(defun sacha-subed-copy-timestamp-to-next ()
  (interactive)
  (let ((ms (subed-subtitle-msecs-stop)))
    (save-excursion
      (subed-forward-subtitle-time-stop) (subed-set-subtitle-time-start ms))))

;;;###autoload
(defun sacha-subed-copy-timestamp-dwim ()
  (interactive)
  (save-excursion
    (skip-chars-backward "0-9:,.")
    (if (bolp)
        (sacha-subed-copy-timestamp-from-previous)
        (sacha-subed-copy-timestamp-to-next))))

```

```

;;;###autoload
(defun sacha-embark-subed-timestamp ()
  (save-excursion
    (skip-chars-backward "0-9:,.")
    (when (looking-at "\\([0-9]+\\):\\([0-9]+\\):\\([0-9]+\\)\\.\\([0-9]+\\)")
      (list 'subed-timestamp
            (propertize
              (match-string 0)
              'ms (compile-media-timestamp-to-msecs (match-string 0))
              'position (if (bolp) 'start 'stop))))))

```

```

(defhydra sacha-subed-adjust-timestamp ()
  ("" sacha-subed-adjust-timestamp-up "Up" :exit nil)
  ("" sacha-subed-adjust-timestamp-down "Down" :exit nil))

```

2.1.11 Embark, symbols, and casual-symbol-overlay

Link: <http://yummymelon.com/devnull/announcing-casual-symbol-overlay.html>

```

(use-package casual-symbol-overlay
  :if sacha-laptop-p
  :after embark
  :init
  (with-eval-after-load 'embark

```

```
(keymap-set embark-symbol-map "z" #'casual-symbol-overlay-tmenu)))
```

2.1.12 Embark and `erefactor-rename-symbol-in-buffer`

`Embark` automatically passes the Embark target as the first value for an interactive prompt, which interferes with `erefactor-rename-symbol-in-buffer`.

```
;;;###autoload
(defun sacha-embark-erefactor-rename-symbol-in-buffer (old-name new-name)
  (interactive (let* ((old-name (read-string "Symbol: "))
                     (new-name (read-string (format "%s -> New name: " old-name)
                                              old-name
                                              'erefactor--read-symbol-history))))
    (list old-name new-name)))
(erefactor-rename-symbol-in-buffer old-name new-name))
```

```
(with-eval-after-load 'embark
  (keymap-set embark-command-map "r" #'sacha-embark-erefactor-rename-symbol-in-buffer)
  (keymap-set embark-symbol-map "r" #'sacha-embark-erefactor-rename-symbol-in-buffer))
```

2.1.13 Embark and Org Mode timestamps

```
(defvar-keymap sacha-embark-org-timestamp-map
  :parent embark-general-map
  :doc "Keymap for Org timestamps."
  "W" #'sacha-org-timestamp-in-time-zones)
(with-eval-after-load 'embark
  (add-to-list 'embark-keymap-alist '(org-timestamp . sacha-embark-org-timestamp-map)))
```

2.2 Extended command list

This code allows me to select a command from a short list of functions so that I can prompt my memory better. I wonder if this makes sense considering transient and hydra make keyboard shortcuts easier.

```
(sacha-execute-extended-command-from-list nil '(org-capture consult-buffer))
```

```
;;; Mostly the same as my/read-extended-command-from-list
;;;###autoload
(defun sacha-read-extended-command-from-list (list)
  "Read command name to invoke in `execute-extended-command'."
  (minibuffer-with-setup-hook
    (lambda ()
      (add-hook 'post-self-insert-hook
        (lambda ()
          (setq execute-extended-command--last-typed
                (minibuffer-contents))))
        nil 'local)
      (setq-local minibuffer-default-add-function
        (lambda ()
          ;; Get a command name at point in the original buffer
          ;; to propose it after M-n.
```

```

        (let ((def (with-current-buffer
                     (window-buffer (minibuffer-selected-window))
                     (and (commandp (function-called-at-point))
                          (format "%S" (function-called-at-point))))))
          (all (sort (minibuffer-default-add-completions)
                     #'string<)))

        (if def
            (cons def (delete def all))
            all))))))

;; Read a string, completing from and restricting to the set of
;; all defined commands. Don't provide any initial input.
;; Save the command read on the extended-command history list.
(completing-read
 (concat (cond
          ((eq current-prefix-arg '-') "- ")
          ((and (consp current-prefix-arg)
                (eq (car current-prefix-arg) 4)) "C-u ")
          ((and (consp current-prefix-arg)
                (integerp (car current-prefix-arg)))
           (format "%d " (car current-prefix-arg)))
          ((integerp current-prefix-arg)
           (format "%d " current-prefix-arg)))
          ;; This isn't strictly correct if `execute-extended-command'
          ;; is bound to anything else (e.g. [menu]).
          ;; It could use (key-description (this-single-command-keys)),
          ;; but actually a prompt other than "M-x" would be confusing,
          ;; because "M-x" is a well-known prompt to read a command
          ;; and it serves as a shorthand for "Extended command: ".
          "M-x ")
  (lambda (string pred action)
    (if (and suggest-key-bindings (eq action 'metadata))
        '(metadata
          (affixation-function . read-extended-command--affixation)
          (category . command))
        (complete-with-action action list string pred)))
    #'commandp t nil 'extended-command-history)))

;;; Mostly the same as execute-extended-command
;;; ###autoload
(defun sachu-execute-extended-command-from-list (prefixarg &optional command-name typed)
  ;; Based on Fexecute_extended_command in keyboard.c of Emacs.
  ;; Aaron S. Hawley <aaron.s.hawley(at)gmail.com> 2009-08-24
  "Read a command name, then read the arguments and call the command.
  To pass a prefix argument to the command you are
  invoking, give a prefix argument to `execute-extended-command'."
  (declare (interactive-only command-execute))
  ;; FIXME: Remember the actual text typed by the user before completion,
  ;; so that we don't later on suggest the same shortening.
  (interactive
   (let ((execute-extended-command--last-typed nil))
     (list current-prefix-arg
           (if (and command-name (listp command-name))
               (sachu-read-extended-command-from-list command-name)
               (read-extended-command))
           execute-extended-command--last-typed)))
  ;; Emacs<24 calling-convention was with a single `prefixarg' argument.
  (when (listp command-name)
    (let ((current-prefix-arg prefixarg) ; for prompt
          (execute-extended-command--last-typed nil))

```

```

    (setq command-name
      (if command-name
        (sacha-read-extended-command-from-list command-name)
        (read-extended-command)))
    (setq typed execute-extended-command--last-typed)))
(let* ((function (and (stringp command-name) (intern-soft command-name)))
      (binding (and suggest-key-bindings
                    (not executing-kbd-macro)
                    (where-is-internal function overriding-local-map t))))
(unless (commandp function)
  (error "%s' is not a valid command name" command-name))
;; Some features, such as novice.el, rely on this-command-keys
;; including M-x COMMAND-NAME RET.
(set--this-command-keys (concat "\M-x" (symbol-name function) "\r"))
(setq this-command function)
;; Normally `real-this-command' should never be changed, but here we really
;; want to pretend that M-x <cmd> RET is nothing more than a "key
;; binding" for <cmd>, so the command the user really wanted to run is
;; `function' and not `execute-extended-command'. The difference is
;; visible in cases such as M-x <cmd> RET and then C-x z (bug#11506).
(setq real-this-command function)
(let ((prefix-arg prefixarg))
  (command-execute function 'record))
;; If enabled, show which key runs this command.
;; But first wait, and skip the message if there is input.
(let* ((waited
      ;; If this command displayed something in the echo area;
      ;; wait a few seconds, then display our suggestion message.
      ;; FIXME: Wait *after* running post-command-hook!
      ;; FIXME: If execute-extended-command--shorter were
      ;; faster, we could compute the result here first too.
      (when (and suggest-key-bindings
                  (or binding
                      (and extended-command-suggest-shorter typed)))
        (sit-for (cond
                  ((zerop (length (current-message))) 0)
                  ((numberp suggest-key-bindings) suggest-key-bindings)
                  (t 2))))))
      (when (and waited (not (consp unread-command-events)))
        (unless (or (not extended-command-suggest-shorter)
                    binding executing-kbd-macro (not (symbolp function))
                    (<= (length (symbol-name function)) 2))
          ;; There's no binding for CMD. Let's try and find the shortest
          ;; string to use in M-x.
          ;; FIXME: Can be slow. Cache it maybe?
          (while-no-input
            (setq binding (execute-extended-command--shorter
                          (symbol-name function) typed)))
          (when binding
            (with-temp-message
              (format-message "You can run the command `%s' with %s"
                             function
                             (if (stringp binding)
                               (concat "M-x " binding " RET")
                               (key-description binding))))
            (sit-for (if (numberp suggest-key-bindings)
                        suggest-key-bindings
                        2)))))))

```

2.3 Menus

Handy when I'm in tablet mode.

```
(define-key-after global-map [menu-bar sacha-menu] (cons "Shortcuts" (make-sparse-keymap ↵
  ↵ "Custom shortcuts"))) 'tools)
(define-key global-map [menu-bar sacha-menu journal] '("Show journal entries" . ↵
  ↵ sacha-show-missing-journal-entries))
(define-key global-map [menu-bar sacha-menu agenda] '("Org agenda" . (lambda () ↵
  ↵ (interactive) (org-agenda nil "a"))))
(define-key global-map [menu-bar sacha-menu audio] '("Process audio" . (lambda () ↵
  ↵ (interactive) (shell-command "~/bin/process-audio &"))))
(define-key global-map [menu-bar sacha-menu new-index-card] '("New index card" . (lambda () ↵
  ↵ (interactive)
  ↵
  ↵ (sacha-org-sketch-edit (sacha-prepare-index-card-template)))))
```

2.4 Context menus

```
(add-hook 'text-mode-hook 'context-menu-mode)
(with-eval-after-load 'dired
  (add-hook 'dired-mode-hook 'context-menu-mode))
(add-hook 'shell-mode-hook 'context-menu-mode)
```

2.5 Repeatable commands

Based on <http://oremacs.com/2015/01/14/repeatable-commands/> . Modified to accept `nil` as the first value if you don't want the keymap to run a command by default, and to use `kbd` for the keybinding definitions.

```
;;;###autoload
(defun sacha-def-rep-command (alist)
  "Return a lambda that calls the first function of ALIST.
  It sets the transient map to all functions of ALIST,
  allowing you to repeat those functions as needed."
  (let ((keymap (make-sparse-keymap))
        (func (car alist)))
    (mapc (lambda (x)
            (when x
              (define-key keymap (kbd (car x)) (cdr x))))
          alist)
    (lambda (arg)
      (interactive "p")
      (when func
        (funcall func arg))
      (set-transient-map keymap t))))
```

2.6 Hydra keyboard shortcuts

`hydra` offers customizable shortcuts. `transient` is another option.

```
(use-package hydra :commands defhydra)
(use-package use-package-hydra)
```

```
(if sacha-laptop-p
  (use-package hydra-posframe
    :defer t
    :if sacha-laptop-p :after hydra
    :vc (:url "https://github.com/Ladicle/hydra-posframe")
  ))
```

```
(with-eval-after-load 'hydra
  (defhydra sacha-window-movement ()
    (<left> windmove-left)
    (<right> windmove-right)
    (<down> windmove-down)
    (<up> windmove-up)
    ("y" other-window "other")
    ("h" switch-window "switch-window")
    ("b" consult-buffer "buffer")
    ("f" find-file "file")
    ("F" find-file-other-window "other file")
    ("v" (progn (split-window-right) (windmove-right)))
    ("o" delete-other-windows :color blue)
    ("a" ace-window)
    ("s" ace-swap-window)
    ("d" delete-window "delete")
    ("D" ace-delete-window "ace delete")
    ("i" ace-maximize-window "maximize")
    ("q" nil)))
```

```
(with-eval-after-load 'hydra
  (defhydra sacha-shortcuts (:exit t)
    ("j" sacha-helm-journal "Journal")
    ("C" sacha-resolve-orgzly-syncthing "Conflicts")
    ("n" sacha-capture-timestamped-note "Note")
    ("c" emacs-news-org-categorize/body "Categorize")
    ("d" emacs-news-check-duplicates "Dupe")
    ("s" save-buffer "Save")
    ("f" sacha-file-shortcuts/body "File shortcut")
    ("+" text-scale-increase "Increase")
    ("-" text-scale-decrease "Decrease")
    ("G" gif-screencast-start-or-stop "GIF screencast")
    ("g" sacha-geeqie/body "Geeqie")
    ("r" sacha-record-ffmpeg-toggle-recording "Record screen")
    ("l" (sacha-toggle-or-create "*scratch*" (lambda () (switch-to-buffer ↵
    ↵ (startup--get-buffer-create-scratch)))) "Lisp")
    ("e" eshell-toggle "Eshell")
    ("w" sacha-engine-dmode-hydra/body "Search web")
    ("E" sacha-emacs-news/body "Emacs News"))
    (keymap-global-set "<f5>" #'sacha-shortcuts/body)
  (defhydra sacha-emacs-news (:exit t)
    "Emacs News"
    ("f" (find-file "~/sync/emacs-news/index.org") "News")
    ("C" (find-file "~/proj/emacs-calendar/README.org") "Calendar")
    ("C" (find-file "/ssh:web:/var/www/emacslife.com/calendar/README.org" "Calendar on ↵
    ↵ server)))
    ("d" emacs-news-check-duplicates "Dupe")
    ("c" emacs-news-org-categorize/body "Categorize")
    ("h" (sacha-org-update-link-description "HN") "Link HN")
    ("i" (sacha-org-update-link-description "Irreal") "Link Irreal"))
```

```
( "m" sacha-share-emacs-news "Mail")
( "t" (browse-url "https://tweetdeck.twitter.com") "Twitter"))))
```

```
;;###autoload
(defun sacha-org-update-link-description (description)
  "Update the current link's DESCRIPTION."
  (interactive "MDescription: ")
  (let (link)
    (save-excursion
      (cond
        ((org-in-regexp org-link-bracket-re 1)
         (setq link (org-link-unescape (match-string-no-properties 1)))
         (delete-region (match-beginning 0) (match-end 0))
         (insert (org-link-make-string link description))
         (sit-for 0))
        ((or (org-in-regexp org-link-angle-re)
              (org-in-regexp org-link-plain-re))
         (setq link (org-unbracket-string "<" ">" (match-string 0)))
         (delete-region (match-beginning 0) (match-end 0))
         (insert (org-link-make-string link description))
         (sit-for 0))))))
```

```
(defalias 'sacha-org-insert-link 'sacha-org-insert-link-dwim)
```

```
;;###autoload
(defun sacha-switch-to-previous-buffer ()
  "Switch to previously open buffer.
   Repeated invocations toggle between the two most recently open buffers."
  (interactive)
  (switch-to-buffer (other-buffer (current-buffer) 1)))
```

```
;;###autoload
(defun sacha-org-check-agenda ()
  "Peek at agenda."
  (interactive)
  (cond
    ((derived-mode-p 'org-agenda-mode)
     (if (window-parent) (delete-window) (bury-buffer)))
    ((get-buffer "*Org Agenda*")
     (switch-to-buffer-other-window "*Org Agenda*"))
    (t (org-agenda nil "a"))))
```

From <https://github.com/abo-abo/hydra/wiki/Nesting-Hydras> :

```
(defvar hydra-stack nil)

(defun sacha-hydra-push (expr)
  (push `(lambda () ,expr) hydra-stack))

(defun sacha-hydra-pop ()
  (interactive)
  (let ((x (pop hydra-stack)))
    (when x (funcall x))))

(defun sacha-hydra-go-and-push (expr)
```

```
(push hydra-curr-body-fn hydra-stack)
(prin1 hydra-stack)
(funcall expr))

;; example (progn (hydra-b/body) (hydra-push '(hydra-a/body)))
;; or ("q" hydra-pop "exit")
```

2.6.1 Emacs Hydra: Allow completion when I can't remember the command name

2021-04-29: Added the ability to complete using an arbitrary Hydra.

So it turns out that I'm pretty much zonked after a day with the kiddo and have a hard time remembering keystrokes or speed-reading [my Hydra cheat sheets](#) (p. 509). I want to be able to use M-x-like completion in my Hydra so that I can type a few characters and then maybe see the shortcuts there. Here's what it looks like:

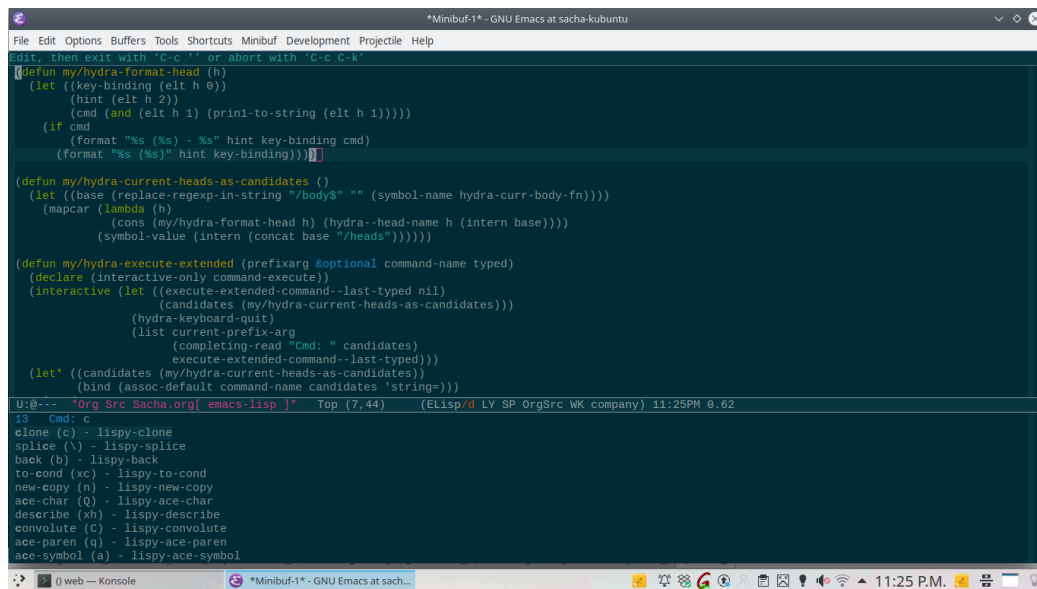


Figure 4: Hydra completion

```
(defun sacha-hydra-format-head (h)
  (let ((key-binding (elt h 0))
        (hint (elt h 2))
        (cmd (and (elt h 1) (prin1-to-string (elt h 1)))))
    (if cmd
      (format "%s (%s) - %s" hint key-binding cmd)
      (format "%s (%s)" hint key-binding))))

(defun sacha-hydra-heads-to-candidates (base)
  (mapcar (lambda (h)
            (cons (sacha-hydra-format-head h) (hydra--head-name h base)))
          (symbol-value (intern (concat (symbol-name base) "/heads")))))

;; ##autoload
(defun sacha-hydra-execute-extended (&optional _ hydra-base)
  (interactive (list current-prefix-arg nil))
  (hydra-keyboard-quit)
  (let* ((candidates (sacha-hydra-heads-to-candidates
```



```

      (or hydra-base
        (intern
         (replace-regexp-in-string "/body$" ""
                                     (symbol-name hydra-curr-body-fn))))))
      (command-name (completing-read "Cmd: " candidates))
      (bind (assoc-default command-name candidates 'string=)))
    (cond
      ((null bind) nil)
      ((hydra--callablep bind) (call-interactively bind))))))

```

This is how I add it to all my hydras:

```

(with-eval-after-load 'hydra
  (define-key hydra-base-map (kbd "<tab>") #'sacha-hydra-execute-extended))

```

Prooooobably works? Very rough. Might be useful for those fuzzy-brain days.

2.7 which-key and which-key-posframe

It's hard to remember keyboard shortcuts.

```

(use-package which-key
  :init (which-key-mode 1)
  :config
  (setq which-key-show-prefix 'top))
(use-package which-key-posframe :if sacha-laptop-p :init (which-key-posframe-mode 1))

```

Sometimes C-h gets weird and calls `which-key-C-h-dispatch`, probably from a transient map that got confused.

```

;;;###autoload
(defun sacha-reset-transients ()
  (interactive)
  (setq overriding-terminal-local-map nil))
(keymap-global-set "s-x" #'sacha-reset-transients)

```

2.8 Casual

```

(use-package casual
  :load-path "~/vendor/casual/lisp")

```

2.9 Foot pedal

```

(defun sacha-speechd-speak-sentence-and-advance ()
  "Speak the current sentence and move forward."
  (interactive)
  (call-interactively #'speechd-speak-read-sentence)
  (forward-sentence))

(defun sacha-speechd-repeat-sentence ()
  "Speak the current sentence and move forward."
  (interactive)
  (backward-sentence))

```

```
(call-interactively #'speechd-speak-read-sentence)
(forward-sentence))
```

```
;(keymap-global-set "S-<f1>" #'sacha-speechd-repeat-sentence)
;(keymap-global-set "S-<f3>" #'sacha-speechd-speak-sentence-and-advance)
```

3 Completion

<https://emacsredux.com/blog/2026/04/04/read-extended-command-predicate/>

```
(setq read-extended-command-predicate
      #'command-completion-default-include-p)
```

```
(global-completion-preview-mode 1)

(use-package vertico
  :config
  (vertico-mode +1)
  (vertico-multiform-mode)
  (with-eval-after-load 'vertico-multiform
    (add-to-list 'vertico-multiform-categories '(embark-keybinding grid))))
```

```
(use-package prescient :config (prescient-persist-mode +1))
                               ;(use-package company-prescient :init ↔
                               ↔ (company-prescient-mode +1))
```

3.0.1 Emacs completion and handling accented characters with orderless

emacs

I like using the [orderless](#) completion package for Emacs because it allows me to specify different parts of a completion candidate than any order I want. Because I'm learning French, I want commands like `consult-line` (which uses minibuffer completion) and `completion-at-point` (which uses in-buffer completion) to also match candidates where the words might have accented characters. For example, instead of having to type "utilisé" with the accented é, I want to type "utilise" and have it match both "utilise" and "utilisé".

```
(defvar sacha-orderless-accent-replacements
  '(("a" . "[aàáâãäåâ]")
    ("e" . "[eèéêë"] )
    ("i" . "[iîíï]" )
    ("o" . "[oòóôõöœ]" )
    ("u" . "[uùúû]" )
    ("c" . "[çç]" )
    ("n" . "[ññ]" ))) ; in case anyone needs ñ for Spanish

;;###autoload
(defun sacha-orderless-accent-dispatch (pattern &rest _)
  (seq-reduce
   (lambda (prev val)
     (replace-regexp-in-string (car val) (cdr val) prev))
   sacha-orderless-accent-replacements
   pattern))
```

```
(use-package orderless
  :custom
  (completion-styles '(orderless basic))
  (completion-category-overrides '(((file (styles basic partial-completion))))
  (orderless-style-dispatchers '(sacha-orderless-accent-dispatch orderless-affix-dispatch)))
```

U:-- journal-fr.org 15% (744,21) (Org Ind OSPE wb CP WK Wrap) 3:06PM 2.70
 1/148 Go to line: utilise
 744 - In 'si je peux l'utiliser dans le porche', 'sur le porche' is more common if it's an open area, though 'dans' works for an enc
 749 C'était le Jour de la famille, donc ma fille n'a pas eu école. Nous avons préparé des petits pains chinois au porc char siu. Cette
 753 Ma fille et moi avons utilisé Claude IA pour générer des cartes mémoire pour apprendre la négation, les conjugaisons du verbe « au
 766 Il faisait beau l'après-midi, donc je me suis assise dehors et j'ai bricolé mon smartphone pour faciliter le basculement entre le
 849 J'ai mis à jour la police de caractères qui est créée à partir de mon écriture. Je l'ai créée il y a 6 ans et je l'utilise pour le
 873 La programmation est généralement utile. Je suis particulièrement intéressée par l'automatisation et les aides cognitives et phys
 875 L'écriture m'aide à penser, à me remémorer et à partager. Org Mode sous Emacs me permet d'utiliser la technique de « programmation
 877 L'apprentissage du français depuis 4 mois touche aussi divers aspects de ma vie quotidienne. J'aide ma fille à l'école, j'essaie d

* mercredi 25 février...

* jeudi 26 fev

Une fois fève fèves février féveroles est encore réveillée à l'heure du petit-déjeuner.

This is an entry for [Emacs Carnival February 2026: Completion](#).

3.1 Consult

```
(use-package consult
  :load-path "~/vendor/consult"
  :after projectile
  :bind (("C-x r x" . consult-register)
        ("C-x r b" . consult-bookmark)
        ("C-c k" . consult-kmacro)
        ("C-x M-:" . consult-complex-command) ;; orig. repeat-complet-command
        ("C-x 4 b" . consult-buffer-other-window) ;; orig. switch-to-buffer-other-window
        ("C-x 5 b" . consult-buffer-other-frame)
        ("M-#" . consult-register-load)
        ("M-'" . consult-register-store) ;; orig. abbrev-prefix-mark (unrelated)
        ("C-M-#" . consult-register)
        ("M-g o" . consult-outline)
        ("M-g h" . consult-org-heading)
        ("M-g a" . consult-org-agenda)
        ("M-g m" . consult-mark)
        ("C-x b" . consult-buffer)
        ("M-g M-g" . consult-goto-line) ;; orig. goto-line
        ("M-g o" . consult-outline)
        ("M-g m" . consult-mark)
        ("M-g k" . consult-global-mark)
        ("M-g i" . consult-imenu)
        ("M-g I" . consult-project-imenu)
        ("M-g e" . consult-error)
        ;; M-s bindings (search-map)
        ("M-s f" . consult-find)
        ("M-s i" . consult-info)
        ("M-s L" . consult-locate)
        ("M-s g" . consult-grep)
        ("M-s G" . consult-git-grep)
        ("M-s r" . consult-ripgrep)
        ("M-s l" . consult-line)
        ("M-s m" . consult-multi-occur)
        ("M-s k" . consult-keep-lines))
```

```

("M-s u" . consult-focus-lines)
;; Isearch integration
("M-s e" . consult-isearch)
("M-g l" . consult-line)
("M-g L" . ace-link)
("M-s m" . consult-multi-occur)
("C-x c o" . consult-multi-occur)
("C-x c SPC" . consult-mark)
:map isearch-mode-map
("M-e" . consult-isearch) ;; orig. isearch-edit-string
("M-s e" . consult-isearch) ;; orig. isearch-edit-string
("M-s l" . consult-line))
:init
(setq register-preview-delay 0
  register-preview-function #'consult-register-format)
:custom
consult-preview-key '(:debounce 0.2 any)
consult-narrow-key "<"
consult-preview-excluded-files '("\\`/[^/|:]+:"
  "\\..gpg\\")
:config
(setq consult-project-root-function #'projectile-project-root))

```

3.1.1 Completing blog posts

```

(defalias 'sacha-complete-blog-post-url #'sacha-org-blog-complete)

;;;###autoload
(defun sacha-blog-posts (&optional start end filter-fn)
  (let ((json-object-type 'alist)
        (data (json-read-file "~/proj/static-blog/_site/blog/all/index.json")))
    (if (or start end)
        (seq-filter
         (lambda (o)
           (and (or (null start) (not (string< (alist-get 'date o) start)))
                (or (null end) (string< (alist-get 'date o) end))
                (or (null filter-fn) (funcall filter-fn o)))))
         data)
        data)))

;;;###autoload
(defun sacha-blog-edit-html (url)
  (interactive (list (sacha-org-blog-complete)))
  (let ((base (replace-regexp-in-string
                (concat "^" (regexp-quote sacha-blog-base-url))
                ""
                url)))
    (filename (sacha-11ty-html-filename url)))
  (if filename
      (find-file filename)
      (error "Could not find file.")))

;;;###autoload
(defun sacha-blog-edit-json (url)
  (interactive (list (sacha-org-blog-complete)))
  (let* ((filename (sacha-11ty-html-filename url))
         (json (and filename (concat (file-name-sans-extension filename) ".11tydata.json"))))

```

```

    (if json
      (find-file json)
      (error "Could not find file.))))

;;###autoload
(defun sacha-view-blog-post-locally (url)
  (interactive (list (sacha-org-blog-complete)))
  (browse-url
   (replace-regexp-in-string
    (concat "^" (regexp-quote sacha-blog-base-url))
    "https://localhost:8080/"
    url)))

;;###autoload
(defun sacha-insert-blog-post-url (url)
  (interactive (list (sacha-complete-blog-post-url)))
  (insert url))

;;###autoload
(defun sacha-blog-title (url)
  (let ((base (replace-regexp-in-string
                (concat "^" (regexp-quote sacha-blog-base-url))
                "/"
                url)))
    (alist-get 'title
               (seq-find (lambda (o)
                           (string= (alist-get 'permalink o) base))
                         (sacha-blog-posts)))))

;;###autoload
(defun sacha-insert-blog-post-link (url)
  (interactive (list (sacha-complete-blog-post-url)))
  (if (derived-mode-p 'org-mode)
      (insert (org-link-make-string
               url
               (sacha-blog-title url)))
      (insert url)))

```

3.1.2 Completing sketches

image

```

(declare-function 'sacha-geeie-view "Sacha.el")

(defvar sacha-sketch-preview 'text
  "*Preview sketches.
'text means show the associated text.
'geeie means open image in Geeie.
t means open image in Emacs.")

(defun sacha-image--state ()
  "Manage preview window and cleanup."
  ;; These functions are closures captured when the state is initialized by consult--read
  (let ((preview (consult--buffer-preview))
        (open (consult--temporary-files)))
    ;; The returned lambda is the actual preview function called by Consult
    (lambda (action cand)
      (unless cand
        (funcall open))

```

```

(when sachas-sketeh-preview
  (let ((filename (cond
    ((and (eq sachas-sketeh-preview 'text)
      (listp cand)
      (alist-get 'source_path cand))
    (alist-get 'source_path cand))
    ((and (listp cand)
      (alist-get 'source_path cand))
    (sachas-image-filename (file-name-base (alist-get 'source_path cand))))
    (t cand))))
    (when filename
      (pcase sachas-sketeh-preview
        ('geeqie (sachas-geeqie-view (list filename)))
        (_ (funcall preview action
          (and cand
            (eq action 'preview)
            (funcall open filename))))))))))

```

```

;;###autoload
(defun sachas-complete-sketeh-filename (&optional filter)
  (interactive)
  (consult--read (sachas-sketches filter)
    :sort nil
    :state (sachas-image--state)
    :prompt "Sketeh: "
    :category 'sketeh))

(defun sachas-date-from-filename (filename)
  (let ((f (file-name-nondirectory filename)))
    (if (string-match "^[-0-9]+" f)
      (replace-regexp-in-string "^[-0-9]" "" (match-string 0 f))
      nil)))

(defvar sachas-sketches nil "Cache for sketeh filenames.")

;;###autoload
(defun sachas-sketches (&optional filter)
  (interactive)
  (let (results)
    (setq results
      (apply 'append (mapcar (lambda (dir)
        (directory-files dir t "\\.(jpe?g\\|png\\|svg\\|)$")
        sachas-sketch-directories)))
      (when filter
        (setq results (seq-filter (lambda (o) (string-match filter o))
          results)))
      (sort
        results
        (lambda (a b)
          (string< (concat (or (sachas-date-from-filename b) "0") (file-name-nondirectory b))
            (concat (or (sachas-date-from-filename a) "0") (file-name-nondirectory a)))))))

;;###autoload
(defun sachas-find-sketeh (file)
  (interactive (list (sachas-complete-sketeh-filename)))
  (find-file file))

;;###autoload

```

```

(defun sacha-sketch-prepare-post (file)
  (interactive (list (sacha-complete-sketch-filename)))
  (insert (org-link-make-string (concat "sketchFull:" (file-name-base file))))
  (let ((text (sacha-sketch-text file)))
    (when text
      (insert (format "\n\n#+begin_my_src \"Text from %s\"\n\n%s\n#\"))))))

(defun sacha-sketch-text (file)
  (cond
   ((file-exists-p (concat (file-name-sans-extension file) ".txt"))
    (with-temp-buffer
      (insert-file-contents (concat (file-name-sans-extension file) ".txt"))
      (buffer-string)))
   ((file-exists-p (concat (file-name-sans-extension file) ".json"))
    (let ((json-object-type 'alist))
      (assoc-default 'description (elt (assoc-default 'textAnnotations (json-read-file (concat (file-name-sans-extension file) ".json"))) 0))))
    (t (error "Can't find text."))))

;; ##autoload
(defun sacha-sketch-insert-text (file)
  (interactive "FFile: ")
  (let ((text (sacha-sketch-text file)))
    (insert (or text ""))))

```

3.1.3 Consult directory navigation

```

(use-package consult-dir
  :ensure t
  :bind (("C-x C-d" . consult-dir)
         :map minibuffer-local-completion-map
         ("C-x C-d" . consult-dir)
         ("C-x C-j" . consult-dir-jump-file)))

```

```

;; https://karthinks.com/software/jumping-directories-in-eshell/
;; ##autoload
(defun eshell/z (&optional regexp)
  "Navigate to a previously visited directory in eshell, or to
any directory proffered by `consult-dir'."
  (let ((eshell-dirs (delete-dups
                      (mapcar 'abbreviate-file-name
                              (ring-elements eshell-last-dir-ring)))))
    (cond
     ((and (not regexp) (featurep 'consult-dir))
      (let* ((consult-dir--source-eshell `(:name "Eshell"
                                             :narrow ?e
                                             :category file
                                             :face consult-file
                                             :items ,eshell-dirs))
              (consult-dir-sources (cons consult-dir--source-eshell
                                           consult-dir-sources)))
        (eshell/cd (substring-no-properties
                    (consult-dir--pick "Switch directory: "))))
      (t (eshell/cd (if regexp (eshell-find-previous-directory regexp)
                          (completing-read "cd: " eshell-dirs)))))))

```

3.1.4 Using projects as a source for consult-buffer

```
(use-package consult
  :after projectile
  :defines consult-buffer-sources
  :config
  (projectile-load-known-projects)
  (setq sachal-consult-source-projectile-projects
    `(:name "Projectile projects"
      :narrow ?P
      :category project
      :action ,#'projectile-switch-project-by-name
      :items ,projectile-known-projects))
  (add-to-list 'consult-buffer-sources 'sachal-consult-source-projectile-projects 'append))
```

3.1.5 consult-omni

For some reason, installing `consult-omni` using the `:vc` keyword was giving me problems, so I checked it out from Github instead.

I also needed to create a Google custom search JSON API key at <https://developers.google.com/custom-search/v1/introduction>.

```
(defun sachal-insert-or-replace-link (url &optional title)
  "Insert a link, wrap the current region in a link, or replace the current link."
  (interactive (list (read-string "URL: ")))
  (cond
    ((derived-mode-p 'org-mode)
     (cond
       ((org-in-regexp org-link-bracket-re 1)
        (when (match-end 2) (setq title (match-string-no-properties 2)))
        (delete-region (match-beginning 0) (match-end 0)))
       ((org-in-regexp org-link-any-re 1)
        (delete-region (match-beginning 0) (match-end 0)))
       ((region-active-p)
        (setq title (buffer-substring-no-properties (region-beginning)
                                                    (region-end)))
        (delete-region (region-beginning) (region-end)))
        ;; update link
        (insert (org-link-make-string url title)))
     ((derived-mode-p 'org-mode) ; not in a link
      (insert (org-link-make-string url title)))
     ((and (region-active-p) (derived-mode-p 'markdown-mode))
      (setq title (buffer-substring-no-properties (region-beginning) (region-end)))
      (delete-region (region-beginning) (region-end))
      (insert (format "[%s](%s)" title url)))
     ((derived-mode-p 'markdown-mode)
      (insert (format "[%s](%s)" title url)))
     ((and (region-active-p) (string-match (regexp-quote "*new toot*") (buffer-name)))
      (setq title (buffer-substring-no-properties (region-beginning) (region-end)))
      (delete-region (region-beginning) (region-end))
      (insert (format "[%s](%s)" title url)))
     ((string-match (regexp-quote "*new toot*") (buffer-name))
      (insert (format "[%s](%s)" (read-string "Title: " (sachal-page-title url))
                                                    url)))
     (t
      (insert (format "%s (%s)" title url)))))
```



```

;; override the embark actions
(defun sacha-consult-omni-embark-copy-url-as-kill (cand)
  "Don't add spaces."
  (when-let ((s (and (stringp cand) (get-text-property 0 :url cand))))
    (kill-new (string-trim s))))

(defun sacha-consult-omni-embark-insert-url (cand)
  "Don't add spaces."
  (when-let ((s (and (stringp cand) (get-text-property 0 :url cand))))
    (insert (string-trim s))))

(defun sacha-consult-omni-embark-copy-title-as-kill (cand)
  "Don't add spaces."
  (when-let ((s (and (stringp cand) (get-text-property 0 :title cand))))
    (kill-new (string-trim s))))

(defun sacha-consult-omni-embark-insert-title (cand)
  "Don't add spaces."
  (when-let ((s (and (stringp cand) (get-text-property 0 :title cand))))
    (insert (string-trim s))))

(defun sacha-consult-omni-embark-insert-link (cand)
  "Don't add spaces."
  (let ((url (and (stringp cand) (get-text-property 0 :url cand)))
        (title (and (stringp cand) (get-text-property 0 :title cand))))
    (sacha-insert-or-replace-link url title)))

(use-package consult-omni
  :defer t
  :commands consult-omni
  :load-path "~/vendor/consult-omni"
  :after (consult embark)
  :custom
  (consult-omni-show-preview t) ;; show previews
  (consult-omni-preview-key "C-o") ;; set the preview key to C-o
  :config
  (add-to-list 'load-path "~/vendor/consult-omni/sources")
  (require 'consult-omni-sources)
  (require 'consult-omni-embark)
  (setq consult-omni-sources-modules-to-load (list 'consult-omni-wikipedia ←
    ↪ 'consult-omni-google))
  (consult-omni-sources-load-modules)
  (setq consult-omni-dynamic-input-debounce 1.0)
  (setq consult-omni-dynamic-refresh-delay consult-omni-dynamic-input-debounce)
  (setq consult-omni-default-interactive-command #'consult-omni-multi)
  (setq consult-omni-multi-sources
    '(consult-omni--source-google
      consult-omni--source-sacha-org-bookmarks
      consult-omni--source-blog))

  :bind
  (("M-g w" . consult-omni)
   ("M-g f" . consult-omni-sacha-org-bookmarks)
   :map consult-omni-embark-general-actions-map
   ("i l" . #'sacha-consult-omni-embark-insert-link)
   ("i u" . #'sacha-consult-omni-embark-insert-url)
   ("i t" . #'sacha-consult-omni-embark-insert-title)
   ("w u" . #'sacha-consult-omni-embark-copy-url-as-kill)
   ("w t" . #'sacha-consult-omni-embark-copy-title-as-kill)))

```

SOMEDAY Using web searches and bookmarks to quickly link placeholders in Org Mode

ORG:EMACS

- [2025-09-22 Mon]: Fix bug in `sacha-org-in-bracketed-text-link-p`.
- [2025-07-23 Wed]: Handle Org link updates from the middle of a link.

I want to make it easy to write with more hyperlinks. This lets me [develop thoughts over time](#), building them out of small chunks that I can squeeze into my day. It also makes it easy for you to dig into things that pique your curiosity without wading through lots of irrelevant details. Looking up the right URLs as I go along tends to disrupt my train of thought. I want to make it easier to write down my thoughts first and then go back and add the URLs. I might have 5 links in a post. I might have 15. Sounds like an automation opportunity.

If I use double square brackets around text to indicate where I want to add links, [Orgzly Revived](#) and [Org Mode](#) both display those as links, so I can preview what the paragraph might feel like. They're valid links. Org Mode prompts me to create new headings if I follow them. I never use these types of links to point to headings, though. Since I only use custom IDs for links, any `[[some text]]` links must be a placeholder waiting for a URL. I want to turn `[[some text]]` into something like `[[https://example.com][some text]]` in the Org Mode markup, which gets exported as a hyperlink like this: [some text](#). To figure out the target, I might search the web, my blog, or my bookmarks, or use the custom link types I've defined for Org Mode with their own completion functions.

Most of my targets can be found with a web search. I can do that with [consult-omni](#) with a default search based on the link text, prioritizing my [bookmarks](#) (p. 57) and [blog posts](#). Then I don't even have to retype the search keywords.

```
;; we're in a bracketed link with no description and the target doesn't look like a link;
;; likely I've actually added the text for the description and now we need to include the link
(defun sacha-org-in-bracketed-text-link-p ()
  (when (and (derived-mode-p 'org-mode) org-link-bracket-re)
    (let* ((bracket-pos (org-in-regexp org-link-bracket-re))
           (bracket-target (and bracket-pos (match-string 1)))
           (bracket-desc (and bracket-pos (match-string 2))))
      (and bracket-pos bracket-target
           (null bracket-desc)
           ;; try to trigger only when the target is plain text and doesn't have a protocol
           (not (string-match ":" bracket-target)))))

;; ###autoload
(defun sacha-org-set-link-target-with-search ()
  "Replace the current link's target with a web search.
  Assume the target is actually supposed to be the description. For
  example, if the link is [[some text]], do a web search for 'some text',
  prompt for the link to use as the target, and move 'some text' to the
  description."
  (interactive)
  (let* ((bracket-pos (org-in-regexp org-link-bracket-re))
         (bracket-target (and bracket-pos (match-string 1)))
         (bracket-desc (match-string 2))
         result)
    (when (sacha-org-in-bracketed-text-link-p)
      (let ((link (consult-omni bracket-target nil nil t)))
        (cond
         ((get-text-property 0 :url link)
          (setq result (org-link-make-string (get-text-property 0 :url link)
                                             bracket-target)))
         ((string-match ":" link) ; might be a URL
          (setq result (org-link-make-string link bracket-target))))))
```

```
(when result
  (delete-region (car bracket-pos) (cdr bracket-pos))
  (insert result)
  result))))
```

This is what that looks like:

[https://sachachua.com/blog/2025/07/finding-unlinked-text/2025-07-21_3.58.10.webm?caption = Screenecast : fillinginalinkURLbasedonthetext](https://sachachua.com/blog/2025/07/finding-unlinked-text/2025-07-21_3.58.10.webm?caption=Screenecast%3AfillinginalinkURLbasedonthetext)

consult-omni shows me the title, first part of the URL, and a few words from the page. `C-o` in consult-omni previews search results, which could be handy.

Sometimes a web search isn't the fastest way to find something. Some links might be to my Emacs configuration, project files, or other things for which I've written custom Org link types. I can store links to those and insert them, or I can choose those with completion.

```
;; ##autoload
(defun sach-org-set-link-target-with-org-completion ()
  "Replace the current link's target with `org-insert-link' completion.
  Assume the target is actually supposed to be the description.  For
  example, if the link is [[some text]], do a web search for 'some text',
  prompt for the link to use as the target, and move 'some text' to the
  description."
  (interactive)
  (let* ((bracket-pos (org-in-regexp org-link-bracket-re))
         (bracket-target (match-string 1))
         (bracket-desc (match-string 2))
         result)
    (when (and bracket-pos bracket-target
                (null bracket-desc)
                ;; try to trigger only when the ↵
                (not (string-match ":" ↵
                                   (org-element-lineage ↵
                                   (org-element-context) '(link) t)) ; ignore text in code blocks, etc.
                ;; we're in a bracketed link with no description and the target ↵
                ↵ doesn't look like a link;
                ;; likely I've actually added the text for the description and now ↵
                ↵ we need to include the link.
                ;; This is a hack so that we don't have to delete the link until the ↵
                ↵ new link has been inserted
                ;; since org-insert-link doesn't tbreak out the link prompting code ↵
                ↵ into a smaller function.
          (let ((org-link-bracket-re "{{{}}}")
                (goto-char (cdr bracket-pos))
                (org-insert-link nil nil bracket-target))
            (delete-region (car bracket-pos) (cdr bracket-pos))))))
```

Here's an example:

[https://sachachua.com/blog/2025/07/finding-unlinked-text/2025-07-21_3.59.47.webm?caption = Screenecastshowinghowtospecifyalinkwithcompletion](https://sachachua.com/blog/2025/07/finding-unlinked-text/2025-07-21_3.59.47.webm?caption=Screenecastshowinghowtospecifyalinkwithcompletion)

If I decide a web search isn't the best way to find the target, I can use `<up>` and `RET` to get out of the consult-omni web search and then go to the usual Org link completion interface.

```
;; ##autoload
```

```
(defun sacha-org-set-link-target-dwim ()
  (interactive)
  (or (sacha-org-set-link-target-with-search)
      (sacha-org-set-link-target-with-org-completion)))
```

If I can't find the page either way, I can use `C-g` to cancel the prompt, look around some more, and get the URL. When I go back to the web search prompt for the link target, I can press `<up> RET` to switch to the Org completion mode, and then paste in the URL with `C-y` or type it in. Not elegant, but it will do.

I can now look for untargeted links and process each one. The `(undo-boundary)` in the function means I can undo one link at a time if I need to.

```
;; ##autoload
(defun sacha-org-scan-for-untargeted-links ()
  "Look for [[some text]] and prompt for the actual targets."
  (interactive)
  (while (re-search-forward org-link-bracket-re nil t)
    (when (and
          (not (match-string 2))
          (and (match-string 1) (not (string-match ":" ↵
↵ (match-string 1))))
          (org-element-lineage (org-element-context) '(link) ↵
↵ t)) ; ignore text in code blocks, etc.
      (undo-boundary)
      (sacha-org-set-link-target-dwim))))
```

Here's how that works:

https://sachachua.com/blog/2025/07/finding-unlinked-text/2025-07-22_7.55.26.webm?caption=ScreencastrhowIcanscanthroughthetextforuntargetedlinksandupdatethem
 nil

If `sacha-org-scan-for-untargeted-links` doesn't find anything, then the post is probably ready to go (at least in terms of links). That might help avoid accidentally posting placeholders. I'm going to experiment with going back to the default of having `org-export-with-broken-links` be `nil` again, so that's another safety net that should catch placeholders before they get published.

1. Next steps I can look at the code for the web search and add the same kind of preview function for my bookmarks and blog posts.

I can modify my `C-c C-l` binding (`sacha-org-insert-link-dwim` (p. 314)) to have the same kind of behaviour: do a web/bookmark/blog search first, and fall back to Org link completion.

Someday it might be nice to add a font-locking rule so that links without proper targets can be shown in a different colour. `org-export-with-broken-links` and `org-link-search` both know about these types of links, so there might be a way to figure out font-locking.

I might not use the exact words from the title, so it would be good to be able to specify additional keywords and rank by relevance. The `p-search` talk from EmacsConf 2024 showed a possible approach that I haven't dug into yet. If I want to get really fancy, it would be neat to use the embedding of the link text to look up the most similar things (blog posts, bookmarks) and use that as the default.

I'm looking forward to experimenting with this. I think it will simplify linking to things when I'm editing my drafts on my computer. That way, it might be easier for me to write about whatever nifty idea I'm curious about while helping people pick up whatever background information they need to make sense of it all.

Bookmarks I've been putting my bookmarks in an Org file. Let's add [my bookmarks](#) (p. 246) as a consult-omni source.

```
;; ###autoload
(defun sacha-consult-omni-bookmarks-builder (input &rest args &key callback &allow-other-keys)
  (let* ((quoted (when input (regexp-quote input)))
         (list (sacha-org-bookmarks))
         (candidates
          (mapcar
           (lambda (o)
             (propertize
              (concat (plist-get o :title) "\s"
                      ↵
              (plist-get o :url))
              :source "Bookmarks"
              :on-callback ↵
              :title (plist-get o :title)
              :url (plist-get o :url)))
           (if quoted
               (seq-filter
                (lambda (o)
                  (string-match quoted ↵
                  (concat (plist-get o :title) " - " (plist-get o :title))))
               list)))
         (when callback (funcall callback candidates))
         candidates))

;; ###autoload
(defun sacha-consult-org-bookmark-visit (o)
  (browse-url (get-text-property 0 :url o)))

;; (consult--multi (list sacha-consult--source-org-bookmark))
(with-eval-after-load 'consult-omni
  (consult-omni-define-source
   "My Org bookmarks"
   :narrow-char ?b
   :type 'sync
   :request #'sacha-consult-omni-bookmarks-builder
   :on-return 'sacha-consult-org-bookmark-visit
   :group #'consult-omni--group-function
   :min-input 1
   :require-match t))
```

DONE Finding my blog posts with consult-omni EMACS Sometimes I just want to quickly get to a blog post by title. I use [consult-omni](#) for quick web searches that I can jump to or insert as a link. Sure, I can limit this search to my blog by specifying `site:sachachua.com` or using the code I wrote to [search my blog, notes, and sketches with consult-ripgrep and consult-omni](#) (p. 59), but the search sometimes gets confused by other text on the page. When I [publish my blog with Eleventy](#), I also create a JSON file with [all my blog post URLs and titles](#). Here's how I can use that data as a consult-omni source.

```
;; ###autoload
(defun sacha-consult-omni-blog-data ()
  (let ((base (replace-regexp-in-string "/"$ " " sacha-blog-base-url))
        (json-object-type 'alist))
```



```
:sort nil
(require-match t))
```

Here's what it looks like by itself when I call `consult-omni-sacha-blog`:

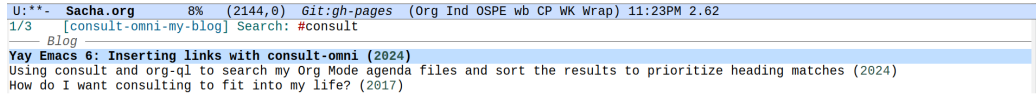


Figure 5: Screenshot of a search through my blog post titles

Then I can add it as one of the sources used by `consult-omni`:

```
(setq consult-omni-multi-sources
      '(consult-omni--source-google
        consult-omni--source-sacha-org-bookmarks
        consult-omni--source-blog))
```

Here's what it looks like when I call `consult-omni` to search my bookmarks, blog posts, and Google results at the same time.

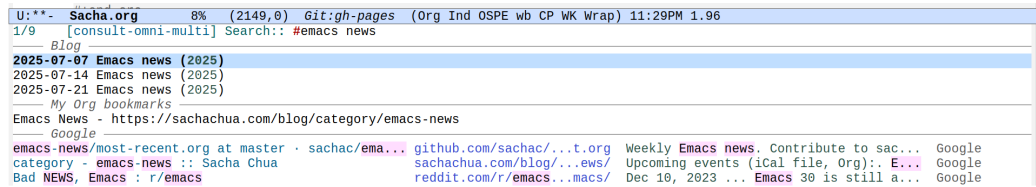


Figure 6: Screenshot of a consult-omni search with multiple sources

Then I can press `RET` to open the blog post in my browser or `C-. i l` to [insert the link using Embark](#) (p. 52) (`sacha-consult-omni-embark-insert-link`).

Related:

- [consult-omni-source-sacha-org-bookmarks](#) (p. 57)
- [Completing blog posts](#) (p. 48) (Org link type with completion)
- [Using web searches and bookmarks to quickly link placeholders in Org Mode](#)

Next steps: I want to find out how to get `:sort nil` to be respected so that more recent blog posts are listed first. Also, it might be fun to define a similar source for the sections of my Emacs configuration, like the way I can use my [dotemacs: Org Mode link type with completion](#) (p. 320).

3.1.6 STARTED Searching my blog, notes, and sketches with `consult-ripgrep` and `consult-omni`

```
;; ##autoload
(defun sacha-search-notes ()
  (interactive)
  (consult-ripgrep '("~/sync/orgzly" "~/sync/static-blog/blog" "~/sync/sketches" ↔
    ↳ "~/sync/topics"))))
;; ##autoload
(defun sacha-search-public-notes ()
  (interactive)
  (consult-ripgrep '("~/sync/static-blog/blog" "~/sync/sketches" "~/sync/topics")))
```

```

;; ##autoload
(cl-defun sachalconsult-omni--google-blog-fetch-results (input &rest args &key callback ←
  ↪ &allow-other-keys)
  "Fetches search results for INPUT from "Google Custom "Search service.
  Narrows to `sachalblog-base-url'."

  Refer to URL `https://programmablesearchengine.google.com/about/' and
  URL `https://developers.google.com/custom-search/' for more info."
  (pcase-let* ((`(. query . ,opts)
    (consult-omni--split-command input (seq-difference args (list :callback ←
  ↪ callback))))
    (opts (car-safe opts))
    (count (plist-get opts :count))
    (page (plist-get opts :page))
    (filter (plist-get opts :filter))
    (count (or (and count (integerp (read count))) (string-to-number count))
      consult-omni-default-count))
    (page (or (and page (integerp (read page))) (string-to-number page))
      consult-omni-default-page))
    (filter (or (and (integerp filter) filter)
      (and filter (string-to-number (format "%s" filter))
        1)))
    (filter (if (member filter '(0 1)) filter 1))
    (count (min count 10))
    (page (+ (* page count) 1))
    (page (min page (- 100 count)))
    (params `(("q" . ,(format "site:%s+%s"
      (url-encode-url sachalblog-base-url)
      (replace-regexp-in-string " " "+" query))))
      ("key" . ,(consult-omni-expand-variable-function ←
  ↪ consult-omni-google-customsearch-key))
      ("cx" . ,(consult-omni-expand-variable-function ←
  ↪ consult-omni-google-customsearch-cx))
      ("gl" . "en")
      ("filter" . ,(format "%s" filter))
      ("num" . ,(format "%s" count))
      ("start" . ,(format "%s" page))))
    (headers '(("Accept" . "application/json")
      ("Accept-Encoding" . "gzip")
      ("User-Agent" . "consult-omni (gzip)"))))
    (consult-omni--fetch-url
      consult-omni-google-customsearch-api-url consult-omni-http-retrieve-backend
      :encoding 'utf-8
      :params params
      :headers headers
      :parser #'consult-omni--json-parse-buffer
      :callback
      (lambda (attrs)
        (let* ((raw-results (gethash "items" attrs))
          (annotated-results
            (mapcar (lambda (item)
              (let*
                ((source "Google")
                 (url (format "%s" (gethash "link" item)))
                 (title (format "%s" (gethash "title" item)))
                 (snippet (string-trim (format "%s" (gethash "snippet" item))))
                 (search-url (consult-omni--make-url-string ←
  ↪ consult-omni-google-search-url params '("key" "cx" "gl"))

```



```

                                (decorated (funcall consult-omni-default-format-candidate ↵
↵ :source source :query query :url url :search-url search-url :title title :snippet ↵
↵ snippet)))

                                (propertize decorated
                                    :source source
                                    :title title
                                    :url url
                                    :search-url search-url
                                    :query query
                                    :snippet snippet)))

                                raw-results)))
(when (and annotated-results (functionp callback))
  (funcall callback annotated-results)
  annotated-results))))))

(use-package consult-omni
  :load-path "~/vendor/consult-omni"
  :after (consult embark)
  :config
  (consult-omni-define-source
    "Google my blog"
    :narrow-char ?b
    :type 'dynamic
    :require-match nil
    :face 'consult-omni-engine-title-face
    :request #'sacha-consult-omni--google-blog-fetch-results
    :on-new (apply-partially #'consult-omni-external-search-with-engine "Google")
    :preview-key consult-omni-preview-key
    :search-hist 'consult-omni--search-history
    :select-hist 'consult-omni--selection-history
    :enabled (lambda () (bound-and-true-p consult-omni-google-customsearch-key))
    :group #'consult-omni--group-function
    :sort t
    :interactive consult-omni-interactive-commands-type
    :annotate nil))

```

To get more results, I can add parameters like `--count 10` and `--page 2`. I wonder if there's an easier keyboard shortcut for paginating through results...

Let's try these shortcuts...

```

(keymap-global-set "M-g b" #'sacha-search-public-notes)
(keymap-global-set "M-g N" #'sacha-search-notes)
(keymap-global-set "M-g B" #'consult-omni-google-sacha-blog)

```

3.2 Marginalia

Marginalia - add function name for aliases

```

(eval-when-compile (require 'marginalia))
;; ##autoload
(defun sacha-marginalia-annotate-variable (cand)
  "Annotate variable CAND with its documentation string.
Omit values when streaming."
  (when-let* ((sym (intern-soft cand)))
    (marginalia--fields
      ((marginalia--symbol-class sym) :face 'marginalia-type)

```

```
((or (documentation-property sym 'variable-documentation)
      (marginalia--definition-prefix sym))
      :truncate 1.0 :face 'marginalia-documentation))))
```

```
(use-package marginalia
  :vc (:url "https://github.com/minad/marginalia")
  :init
  (marginalia-mode)
  :bind (:map minibuffer-local-completion-map
            ("M-m" . marginalia-cycle))
  :config
  (add-to-list 'marginalia-prompt-categories '("sketch" . sketch))
  (add-to-list 'marginalia-censor-variables "-api-key")
  (add-to-list 'marginalia-censor-variables "-private")
  (cl-pushnew #'marginalia-annotate-symbol-with-alias
    (alist-get 'command marginalia-annotators))
  (cl-pushnew #'marginalia-annotate-symbol-with-alias
    (alist-get 'function marginalia-annotators))
  (cl-pushnew #'marginalia-annotate-symbol-with-alias
    (alist-get 'symbol marginalia-annotators)))
```

```
;; ##autoload
(defun marginalia-annotate-alias (cand)
  "Annotate CAND with the function it aliases."
  (when-let ((sym (intern-soft cand))
              (alias (car (last (function-alias-p sym))))
              (name (and (symbolp alias) (symbol-name alias))))
    (format " (%s)" name)))

;; ##autoload
(defun marginalia-annotate-symbol-with-alias (cand)
  "Annotate symbol CAND with its documentation string.
  Similar to `marginalia-annotate-symbol'."
  (when-let (sym (intern-soft cand))
    (concat
      (marginalia-annotate-binding cand)
      (marginalia--fields
        ((marginalia-annotate-alias cand) :face 'marginalia-function)
        ((marginalia--symbol-class sym) :face 'marginalia-type)
        ((cond
          ((fboundp sym) (marginalia--function-doc sym))
          ((facep sym) (documentation-property sym 'face-documentation))
          (t (documentation-property sym 'variable-documentation)))
          :truncate 1.0 :face 'marginalia-documentation))))))
```

3.2.1 Marginalia and annotating journal entries

The following code annotates journal entries with their categories.

```
;; ##autoload
(defun sachamarginalia-annotate-journal (cand)
  (when-let ((o (cdr (assoc cand sachajournal-search-cache))))
    (marginalia--fields
      ((plist-get o :Category)
       :face 'marginalia-documentation
       :truncate 13))))
```

```
(use-package marginalia
  :config
  (add-to-list 'marginalia-annotators '(journal sachamarginalia-annotate-journal builtin ↵
    ↵ none)))
```

3.3 Cargo-culted stuff

```
;; ###autoload
(defun sachastore-action-key+cmd (cmd)
  (setq keycast--this-command-keys (this-single-command-keys) keycast--this-command cmd))
;; ###autoload
(defun sachaforce-keycast-update (&rest _)
  (force-mode-line-update t))
```

```
(use-package keycast
  :if sachalaptop-p
  :after embark
  :defer t
  :config (dolist (cmd '(embark-act embark-act-noexit embark-become))
    (advice-add cmd
      :before #'sachaforce-keycast-update)))

(use-package
  embark
  :config
  ;(setq embark-prompter ↵
  ↵ 'embark-completing-read-prompter)
  (advice-add 'embark-keymap-prompter :filter-return #'sachastore-action-key+cmd)
  (add-to-list 'embark-target-injection-hooks '(sachastream-message embark--allow-edit)))
```

4 Appearance

4.1 color-theme sometimes comes across lists. Odd!

```
(defadvice face-attribute (around sacha activate)
  (if (symbolp (ad-get-arg 0))
      ad-do-it))
```

4.2 Display

```
;; ###autoload
(defun sanityinc/adjust-opacity (frame incr)
  (let* ((oldalpha (or (frame-parameter frame 'alpha) 100))
        (newalpha (+ incr oldalpha)))
    (when (and (<= frame-alpha-lower-limit newalpha) (>= 100 newalpha))
      (modify-frame-parameters frame (list (cons 'alpha newalpha))))))
```

```
(keymap-global-set "C-M-8" (lambda () (interactive) (sanityinc/adjust-opacity nil -2)))
(keymap-global-set "C-M-9" (lambda () (interactive) (sanityinc/adjust-opacity nil 2)))
```

```
(keymap-global-set "C-M-0" (lambda () (interactive) (modify-frame-parameters nil `((alpha . ↵ ↵ 100)))))
```

From <https://protesilaos.com/codelog/2024-11-28-basic-emacs-configuration/>:

```
(add-to-list 'display-buffer-alist
  '("\\`\\*\\|(Warnings\\|Compile-Log\\|)\\*\\|'"
    (display-buffer-no-window)
    (allow-no-window . t)))
```

4.3 Color theme

4.3.1 Set up a color scheme

```
;;###autoload
(defun sacha-setup-color-theme ()
  (interactive)
  (when (display-graphic-p)
    (load-theme (car modus-themes-to-toggle) t)))
```

```
(use-package modus-themes
  :vc (:url "https://github.com/protesilaos/modus-themes")
  :init (setq modus-themes-to-toggle '(modus-operandi-tinted modus-vivendi-tinted))
  :config (sacha-setup-color-theme))
```

I sometimes need to switch to a lighter background for screenshots. For that, I use `modus-operandi-tinted`.

4.3.2 Making highlight-sexp follow modus-themes-toggle

elisp:emacs

[2023-01-27 Fri] Prot just added a `modus-themes-get-color-value` function. Yay! Also, it turns out that I need to update the overlay in all the buffers.

I'm experimenting with using the `highlight-sexp` minor mode to highlight my current s-expression, since I sometimes get confused about what I'm modifying with smartparens. The `highlight-sexp` background colour is hardcoded in the variable `hl-sexp-background-color`, and will probably look terrible if you use a light background. I wanted it to adapt when I use `modus-themes-toggle`. Here's how that works:

```
(defun sacha-hl-sexp-update-overlay ()
  (when (overlayp hl-sexp-overlay)
    (overlay-put
      hl-sexp-overlay
      'face
      `(:background
        ,(modus-themes-get-color-value 'bg-inactive)))))
(defun sacha-hl-sexp-update-all-overlays (&rest args)
  (dolist (buf (buffer-list))
    (with-current-buffer buf
      (when highlight-sexp-mode
        (sacha-hl-sexp-update-overlay)))))
```

```
(use-package highlight-sexp
  :vc (:url "https://github.com/daimrod/highlight-sexp")
  :after modus-themes
  :hook
  ((emacs-lisp-mode . highlight-sexp-mode)
   (modus-themes-after-load-theme . sachahl-sexp-update-all-overlays))
  :config
  (advice-add 'hl-sexp-create-overlay :after 'sachahl-sexp-update-overlay))
```

This is what it looks like:

[images/highlight-sexp.gif](#)

4.4 Modeline

4.4.1 Time in the modeline

I like having the clock.

```
(display-time-mode 1)
```

4.4.2 Diminish mode names in modeline

```
(use-package diminish :ensure t)
```

4.4.3 Highlight the active modeline using colours from modus-themes

emacs

I wanted to experiment with [Ignacio Paz Posse's snippet](#) for colouring the mode line of the active window ever so slightly different to make it easier to see where the active window is. I usually have `global-hl-line-mode` turned on, so that highlight is another indicator, but let's see how this tweak feels. I modified the code so that it uses the theme colours from the currently-selected Modus themes, since I trust Prot's colour choices more than I trust mine. Thanks to Irreal for sharing Ignacio's comment!

```
;; ##autoload
(defun sachahl-update-active-mode-line-colors ()
  (set-face-attribute
   'mode-line nil
   :foreground (modus-themes-get-color-value 'fg-mode-line-active)
   :background (modus-themes-get-color-value 'bg-blue-subtle)))
```

```
(use-package modus-themes
  :hook
  (modus-themes-after-load-theme . sachahl-update-active-mode-line-colors))
```

```
Marking The Active Window | Irreal: https://irreal.org/blog/?p=11867#comment-6354017310
Not A Text Editor --

Marking The Active Window

Posted on 2023-12-27 by jcs

I use the default mode line and am mostly happy with it except that the difference between the mode line of the active window and the other windows can be hard to discern. I know I could make myself a custom mode line like Prot did that makes the active window obvious but I'm not a fan of bling and it always seemed more trouble than it's worth.

Still, it would be nice to have the active window clearly marked. James Dyer had a similar problem and found a nice solution. His idea was to color the left hand fringe of the active window red. That turns out to be pretty easy as you can see from his post.

Another, possibly better, solution is to set the background color of the active window's mode line to a different color. That's what happens now, of course. But I have in mind something a little more distinctive than slightly different shades of gray. In theory, that shouldn't be too hard but as Prot showed messing around with the mode line is trickier than you might think.

'mode-line nil
:foreground (modus-themes-get-color-value 'fg-mode-line-active)
:background (modus-themes-get-color-value 'bg-blue-subtle)
:box '(:line-width
1
:color
(modus-themes-get-color-value 'border-mode-line-active)))
(use-package modus-themes
:hook
(modus-themes-after-load-theme . my-update-active-mode-line-colors))
#+end_src

** Discovering things
:PROPERTIES:...
*** Helm-descbinds
:PROPERTIES:...
Great for describing bindings. I'll replace the binding for =where-is= too.
U : %*- *eww* 7% (25,9) [emacs.,#emacsconf-q,#f,#emacsconf-g,#emacsconf-d,#emacsconf-a,#emacsconf,#emacs,L] (eww wb WK) 8:10
```

```
Marking The Active Window | Irreal: https://irreal.org/blog/?p=11867#comment-6354017310
Not A Text Editor --

Marking The Active Window

Posted on 2023-12-27 by jcs

I use the default mode line and am mostly happy with it except that the difference between the mode line of the active window and the other windows can be hard to discern. I know I could make myself a custom mode line like Prot did that makes the active window obvious but I'm not a fan of bling and it always seemed more trouble than it's worth.

Still, it would be nice to have the active window clearly marked. James Dyer had a similar problem and found a nice solution. His idea was to color the left hand fringe of the active window red. That turns out to be pretty easy as you can see from his post.

Another, possibly better, solution is to set the background color of the active window's mode line to a different color. That's what happens now, of course. But I have in mind something a little more distinctive than slightly different shades of gray. In theory, that shouldn't be too hard but as Prot showed messing around with the mode line is trickier than you might think.

'mode-line nil
:foreground (modus-themes-get-color-value 'fg-mode-line-active)
:background (modus-themes-get-color-value 'bg-blue-subtle)
:box '(:line-width
1
:color
(modus-themes-get-color-value 'border-mode-line-active)))
(use-package modus-themes
:hook
(modus-themes-after-load-theme . my-update-active-mode-line-colors))
#+end_src

#+CAPTION: with dark mode
file:/home/sacha/recordings/2024-01-01-08-10-01.svg

** Discovering things
U : **- Sacha.org 9% (1815,0) Git:gh-pages [emacs.,#emacsconf-q,#f,#emacsconf-g,#emacsconf-d,#emacsconf-a,#emacsconf,#emacs,L] (0)
```

4.5 Quickly adding face properties to regions

emacs

- [2024-09-20 Fri]: Set the first frame of the animated GIF to a reasonable backup image.
- [2024-09-20 Fri]: Add `:init-value nil` to the mode.

images/output-2024-09-20-13:09:17.gif

Sometimes I just want to make some text look a little fancier in the buffer so that I can make a thumbnail or display a message. This `sacha-add-face-text-property` function lets me select a region and temporarily change its height, make it bold, or do other things. It will work in `text-mode` or `enriched-mode` buffers (not Org Mode or programming buffers like `*scratch*`, as those do a lot of font-locking).

```
;; ###autoload
(defun sacha-add-face-text-property (start end attribute value)
  (interactive
   (let ((attribute (intern
```

```

↳ (completing-read
↳ "Attribute: "
↳ (mapcar (lambda (o) (symbol-name (car o)))
↳
      (list (point)
              face-attribute-name-alist))))
      (list (point)
              (mark)
              attribute
              (read-face-attribute '(() attribute))))
      (add-face-text-property start end (list attribute value)))

```

`enriched-mode` has some keyboard shortcuts for face attributes (`M-o b` for bold, `M-o i` for italic). I can add some keyboard shortcuts for other properties even if they can't be saved in `text/enriched` format.

```

;;;###autoload
(defun sacha-face-text-larger (start end)
  (interactive "r")
  (add-face-text-property
    start end
    (list :height (floor (+ 50 (car (alist-get :height (get-text-property start 'face)
↳ ' (100)))))))
↳ ' (100))))))
;;;###autoload
(defun sacha-face-text-smaller (start end)
  (interactive "r")
  (add-face-text-property
    start end
    (list :height (floor (- (car (alist-get :height (get-text-property start 'face)
↳ ' (100))) 50))))
↳ ' (100))) 50))))

```

What's an easy way to make this keyboard shortcut available during the rare times I want it? I know, maybe I'll make a quick minor mode so I don't have to dedicate those keyboard shortcuts all the time. `repeat-mode` lets me change the size by repeating just the last keystroke.

```

(defvar-keymap sacha-face-text-property-mode-map
  "M-o p" #'sacha-add-face-text-property
  "M-o +" #'sacha-face-text-larger
  "M-o -" #'sacha-face-text-smaller)
(define-minor-mode sacha-face-text-property-mode
  "Make it easy to modify face properties."
  :init-value nil
  (repeat-mode 1))
(defvar-keymap sacha-face-text-property-mode-repeat-map
  :repeat t
  "+" #'sacha-face-text-larger
  "-" #'sacha-face-text-smaller)
(dolist (cmd '(sacha-face-text-larger sacha-face-text-smaller))
  (put cmd 'repeat-map 'sacha-face-text-property-mode-repeat-map))

```

5 Navigation

```

(transient-mark-mode 1)
(minibuffer-regex-mode 1)

```

```
(keymap-global-set "C-x !" #'delete-other-windows-vertically)
```

```
(defun sacha-close-other-buffers ()  
  (interactive)  
  (mapc (lambda (buf)  
    (unless (buffer-modified-p buf)  
      (kill-buffer buf)))  
    (delete (current-buffer)  
      (buffer-list))))
```

5.1 Going to the last change

```
(use-package goto-chg  
  :bind  
  ((("C-" . goto-last-change)  
    ("C-" . goto-last-change-reverse)))
```

5.2 Substitution

```
(use-package substitute  
  :bind ("C-;" . substitute-target-in-buffer)  
  :hook  
  (substitute-post-replace . #'substitute-report-operation)  
  )
```

5.3 Expand region

```
(use-package expand-region  
  :bind ("C-=" . er/expand-region)  
  )
```

5.4 `selected.el`: Special keymap when the region is active

[2026-03-31 Tue] I removed the binding for "s" from the linked config so that it doesn't interfere with staging partial hunks in Magit.

I like context-sensitive keyboard shortcuts, so when I came across [this selected.el config](#) in the process of preparing Emacs News, I wanted to give it a try. It provides a special keymap when the region is active.

```
(use-package ctrlf  
  ;:init (ctrlf-mode +1) ; I'm also getting used to isearch-lazy-count, though.  
  )  
(use-package selected  
  :init (selected-global-mode 1)  
  :bind (:map selected-keymap  
    ("q" . selected-off)  
    ("u" . upcase-dwim)  
    ("d" . downcase-dwim)  
    ("w" . kill-ring-save)  
    ("n" . next-line)
```



```

("p" . previous-line)
("f" . forward-char)
("b" . backward-char)
("F" . forward-word)
("B" . backward-word)
("a" . beginning-of-visual-line)
("e" . end-of-visual-line)
("{ " . backward-paragraph)
("}" . forward-paragraph)
("(" . backward-sentence)
(")" . forward-sentence)
("r" . ctrlf-backward-fuzzy)
("[ " . scroll-down-line)
("]" . scroll-up-line)
("M" . rectangle-mark-mode)
("R" . replace-rectangle)
("x" . exchange-point-and-mark)))

```

5.5 Copy and append string

```

(defvar sach-copy-append-string nil "String to append when copying.")
;; ###autoload
(defun sach-copy-and-append (beg end append)
  (interactive
   (list
    (if (region-active-p) (region-beginning) (point-min))
    (if (region-active-p) (region-end) (point-max))
    (if current-prefix-arg (read-string "Append: ")
      sach-copy-append-string)))
  (when append
   (setq sach-copy-append-string append)
   (kill-new (concat (buffer-substring beg end) append))))

```

5.6 PDF view

5.6.1 TOBLOG Copy text from current PDFview page in other window

This makes it easier to pick up text from my annotated PDFs.

```

;; ###autoload
(defun sach-pdf-view-insert-current-page-text ()
  (interactive)
  (let (text)
    (catch 'found
      (walk-window-tree
       (lambda (win)
         (with-selected-window win
           (when (derived-mode-p 'pdf-view-mode)
             (setq text
                    (pdf-info-gettext (pdf-view-current-page)
                                       (list 0 0 1 1))))
             (throw 'found text))))))
    (when text (save-excursion (insert text)))))

```

5.7 Links

```
(use-package ace-link)
```

5.8 Manage windows

<https://emacsredux.com/blog/2026/04/07/stealing-from-the-best-emacs-configs/>

```
(setq window-combination-resize t)
```

5.8.1 Rotate windows

```
(keymap-global-set "M-r" #'window-layout-transpose) ; Emacs 31: override ↔  
↔ move-to-window-line-top-bottom  
(keymap-global-set "M-S-r" #'rotate-windows-back) ; Emacs 31
```

5.8.2 Jump between windows

```
(use-package ace-window  
  :config  
  (setq aw-keys '(?a ?o ?e ?u ?h ?t ?n ?s))  
  :bind  
  ("M-o" . 'ace-window)  
  ("C-x o" . 'ace-window)  
  )
```

5.8.3 Focus on the current window

From <https://emacsredux.com/blog/2026/04/07/stealing-from-the-best-emacs-configs/>

```
;; ##autoload  
(defun sacha-toggle-delete-other-windows ()  
  "Delete other windows in frame if any, or restore previous window config.  
  
  From https://emacsredux.com/blog/2026/04/07/stealing-from-the-best-emacs-configs/  
  and Steve Purcell's configuration."  
  (interactive)  
  (if (and winner-mode  
        (equal (selected-window) (next-window)))  
      (winner-undo)  
      (delete-other-windows)))
```

```
(keymap-global-set "C-x 1" #'sacha-toggle-delete-other-windows)
```

5.8.4 Winner mode - undo and redo window configuration

`winner-mode` lets you use `C-c <left>` and `C-c <right>` to switch between window configurations. This is handy when something has popped up a buffer that you want to look at briefly before returning to whatever you were working on. When you're done, press `C-c <left>`.

```
(use-package winner
  :init
  (winner-mode 1))
```

5.9 TODO Get the hang of using vundo

```
(use-package vundo)
```

<https://mastodon.online/@arialdo/113282285763018537>

5.10 Get scroll-other-window to work with PDFs

Thanks to [Reddit: How to navigate to previous/next PDF page in DocView from another window split without moving into it?](#)

```
(use-package scroll-other-window
  :vc (:url "https://gist.github.com/politza/3f46785742e6e12ba0d1a849f853d0b9")
  :commands sow-mode
  :init (sow-mode 1))
```

5.11 Manage positions

5.11.1 Save place between sessions

Based on [Stealing from the Best Emacs Configs | Emacs Redux](#)

```
(save-place-mode 1)
(advice-add 'save-place-find-file-hook
  :after (lambda (&rest _) (when buffer-file-name (ignore-errors (recenter))))))
```

5.11.2 Quickly jump to positions

embark

Quickly jump to a position in the current view.

- <https://karthinks.com/software/avy-can-do-anything/>
- https://www.reddit.com/r/emacs/comments/r6px3r/avy_can_do_anything_youre_using_avy_wrong/

```
;; ##autoload
(defun avy-action-exchange (pt)
  "Exchange sexp at PT with the one at point."
  (set-mark pt)
  (transpose-sexps 0))
;; ##autoload
(defun avy-action-embark (pt)
  (save-excursion
    (goto-char pt)
    (embark-act))
  (select-window
    (cdr (ring-ref avy-ring 0)))
  t)
```

```

;;###autoload
(defun sacha-avy-action-copy-whole-line (pt)
  "From https://karthink.com/software/avy-can-do-anything/"
  (save-excursion
    (goto-char pt)
    (cl-destructuring-bind (start . end)
      (bounds-of-thing-at-point 'line)
      (copy-region-as-kill start end)))
  (select-window
   (cdr
    (ring-ref avy-ring 0)))
  t)

;;###autoload
(defun sacha-avy-action-yank-whole-line (pt)
  "From https://karthink.com/software/avy-can-do-anything/"
  (sacha-avy-action-copy-whole-line pt)
  (save-excursion (yank))
  t)

```

```

(use-package avy
  :if sacha-laptop-p
  :config
  (add-to-list 'avy-dispatch-alist '(?e . avy-action-exchange))
  (setf (alist-get ?. avy-dispatch-alist) 'avy-action-embark)
  (setf (alist-get ?Y avy-dispatch-alist) 'sacha-avy-action-copy-whole-line)
  (setf (alist-get ?y avy-dispatch-alist) 'sacha-avy-action-yank-whole-line)
  (setf (alist-get ?w avy-dispatch-alist) 'avy-action-copy)
  (with-eval-after-load 'lispy
    (keymap-set lispy-mode-map
                 "M-j" #'avy-goto-char-timer))

  :bind
  ("M-j" . avy-goto-char-timer)
  )

(use-package avy-zap
  :if sacha-laptop-p
  :config
  (setq avy-zap-forward-only t)
  (setq avy-keys '(?a ?o ?e ?u ?i ?d ?h ?t ?n ?s))
  :bind
  (("M-z" . avy-zap-up-to-char-dwim)
   ("M-Z" . avy-zap-to-char-dwim)))

```

5.12 TODO Sort files in read-file-name

Updated version of <https://emacs.stackexchange.com/questions/55502/list-files-in-directory-in-reverse-order-of-date> - added cycle-sort-function and sorting by modification date.

```

(defcustom file-name-completions-sort-function #'files-sort-modification-time
  "Function for sorting the completion list of file names.
The function takes the list of file names as argument
and returns the sorted list."
  :type '(choice (function :tag "Sort Function") (const :tag "Natural Order" nil))
  :group 'files)

```

```

;;;###autoload
(defun files-sort-access-time (files)
  "Sort FILES list with respect to access time."
  (sort
   files
   (lambda (fn1 fn2)
     (time-less-p
      (file-attribute-access-time (file-attributes fn2))
      (file-attribute-access-time (file-attributes fn1))))))

;;;###autoload
(defun files-sort-modification-time (files)
  "Sort FILES list with respect to modification time."
  (sort
   files
   :key (lambda (f) (file-attribute-modification-time (file-attributes f)))
   :lessp #'time-less-p
   :reverse t))

;;;###autoload
(defun ad-completion-file-name-table (fun string pred action)
  "Add 'display-sort-function' to metadata.
If the completion action is metadata, add
`file-name-completions-sort-function' as display-sort-function.
Otherwise call FUN with STRING, PRED and ACTION as arguments."
  (if (and (functionp file-name-completions-sort-function)
           (eq action 'metadata))
      `(metadata
         (category . file)
         (cycle-sort-function . identity)
         (display-sort-function . ,file-name-completions-sort-function))
      (funcall fun string pred action)))

(advice-add 'completion-file-name-table :around #'ad-completion-file-name-table)

```

5.13 Downloaded files

```

(defvar sacha-download-dir "~/Downloads")
;;;###autoload
(defun sacha-open-latest-download ()
  (interactive)
  (find-file (sacha-latest-file sacha-download-dir)))

;;;###autoload
(defun sacha-attach-and-link-latest-download ()
  (interactive)
  (org-attach-attach (sacha-latest-file sacha-download-dir) nil 'cp)
  (org-insert-link nil (caar org-stored-links)))

;;;###autoload
(defun sacha-link-latest-download ()
  (interactive)
  (org-insert-link nil (concat "file:" (sacha-latest-file sacha-download-dir))
                    ↵
                    ↵ (file-name-nondirectory (sacha-latest-file sacha-download-dir))))

```

```
;; ###autoload
(defun sach Include-latest-download ()
  (interactive)
  (sach-insert-file-as-org-include (sach-latest-file sach-download-dir)))

;; ###autoload
(defun sach-copy-latest-download (dest &optional force)
  (interactive "FDestination: ")
  (copy-file (sach-latest-file sach-download-dir) dest force))
;; ###autoload
(defun sach-download-dired ()
  (interactive)
  (dired sach-download-dir "-lt"))
```

5.13.1 Replace with latest download

```
;; ###autoload
(defun sach-replace-with-latest-download ()
  "Replace file contents with latest download."
  (interactive)
  (widen)
  (erase-buffer)
  (insert-file-contents (sach-latest-file sach-download-dir)))
```

5.14 Searching

<https://emacsredux.com/blog/2026/03/15/isearch-lazy-count/>

```
(setopt isearch-lazy-count t)
```

I should get the hang of using `helm-org-rifle` and `ripgrep`.

```
;; ###autoload
(defun sach-helm-org-rifle-org-directory ()
  (interactive)
  (helm-org-rifle-directories (list org-directory) t))
;; ###autoload
(defun sach-consult-recoll-without-emacs-news ()
  (interactive)
  (consult-recoll--open (consult-recoll--search "-\\\"Emacs News\\\" ")))
```

```
(use-package helm-org-rifle
  :bind
  ("M-g r r" . helm-org-rifle)
  ("M-g r a" . helm-org-rifle-org-agenda-files)
  ("M-g r o" . helm-org-rifle-org-directory)
)

(use-package consult-recoll
  :config
  (setq consult-recoll-search-flags nil)
  :bind
  ("M-s S" . consult-recoll))
```

5.14.1 Deleting things

From Steve Purcell, who linked to <http://www.emacswiki.org/emacs/ZapToISearch>

```
;;###autoload
(defun zap-to-isearch (rbeg rend)
  "Kill the region between the mark and the closest portion of
  the isearch match string. The behaviour is meant to be analogous
  to zap-to-char; let's call it zap-to-isearch. The deleted region
  does not include the isearch word. This is meant to be bound only
  in isearch mode. The point of this function is that oftentimes
  you want to delete some portion of text, one end of which happens
  to be an active isearch word. The observation to make is that if
  you use isearch a lot to move the cursor around (as you should,
  it is much more efficient than using the arrows), it happens a
  lot that you could just delete the active region between the mark
  and the point, not include the isearch word."
  (interactive "r")
  (when (not mark-active)
    (error "Mark is not active"))
  (let* ((isearch-bounds (list isearch-other-end (point)))
         (ismin (apply 'min isearch-bounds))
         (ismax (apply 'max isearch-bounds))
         )
    (if (< (mark) ismin)
        (kill-region (mark) ismin)
      (if (> (mark) ismax)
          (kill-region ismax (mark))
        (error "Internal error in isearch kill function.")))
    (isearch-exit)
  ))

(define-key isearch-mode-map [(meta z)] 'zap-to-isearch)
```

5.14.2 Transient for isearch

From <https://github.com/kickingvegas/cclisp/blob/fae13b5adb6cb667af23070d000f9bd91b6ba3d8/cc-isearch-menu.el#L96>

```
(require 'transient)
(transient-define-prefix cc/isearch-menu ()
  "isearch Menu"
  ["Edit Search String"
   ("e"
    "Edit the search string (recursive)"
    isearch-edit-string
    :transient nil)
   ("w"
    "Pull next word or character word from buffer"
    isearch-yank-word-or-char
    :transient nil)
   ("s"
    "Pull next symbol or character from buffer"
    isearch-yank-symbol-or-char
    :transient nil)
   ("l"
    "Pull rest of line from buffer"
    isearch-yank-line
```

```

      :transient nil)
    ("y"
     "Pull string from kill ring"
     isearch-yank-kill
     :transient nil)
    ("t"
     "Pull thing from buffer"
     isearch-forward-thing-at-point
     :transient nil)]

["Replace"
 ("q"
  "Start 'query-replace"
  isearch-query-replace
  :if-nil buffer-read-only
  :transient nil)
 ("x"
  "Start 'query-replace-regexp"
  isearch-query-replace-regexp
  :if-nil buffer-read-only
  :transient nil)]]

[[ "Toggle"
  ("X"
   "Regex searching"
   isearch-toggle-regexp
   :transient nil)
  ("S"
   "Symbol searching"
   isearch-toggle-symbol
   :transient nil)
  ("W"
   "Word searching"
   isearch-toggle-word
   :transient nil)
  ("F"
   "Case fold"
   isearch-toggle-case-fold
   :transient nil)
  ("L"
   "Lax whitespace"
   isearch-toggle-lax-whitespace
   :transient nil)]

["Misc"
 ("o"
  "occur"
  isearch-occur
  :transient nil)
 ("h"
  "highlight"
  isearch-highlight-regexp
  :transient nil)
 ("H"
  "highlight lines"
  isearch-highlight-lines-matching-regexp
  :transient nil)]]

(define-key isearch-mode-map (kbd "M-S") 'cc/isearch-menu)

```


5.14.3 Search invisible text

```
(setq isearch-invisible t
      search-invisible t)
```

5.14.4 Occur

From <https://emacs.ch/@bram85/111724372485640053>:

```
(with-eval-after-load 'occur
  (keymap-set occur-mode-map "C-x C-q" #'occur-edit-mode))
```

5.15 Ediff

<http://yumymelon.com/devnull/surprise-and-emacs-defaults.html>

```
(setq ediff-split-window-function 'split-window-horizontally)
(setq ediff-window-setup-function 'ediff-setup-windows-plain)
(add-hook 'ediff-before-setup-hook #'sacha-store-pre-ediff-winconfig)
(add-hook 'ediff-quit-hook #'sacha-restore-pre-ediff-winconfig)
```

```
(defvar sacha-ediff-last-windows nil)

;;;###autoload
(defun sacha-store-pre-ediff-winconfig ()
  "Store `current-window-configuration' in variable `sacha-ediff-last-windows'."
  (setq sacha-ediff-last-windows (current-window-configuration)))

;;;###autoload
(defun sacha-restore-pre-ediff-winconfig ()
  "Restore window configuration to stored value in `sacha-ediff-last-windows'."
  (set-window-configuration sacha-ediff-last-windows))
```

5.16 Hideshow

From <https://karthinks.com/software/simple-folding-with-hideshow/> :

```
(use-package hideshow
  :hook
  (prog-mode . hs-minor-mode)
  :bind
  ("C-<tab>" . hs-cycle)
  ("C-<iso-lefttab>" . hs-global-cycle)
  ("C-S-<tab>" . hs-global-cycle))
```

```
;;;###autoload
(defun hs-cycle (&optional level)
  (interactive "p")
  (let (message-log-max
```

```

(inhibit-message t))
(if (= level 1)
  (pcase last-command
    ('hs-cycle
     (hs-hide-level 1)
     (setq this-command 'hs-cycle-children))
    ('hs-cycle-children
     ;; TODO: Fix this case. `hs-show-block' needs to be
     ;; called twice to open all folds of the parent
     ;; block.
     (save-excursion (hs-show-block))
     (hs-show-block)
     (setq this-command 'hs-cycle-subtree))
    ('hs-cycle-subtree
     (hs-hide-block))
    (_
     (if (not (hs-already-hidden-p))
         (hs-hide-block)
         (hs-hide-level 1)
         (setq this-command 'hs-cycle-children))))
  (hs-hide-level level)
  (setq this-command 'hs-hide-level))))

;;;###autoload
(defun hs-global-cycle ()
  (interactive)
  (pcase last-command
    ('hs-global-cycle
     (save-excursion (hs-show-all))
     (setq this-command 'hs-global-show))
    (_ (hs-hide-all))))

```

5.17 Pop to mark

Handy way of getting back to previous places.

```

(keymap-global-set "C-x p" 'pop-to-mark-command)
(setq set-mark-command-repeat-pop t)

```

5.18 Helm-swoop - quickly finding lines

This promises to be a fast way to find things. Let's bind it to **Ctrl-Shift-S** to see if I can get used to that...

```

(use-package helm-swoop
  :if sachal-laptop-p
  :bind
  (("C-S-s" . helm-swoop)
   ("M-i" . helm-swoop)
   ("M-s M-s" . helm-swoop)
   ("M-I" . helm-swoop-back-to-last-point)
   ("C-c M-i" . helm-multi-swoop)
   ("C-x M-i" . helm-multi-swoop-all)
  )
  :config
  (progn

```

```
(define-key isearch-mode-map (kbd "M-i") 'helm-swoop-from-isearch)
(define-key helm-swoop-map (kbd "M-i") 'helm-multi-swoop-all-from-helm-swoop))
)
```

5.19 Highlight the current line while still being able to easily customize/describe underlying faces

I use `hl-line-mode` to highlight the current line.

```
(add-hook 'prog-mode-hook 'hl-line-mode)
(add-hook 'text-mode-hook 'hl-line-mode)
```

However, I don't want `hl-line` to interfere with the default face suggested by `customize-face`, which is returned by `face-at-point`.

```
(defun sachas-suggest-other-faces (func &rest args)
  "Disable `hl-line-mode' when choosing a face."
  (if hl-line-mode
      (progn
        (hl-line-mode -1)
        (progn (apply func args)
               (hl-line-mode 1)))
      (apply func args)))
```

```
(advice-add #'face-at-point :around #'sachas-suggest-other-faces)
```

Now I can use `customize-face` and `describe-face` without `hl-line` interfering all the time.

5.20 Windmove - switching between windows

Windmove lets you move between windows with something more natural than cycling through `C-x o` (`other-window`). Windmove doesn't behave well with Org, so we need to use different keybindings.

```
(use-package windmove
  :bind
  ((("<f2> <right>" . windmove-right)
    ("<f2> <left>" . windmove-left)
    ("<f2> <up>" . windmove-up)
    ("<f2> <down>" . windmove-down)
  ))
```

5.21 Frequently-accessed files

Registers allow you to jump to a file or other location quickly. To jump to a register, use `C-x r j` followed by the letter of the register. Using registers for all these file shortcuts is probably a bit of a waste since I can easily define my own keymap, but since I rarely go beyond register A anyway. Also, I might as well add shortcuts for refiling.

```
(setq bookmark-watch-bookmark-file 'silent)
(setq sachas-file-shortcuts
  '(("C" "~/proj/emacs-calendar/README.org" "Emacs calendar")
    ("e" "~/sync/emacs/Sacha.org" "Config"))
```

```

("E" "~/sync/emacs-news/index.org" "Emacs News")
("f" "~/sync/orgzly/journal-fr.org" "French journal")
("F" "~/sync/orgzly/french.org" "French")
("I" "~/sync/orgzly/computer-inbox.org" "Computer inbox")
("i" "~/sync/orgzly/Inbox.org" "Phone inbox")
("o" "~/sync/orgzly/organizer.org" "Main org file")
("s" "~/proj/stream/index.org" "Yay Emacs")
("b" "~/sync/orgzly/business.org" "Business")
("P" "~/ssh:web:/home/sacha/planet/data/feeds.json" "Planet Emacsen")
("p" "~/sync/orgzly/posts.org" "Posts")
("m" "~/sync/web/beginner-map.org" "Map")
("n" "~/sync/topics/now.org" "Now")
("N" "~/ssh:web|sudo::/etc/nginx/sites-available" "Nginx sites")
("w" "~/sync/topics/workflows.org" "Workflows")
("W" "~/Dropbox/public/sharing/blog.org" "Blog index")
("1" "~/proj/static-blog/" "Static blog")
("r" "~/sync/orgzly/reference.org" "Reference")
("R" "~/personal/reviews.org" "Reviews")
("v" "~/proj/emacsstv.github.io/videos.org" "Videos")
("g" "~/proj/sachac.github.io/evil-plans/index.org" "Evil plans"))

```

```
(sacha-navigate-set-up-file-shortcuts)
```

```

(require 'bookmark)
(defvar sacha-org-refile-map (make-sparse-keymap))

(defmacro sacha-defshortcut (key file &optional label)
  `(progn
    (set-register ,(string-to-char key) (cons 'file ,file))
    (bookmark-store ,file (list (cons 'filename ,file)
                                (cons 'position 1)
                                (cons 'front-context-string "")) nil)
    (define-key sacha-org-refile-map
      ,key
      ,(if label
        `(cons ,label
              (lambda (prefix)
                (interactive "p")
                (let ((org-refile-targets '(((,file) :maxlevel . 6)))
                    (current-prefix-arg (or current-prefix-arg '(4))))
                  (call-interactively 'org-refile))))
        `(lambda (prefix)
          (interactive "p")
          (let ((org-refile-targets '(((,file) :maxlevel . 6)))
              (current-prefix-arg (or current-prefix-arg '(4))))
            (call-interactively 'org-refile)))))))

(defmacro defshortcuts (name body &optional docstring &rest heads)
  (declare (indent defun) (doc-string 3))
  (cond ((stringp docstring))
        (t
         (setq heads (cons docstring heads))
         (setq docstring "")))
  ;; unwrap
  (when (and (= (length heads) 1) (stringp (car (car (car heads))))))
    (setq heads (car heads)))
  (list
   'progn

```

```

      (cons 'progn
        (mapcar (lambda (h) (list 'sacha-defshortcut (string-to-char (elt h 0)) (elt h 1)))
          heads))))

(defmacro defshortcuts+ (name body &optional docstring &rest heads)
  (declare (indent defun) (doc-string 3))
  (cond ((stringp docstring))
    (t
     (setq heads (cons docstring heads))
     (setq docstring "")))
  (list
   'progn
   (append `(defhydra+ ,name (:exit t))
     (mapcar (lambda (h)
       (list (elt h 0) (list 'find-file (elt h 1)) (elt h 2)))
       heads))
   (cons 'progn
     (mapcar (lambda (h) (list 'sacha-defshortcut (string-to-char (elt h 0)) (elt h 1)))
       heads))))

(defvar sacha-file-shortcuts nil
  "List of ((character filename label) ...).")

;;;###autoload
(defun sacha-navigate-set-up-file-shortcuts ()
  (interactive)
  (mapcar (lambda (o)
    (eval `(sacha-defshortcut ,(elt o 0) ,(elt o 1)))
    sacha-file-shortcuts))

```

Let's borrow Prot's way of working with registers too.

```

(use-package prot-register
  :load-path "~/vendor/prot-dotfiles/emacs/.emacs.d/prot-lisp"
  :commands (prot-register-add-dwim)
  :bind
  (("C-, a" . prot-register-add-dwim)))

```

5.22 C-g improvement

From <https://protesilaos.com/codelog/2024-11-28-basic-emacs-configuration/>

```

;;;###autoload
(defun prot/keyboard-quit-dwim ()
  "Do-What-I-Mean behaviour for a general `keyboard-quit'."

  The generic `keyboard-quit' does not do the expected thing when
  the minibuffer is open. Whereas we want it to close the
  minibuffer, even without explicitly focusing it.

  The DWIM behaviour of this command is as follows:

  - When the region is active, disable it.
  - When a minibuffer is open, but not focused, close the minibuffer.
  - When the Completions buffer is selected, close it.
  - In every other case use the regular `keyboard-quit'."
  (interactive)

```

```
(cond
  ((region-active-p)
   (keyboard-quit))
  ((derived-mode-p 'completion-list-mode)
   (delete-completion-window))
  ((> (minibuffer-depth) 0)
   (abort-recursive-edit))
  (t
   (keyboard-quit))))

(define-key global-map (kbd "C-g") #'prot/keyboard-quit-dwim)
```

5.23 Smartscan

From <https://github.com/itsjeyd/emacs-config/blob/emacs24/init.el>, this makes **M-n** and **M-p** look for the symbol at point.

```
(use-package smartscan
  :if sachal-laptop-p
  :defer t
  :config (global-smartscan-mode t))
```

5.24 Dired

```
(setq dired-listing-switches "-altr")
(setq dired-dwim-target 'dired-dwim-target-next)
```

From <http://www.masteringemacs.org/articles/2011/03/25/working-multiple-files-dired/>

```
(require 'find-dired)
(setq find-ls-option '("-print0 | xargs -0 ls -ld" . "-ld"))
```

From <https://protesilaos.com/codelog/2024-11-28-basic-emacs-configuration/>

```
(use-package dired-subtree
  :ensure t
  :after dired
  :bind
  ( :map dired-mode-map
    ("" . dired-subtree-toggle)
    ("TAB" . dired-subtree-toggle)
    ("" . dired-subtree-remove)
    ("S-TAB" . dired-subtree-remove))
  :config
  (setq dired-subtree-use-backgrounds nil))
```

5.24.1 Using image-dired to browse the latest screenshots from multiple directories

[2025-01-06 Mon]: [Patch in progress](#), Stefan Kangas is looking into it.

Since A+ and I play lots of Minecraft together, I figured it's a good opportunity to slowly get her into the idea of documenting learning. Besides, I can always practise it myself. Screenshots are

handy for that. In Minecraft Java, **F1** hides the usual heads-up display, and **F2** takes the screenshot. Usually, when I start taking screenshots. A+ starts taking screenshots too. I want to build on her enthusiasm by including her screenshots in notes. To make it easy to incorporate her pictures into our notes, I've shared her [GDLauncher](#) folder and her Videos folder with my computer using [Syncthing](#) so that I can grab any screenshots or videos that she's taken.

In Emacs, [image-dired](#) makes it easy to see thumbnails. The neat thing is that it doesn't just work with a single directory. Just like Dired, you can give it a cons cell with a directory in the first part and a list of files in the second part as the first argument to the function, and it will display those files. This means I can use [directory-files-recursively](#) to make a list of files, sort it to show most recent screenshots first, limit it to the most recent items, and then display a buffer with thumbnails. [image-dired-show-all-from-dir](#) reports a small error when you do this (I need to send a patch [upstream](#)), so we hush it with [condition-case](#) in [sacha-show-combined-screenshots](#).

```
(defvar sacha-screenshot-dirs
  '("~/recordings"
    "~/var/app/org.prismlauncher.PrismLauncher/data/PrismLauncher/instances/"
    "~/sync/gdlauncher-instances/"
  ))

(defvar sacha-recent-screenshot-limit 50)

;; ##autoload
(defun sacha-combined-screenshots (&optional limit)
  (seq-take
   (sort
    (seq-mapcat (lambda (dir)
      ↪ (directory-files-recursively dir ↪
      ↪ "[0-9][0-9][0-9][0-9]-.*\\.\\(png\\|webm\\|gif\\|svg\\|mkv\\)")
      sacha-screenshot-dirs)
    :key (lambda (o) (file-attribute-modification-time (file-attributes o)))
    :reverse t)
   (or limit sacha-recent-screenshot-limit)))

;; ##autoload
(defun sacha-latest-screenshot ()
  (car (sacha-combined-screenshots)))

;; ##autoload
(defun sacha-show-combined-screenshots (&optional limit)
  "Show thumbnails for combined screenshots."
  (interactive (list (when current-prefix-arg (read-number "Limit: "))))
  (condition-case nil
    ;; ignore errors from image-dired trying to set default-directory
    (image-dired-show-all-from-dir
     (cons (car sacha-screenshot-dirs) (sacha-combined-screenshots limit)))
    (error nil)))
```

In the `*image-dired*` buffer created by `sacha-show-combined-screenshots`, I can use `m` (`image-dired-mark-thumb-original-file`) to mark images and `C-u w` (`image-dired-copy-filename-as-kill`) to copy their absolute paths.

To make it easier to create links to a file by using `org-store-link` (which I've bound to `C-c l`) and `org-insert-link` (`C-c C-l` in an Org buffer), I can define a link-storing function that takes the original filename:

```
;; ##autoload
(defun sacha-org-image-dired-store-link ())
```

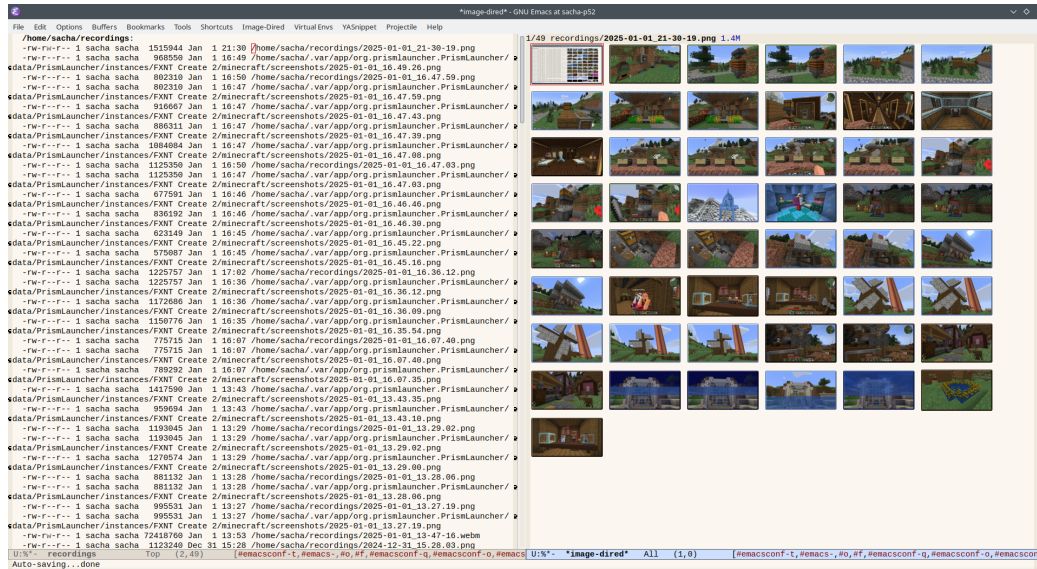


Figure 7: The result of `sacha-show-combined-screenshots`

```
(when (and (derived-mode-p 'image-dired-thumbnail-mode)
  (get-text-property (point) 'original-file-name))
  (org-link-store-props
   :link (concat "file:" (get-text-property (point) 'original-file-name))))
```

```
(with-eval-after-load 'org
  (org-link-set-parameters
   "image-dired"
   :store #'sacha-org-image-dired-store-link))
```

To make it easier to insert the marked links that have been copied as absolute paths:

```
;;###autoload
(defun sacha-org-yank-file-links-from-kill-ring ()
  (interactive)
  (dolist (file (read (concat "(" (current-kill 0) ")")))
    (insert (org-link-make-string (concat "file:" file)) "\n")))
```

I usually want to copy those files to another directory anyway. I have a `sacha-org-copy-linked-files` function in [Copy linked file and change link](#) (p. 295) that copies the files and rewrites the Org links. This means that I can copy my notes to an index.org in a directory I share with A+, save the images to an `images` subdirectory, and export the index.html so that she can read the notes any time she likes.

5.24.2 peep-dired

Allow my use of `C-x C-q` while in peep-dired mode.

```
(use-package peep-dired
  :if sacha-laptop-p
  :bind (:map peep-dired-mode-map
    ("SPC" . nil)
    ("<backspace>" . nil)))
```


5.24.3 Saving photos

```
; ; ##autoload
(defun sach-save-photo (name)
  (interactive "MName: ")
  (let* ((file (dired-get-filename))
         new-name)
    (cond
     ((string-match ↵
      ↵ "CameraZOOM-\\([0-9][0-9][0-9][0-9]\\)\\([0-9][0-9]\\)\\([0-9][0-9]\\)\\([0-9][0-9]\\)\\([0-9][0-9]\\)\\([0-9][0-9]\\)\\([0-9][0-9]\\)"
      file)
      (setq new-name
            (format "%s-%s-%s %s.%s.%s.%s %s.jpg"
                    (match-string 1 file)
                    (match-string 2 file)
                    (match-string 3 file)
                    (match-string 4 file)
                    (match-string 5 file)
                    (match-string 6 file)
                    (match-string 7 file)
                    name)))
     ((string-match ↵
      ↵ "\\([0-9][0-9][0-9][0-9]\\)[\\.\\-]\\([0-9][0-9]\\)[\\.\\-]\\([0-9][0-9]\\)[\\.\\-]↵
      ↵ \\([0-9][0-9]\\)\\.\\([0-9][0-9]\\)\\.\\([0-9][0-9]\\)" file)
      (setq new-name
            (format "%s-%s-%s %s.%s.%s %s.jpg"
                    (match-string 1 file)
                    (match-string 2 file)
                    (match-string 3 file)
                    (match-string 4 file)
                    (match-string 5 file)
                    (match-string 6 file)
                    name)))
     (t (setq new-name (concat (file-name-sans-extension (file-name-nondirectory file)) " " ↵
                               ↵ name ".jpg"))))
    (when (string-match "A-" name)
      (copy-file file (expand-file-name new-name sach-kid-photo-directory)))
    (rename-file file (expand-file-name new-name "~/archives/2016/photos/selected/"))))
; ; ##autoload
(defun sach-backup-media ()
  (interactive)
  (mapcar (lambda (file)
            (rename-file
             file
             (expand-file-name
              (file-name-nondirectory file)
              (cond
               ((string-match "mp4" file) "~/archives/2016/videos/")
               ((string-match "mp3\\\\wav" file) "~/archives/2016/audio/")
               (t "~/archives/2016/photos/backup/"))))))
           (dired-get-marked-files)))
```

```
(with-eval-after-load 'dired
  (keymap-set dired-mode-map "b" 'sacha-save-photo)
  (keymap-set dired-mode-map "r" 'sacha-backup-media))
```

5.25 Move to beginning of line

Copied from <http://emacsredux.com/blog/2013/05/22/smarter-navigation-to-the-beginning-of-a-line/>

```
(defun sacha-smarter-move-beginning-of-line (arg)
  "Move point back to indentation of beginning of line.

  Move point to the first non-whitespace character on this line.
  If point is already there, move to the beginning of the line.
  Effectively toggle between the first non-whitespace character and
  the beginning of the line.

  If ARG is not nil or 1, move forward ARG - 1 lines first. If
  point reaches the beginning or end of the buffer, stop there."
  (interactive "^p")
  (setq arg (or arg 1))

  ;; Move lines first
  (when (≠ arg 1)
    (let ((line-move-visual nil))
      (forward-line (1- arg))))

  (let ((orig-point (point)))
    (back-to-indentation)
    (when (= orig-point (point))
      (move-beginning-of-line 1))))
```

```
;; remap C-a to `smarter-move-beginning-of-line'
(global-set-key [remap move-beginning-of-line]
  'sacha-smarter-move-beginning-of-line)
```

5.26 Recent files

```
(require 'recentf)
(setq recentf-max-saved-items 200
      recentf-max-menu-items 15)
(recentf-mode)
```

5.27 Open files externally

Copied from Prelude: <http://emacsredux.com/blog/2013/03/27/open-file-in-external-program/>

```
(defun sacha-open-with (arg)
  "Open visited file in default external program.

  With a prefix ARG always prompt for command to use."
  (interactive "P")
  (when buffer-file-name
    (shell-command (concat
                     (cond
                      ((and (not arg) (eq system-type 'darwin)) "open")
                      ((and (not arg) (member system-type '(gnu gnu/linux gnu/kfreebsd))) ↵
                     "xdg-open"))
```

```
(t (read-shell-command "Open current file with: "))
" "
(shell-quote-argument buffer-file-name))))
```

Don't use docview for PDFs.

```
(with-eval-after-load 'org
  (add-to-list 'org-file-apps '("pdf" . "evince %s")))
```

5.28 Emacs: Toggle a buffer for quick reference

I want to quickly bring up some notes and then tuck them away again. I took the idea from [this Reddit thread](#) and modified it so that I can specify a function name for later use and filenames for easy reference.

```
;;###autoload
(defun sacha-toggle-or-create (buffer-name &optional buffer-create-fn switch-cont)
  "Raises or hides BUFFER-NAME.
If BUFFER-CREATE-FN is specified, use that when creating a function.
Use SWITCH-CONT for additional setup if needed."
  (interactive)
  (let ((target-buf
        (if (file-exists-p buffer-name)
            (find-file-noselect buffer-name)
            (get-buffer buffer-name))))
    (cond
     ((equal (current-buffer) target-buf)
      (bury-buffer))
     (target-buf
      (switch-to-buffer target-buf)
      (when switch-cont (funcall switch-cont)))
     (t (if buffer-create-fn
            (funcall buffer-create-fn)
            (switch-to-buffer
             (get-buffer-create buffer-name)))
      (when switch-cont (funcall switch-cont))))))

;;###autoload
(defmacro sacha-make-toggle-buffer-function (function-name buffer-name &optional ↵
  ↵ buffer-create-fn switch-cont)
  "Makes a toggle-function to have raise-or-create behaviour.

Creates a toggle-function that executes BUFFER-CREATE-FN if a
buffer named BUFFER-NAME doesn't exist, switches to the buffer
named BUFFER-NAME if it exists, and switches to the previous
buffer if we are currently visiting buffer BUFFER-NAME.

The SWITCH-CONT argument is a function which, if given, is called
after the buffer has been created or switched to. This allows
running further actions that setup the state of the buffer or
modify it.

Based on ↵
↵ https://www.reddit.com/r/emacs/comments/l4v1ux/one_of_the_most_useful_small_lisp_functions_in_my/"
  (declare (debug t))
  `(defun ,function-name ()
    ,(format "Toggle %s." buffer-name)
```

```
(interactive)
(sacha-toggle-or-create
 ,buffer-name
 ,buffer-create-fn
 ,switch-cont)))
```

Now for some shortcuts to quickly bring up my livestream notes, my current focus, or a shell.

```
(keymap-global-set
 "C-z"
 (sacha-make-toggle-buffer-function
  sacha-toggle-live
  "~/sync/topics/live.org"))
(keymap-global-set
 "C-S-z"
 (sacha-make-toggle-buffer-function
  sacha-toggle-now
  "~/sync/topics/now.org"))
(keymap-global-set
 "C-x C-z"
 (sacha-make-toggle-buffer-function
  sacha-toggle-eshell
  "*eshell*"
  (lambda () (eshell) (end-of-buffer))
  (lambda ()
   (eshell/cd default-directory)
   (end-of-buffer)))))
```

5.29 link-hint

This should make it easier to jump to a link.

```
(use-package link-hint
 :bind
 ("M-g u" . link-hint-open-link)
 ("M-g U" . link-hint-open-multiple-links))
```

5.30 Bookmarks

<http://yumymelon.com/devnull/using-bookmarks-in-emacs-like-you-do-in-web-browsers.html>

```
(easy-menu-define cc/bookmarks-menu nil
 "Keymap for CC Bookmarks Menu"
 '("Bookmarks"
  ["Edit Bookmarks" list-bookmarks
   :help "Display a list of existing bookmarks."]
  ["--" nil]
  ["Add ...Bookmark" bookmark-set-no-overwrite
   :help "Set a bookmark named NAME at the current location."]
  ["—" nil]
  ["Jump to ...Bookmark" bookmark-jump
   :help "Jump to bookmark"]]))
(easy-menu-add-item global-map '(menu-bar)
                    cc/bookmarks-menu
                    "Tools")
```

```
(defhydra+ sacha-shortcuts (:exit t)
  ("b" bookmark-jump "Jump to bookmark")
  ("B" bookmark-set-no-overwrite "Set bookmark"))
```

5.31 Dogears

<https://github.com/alphapapa/dogears.el>

```
(use-package dogears
  ;; These bindings are optional, of course:
  :bind (:map global-map
    ("M-g d" . dogears-go)
    ("M-g M-b" . dogears-back)
    ("M-g M-f" . dogears-forward)
    ("M-g M-d" . dogears-list)
    ("M-g M-D" . dogears-sidebar)))
```

5.32 Randomness for serendipity

```
;;;###autoload
(defun sacha-goto-random-char ()
  (interactive)
  (goto-char (random (point-max))))
```

5.32.1 Building a today-I-learned habit, and displaying the documentation for random Emacs commands emacs

I'd like to build a habit of regularly learning one small thing each day in one of three domains: tech, life, and learning. My measurable output would probably be in the form of index cards, tweets, blog posts, and notes (in org-capture, Dropbox, or Evernote). I can get input from various sources like blog posts, videos, books, webpages, and so on.

A little bit of randomness might be useful for learning more about Emacs. Emacswiki has a [random page](#) function, but the chunks are often a little large or irrelevant. On the other hand, displaying a random command from the packages that I already have loaded into my Emacs - that might be a good way to discover interesting things.

I started by looking at `apropos-command`, which led me to `apropos-internal`, which is a C function that referred to `obarray`. Using `obarray` by itself didn't work (suspiciously few elements, so I often ended up looking at emms-related functions). I eventually found `mapatoms`, which seems to do a better job at listing an appreciable number of interactive functions. I filtered the list to include only documented functions that had not been marked as obsolete: 8,415 in my current Emacs, which should be plenty to go through. =)

```
;;;###autoload
(defun sacha-describe-random-interactive-function ()
  (interactive)
  "Show the documentation for a random interactive function.
Consider only documented, non-obsolete functions."
  (let (result)
    (mapatoms
     (lambda (s)
       (when (and (commandp s)
                  (documentation s t))
         (documentation s t))
```

```
(null (get s 'byte-obsolete-info)))
(setq result (cons s result))))
(describe-function (elt result (random (length result))))))
```

I've added this to a [key-chord](#) + [hydra](#) keymap as a repeatable function, so I can type `hh` to start my Hydra and then type `r` as many times as I want in order to show the documentation for a random interactive function. If you're curious about that, you can see the [key-chord section of my config](#).

Anyway, today I learned more about `obarray` and `mapatoms` - they're not interactive functions, but they were handy for building this little bit of code. We'll see how it goes! =)

5.32.2 Shuffling lines

```
;; ##autoload
(defun sach-shuffle-lines-in-region (beg end)
  "Randomize the order of lines from BEG to END."
  (interactive "r")
  (let ((list (split-string (buffer-substring beg end) "[\r\n]+")))
    (delete-region beg end)
    (insert (string-join (seq-sort-by (lambda (_) (random)) #'<= list) "\n"))))
```

5.33 Network: TRAMP and editing files over SSH

Emacs lets you edit files on remote servers, which is pretty darn cool. On Windows, these things help a little.

```
(when (eq system-type 'windows-nt)
  (setq tramp-default-method "plink")
  (setq tramp-auto-save-directory "c:\\sacha\\tmp"))
```

`tramp-hlo` is a new package that might improve TRAMP performance by using shell scripts instead of lots of little TRAMP calls.

```
(use-package tramp-hlo
  :ensure t
  :config
  (tramp-hlo-setup))
```

5.34 Touch gestures

[2024-02-01 Thu] Based on <https://kitchingroup.cheme.cmu.edu/blog/2014/08/31/Using-Mac-gestures-in-Emacs/>. This switches with a debouncing function, so you only change one at a time.

```
(defvar sach-navigate-swipe-debounce t "Non-nil means allow swiping.")

;; ##autoload
(defun sach-navigate-previous-buffer-debounced ()
  (interactive)
  (when sach-navigate-swipe-debounce
    (previous-buffer)
    (setq sach-navigate-swipe-debounce nil)
    (run-at-time "1 sec" nil (lambda () (setq sach-navigate-swipe-debounce t)))))

;; ##autoload
```

```
(defun sacha-navigate-next-buffer-debounced ()
  (interactive)
  (when sacha-navigate-swipe-debounce
    (next-buffer)
    (setq sacha-navigate-swipe-debounce nil)
    (run-at-time "1 sec" nil (lambda ()
                               (setq sacha-navigate-swipe-debounce t)))))
```

```
(keymap-global-set "<triple-wheel-right>" 'sacha-navigate-previous-buffer-debounced)
(keymap-global-set "<triple-wheel-left>" 'sacha-navigate-next-buffer-debounced)
```

5.35 Applications

```
;;;###autoload
(defun sacha-navigate-focus-or-launch (app &optional start)
  "Switch to the specified APP or START it."
  (interactive)
  (apply
   'call-process
   "~/bin/focus_or_launch" nil t nil
   (delq nil
          (list
            app start)))))
```

Ex: (sacha-navigate-focus-or-launch "obs")

6 Reading

```
(use-package pdf-tools
  :if sacha-laptop-p
  :config
  (pdf-tools-install)
  (setq pdf-view-resize-factor 1.1)
  (setq-default pdf-view-display-size 'fit-page)
  :defer t
)
```

7 Writing and editing

```
(keymap-global-set "M-c" #'sacha-capitalize-dwim)
(setq-default fill-column 70)
(keymap-global-set "M-o" #'join-line)
(keymap-global-set "M-T" #'transpose-sentences) ; ↵
↵ https://www.matem.unam.mx/~omar/apropos-emacs.html#writing-experience
```

<https://emacsredux.com/blog/2026/04/07/stealing-from-the-best-emacs-configs/>

```
(setq-default bidi-display-reordering 'left-to-right
  bidi-paragraph-direction 'left-to-right)
(setq bidi-inhibit-bpa t)
```

```
(setq redisplay-skip-fontification-on-input t)
(setq save-interprogram-paste-before-kill t)
(setq kill-do-not-save-duplicates t)
```

```
(defun sacha-capitalize-dwim ()
  "Capitalize the previous word if typing and at the end of a word."
  (interactive)
  (if (region-active-p)
      (capitalize-region (region-beginning) (region-end))
      (when (and (not (bolp))
                  (looking-back "\\w" 1)
                  (eq last-command 'self-insert-command))
          (backward-word))
        (capitalize-word 1)))

(defun sacha-copy-filename ()
  "Copy the current buffer file name to the clipboard."
  (interactive)
  (cond
   ((derived-mode-p 'dired-mode) (dired-copy-filename-as-kill 0))
   (t (kill-new (buffer-file-name)))))
```

```
;; Bind it to the original M-c key
(global-set-key (kbd "M-c") 'sacha-capitalize-dwim)
```

7.1 Learning French

french

```
(use-package learn-lang :load-path "~/proj/learn-lang"
  :preface (load "~/proj/learn-lang/learn-lang-autoloads.el" nil t)
  :config
  (setq learn-lang-language "fr")
  (setq learn-lang-tatoeba-files
        '(("fr" . "~/proj/french/tatoeba-fr-en.tsv")))
)
```

```
(setq search-default-mode 'char-fold-to-regexp)

(defvar-keymap sacha-learn-lang-map
  :prefix t
  "l" (cons "lookup" #'sacha-learn-lang-lexique-complete-word)
  "w" (cons "wordref" #'sacha-learn-lang-wordreference-lookup)
  "c" (cons "conj" #'sacha-learn-lang-conjugate)
  "f" (cons "→ fr" #'sacha-learn-lang-consult-en-fr)
  "s" (cons "say" #'sacha-learn-lang-say-word-at-point)
  "x" (cons "example" #'learn-lang-tatoeba-consult)
  "t" (cons "→ en" #'sacha-learn-lang-translate-dwim))

(with-eval-after-load 'org
  (keymap-set org-mode-map "C-," 'sacha-learn-lang-map)
  (keymap-set org-mode-map "C-c u" 'sacha-learn-lang-map))

(with-eval-after-load 'message
  (keymap-set message-mode-map "C-," 'sacha-learn-lang-map)
)
```



```
(with-eval-after-load 'flyspell
  (keymap-set flyspell-mode-map "C-", 'sacha-learn-lang-map))

;; (use-package wiktioary-bro
;;   :config
;;   (setq wiktioary-bro-language "fr")
;;   )

(use-package flycheck-grammalecte
  :config
  (setq flycheck-grammalecte-report-apos nil)
  (setq flycheck-grammalecte-report-nbsp nil)
  (setq flycheck-grammalecte-report-esp nil)
  (with-eval-after-load 'flycheck
    (flycheck-grammalecte-setup)))
```

```
;;;###autoload
(defun sacha-learn-lang-chrome-speech-new-session ()
  (interactive)
  (sacha-speech-chrome-new-session "french" "fr-FR"))
```

7.1.1 Practice pronunciation

```
(defvar sacha-learn-lang-practice-dir "~/proj/french/audio")
(defvar sacha-learn-lang-practice-temp (expand-file-name "temp.wav" ←
  ↪ sacha-learn-lang-practice-dir))
;;;###autoload
(defun sacha-learn-lang-practice-line ()
  (interactive)
  (sit-for 1)
  (let* ((date (format-time-string "%Y-%m-%d-%H-%M-%S"))
    (filename
      (expand-file-name
        (concat date " "
          (string-trim
            (buffer-substring (line-beginning-position)
              (line-end-position)))
          ".opus")
        sacha-learn-lang-practice-dir))
    (process (start-process "arecord"
      (get-buffer-create "*record*")
      "arecord" "-D" "hw:2,0" "-t" "wav" "-f" "cd"
      sacha-learn-lang-practice-temp)))
    (read-key "Press a key")
    (kill-process process)
    (call-process "ffmpeg" nil (get-buffer-create "*record*") nil
      "-y"
      "-i"
      sacha-learn-lang-practice-temp
      "-af" ←
      ↪ "silenceremove=start_periods=1:start_duration=0:start_threshold=-60dB,areverse,silenceremove=start_periods=1"
      filename)
    (mpv-play filename)))
```

```

;; ##autoload
(defun sacha-learn-lang-practice-replay ()
  (interactive)
  (mpv-play (sacha-latest-file sacha-learn-lang-practice-dir)))

;; ##autoload
(defun sacha-learn-lang-practice-transcribe ()
  (interactive)
  (let* ((default-directory sacha-learn-lang-practice-dir)
         (file (sacha-latest-file sacha-learn-lang-practice-dir "\\opus\\\\.m4a\\\\.webm")))
    ;; (call-process "/bin/bash" nil nil nil "/home/sacha/bin/whisperx" (expand-file-name <
    ↪ file))
    (message "%s"
      (with-temp-buffer
        (insert-file-contents (concat (file-name-sans-extension file) ".txt"))
        (string-trim (buffer-string))))))

;; ##autoload
(defun sacha-learn-lang-practice-play-current-reference (&optional beg end)
  "Play the current segment."
  (interactive)
  (when (derived-mode-p 'subed-mode)
    (let ((comment (subed-subtitle-comment)))
      (cond
        ((string-match "#\\+REFERENCE: \\(\\.+\\) +\\(\\.+?\\) +--> +\\([^\ ]+\\)" comment)
         (let ((file (expand-file-name (match-string 1 comment)))
              (start (match-string 2 comment))
              (stop (match-string 3 comment)))
           (call-process "mpv" nil nil nil
            file
            (format "--start=%.3f" (/ (subed-timestamp-to-msecs start) 1000.0))
            (format "--end=%.3f" (/ (subed-timestamp-to-msecs stop) 1000.0)))))
        ((string-match "#\\+REFERENCE: \\(\\.+\\)" comment)
         (call-process "mpv" nil nil nil (match-string 1 comment)))
        (t (let ((start
                  (if beg (save-excursion
                           (goto-char beg)
                           (subed-subtitle-msecs-start))
                  (subed-subtitle-msecs-start)))
                 (stop
                  (if beg (save-excursion
                           (goto-char end)
                           (subed-subtitle-msecs-stop))
                  (subed-subtitle-msecs-stop))))
             (call-process "mpv" nil nil nil (subed-media-file)
              (format "--start=%.3f" (/ start 1000.0))
              (format "--end=%.3f" (/ stop 1000.0))))))))))

;; ##autoload
(defun sacha-learn-lang-practice-record-loop (&optional extra)
  (interactive)
  (list (cond
        ((and (derived-mode-p 'subed-mode)
              (region-active-p))
         (subed-subtitle-list-text
          (subed-subtitle-list
           (region-beginning)
           (region-end))))
        ((region-active-p)

```

```

        (concat
         " "
         (string-trim
          (buffer-substring (region-beginning) (region-end))))))
      ((derived-mode-p 'subed-mode)
       (concat " " (subed-subtitle-text))))))
(let* ((date (format-time-string "%Y-%m-%d-%H-%M-%S"))
      (key-prompt "%d (SPC to review, RET get feedback, q to quit, any other key to retry)"
      (filename
       (expand-file-name
        (concat date
                  (if extra
                      (concat " "
                              (string-trim (replace-regexp-in-string
                                             "[\\?]+" " " "
                                             (car
                                              (split-string
                                               extra "\\n")))))
                      ""))
        ".opus")
       sachal-learn-lang-practice-dir))
      done
      char
      (count 0)
      process)
  (while (not done)
    (when (derived-mode-p 'subed-mode)
      (if (region-active-p)
          (sachal-learn-lang-practice-play-current-reference (region-beginning) (region-end))
          (sachal-learn-lang-practice-play-current-reference)))
      (setq count (1+ count))
      (setq process (start-process "ffmpeg"
                                   (get-buffer-create "*record*")
                                   "ffmpeg" "-y" "-f" "pulse" "-i" ↵
                                   ↵ "alsa_input.usb-Blue_Microphones_Yeti_Stereo_Microphone_REV8-00.analog-stereo"
                                   sachal-learn-lang-practice-temp))
      (setq char
              (read-key (format key-prompt
                                count))))
    (while char
      (pcase char
        (?\ (when (process-live-p process)
                 (sit-for 1)
                 (kill-process process))
              (sachal-learn-lang-practice-play-current-reference)
              (call-process "mpv" nil nil nil sachal-learn-lang-practice-temp)
              (setq char
                      (read-key (format "%d (SPC to review, RET get feedback, q to quit, any ↵
                                   ↵ other key to retry"
                                   count))))
        (13 (when (process-live-p process)
                  (sit-for 1)
                  (kill-process process))
              (setq done 'feedback)
              (setq char nil))
        (?w ; transcribe with Whisper
              (sachal-learn-lang-practice-transcribe))
        (?q

```

```
(when (process-live-p process) (kill-process process))
(setq done 'ignore)
(setq char nil))
(_
  (when (process-live-p process) (kill-process process))
  (setq char nil))))
(when (member done '(keep feedback))
  (call-process "ffmpeg" nil (get-buffer-create "*record*") nil
    "-y"
    "-i"
    sachal-learn-lang-practice-temp
    "-af" "loudnorm=I=-16:LRA=11:TP=-1.5"
    ;; "-af" ←
    ↪ "silenceremove=start_periods=1:start_duration=0:start_threshold=-100dB,areverse,silenceremo
      filename)
  (when (eq done 'feedback)
    (sachal-learn-lang-get-audio-feedback filename extra t))))
```

7.1.2 Start the process for transcribing the latest recording

```
;; ##autoload
(defun sacha-learn-lang-process-latest-recording (annotation)
  (interactive "MAnnotation: ")
  (let* ((file (sacha-latest-file sacha-recordings-dir))
         (audio (expand-file-name
                  (concat (file-name-base file) "-" annotation ".opus")
                  "~/sync/recordings/))))
    (make-process :name "whisperx"
                  :buffer (get-buffer-create "*whisperx*")
                  :command (list
                            "bash"
                            "-c"
                            (format
                             "ffmpeg -y -i %s %s; cd ~/sync/recordings; ~/bin/whisperx %s"
                             (shell-quote-argument file)
                             (shell-quote-argument audio)
                             (shell-quote-argument audio))))
    (message "Started %s" audio)))
```

7.1.3 Emacs Lisp and NodeJS: Getting the bolded words from a section of a Google Documentjs:emacs

- [2026-03-13 Fri]: Cleaned up links from Google
- [2026-03-12 Thu]: Simplified getting a section or finding the bolded text by using the Org Mode format instead.

During the sessions with my French tutor, I share a Google document so that we can mark the words where I need to practice my pronunciation some more or tweak the wording. Using Ctrl+B to make the word as bold is an easy way to make it jump out.

I used to copy these changes into my Org Mode notes manually, but today I thought I'd try automating some of it.

First, I need a script to download the HTML for a specified Google document. This is probably easier to do with the NodeJS library rather than with `oauth2.el` and `url-retrieve-synchronously` because of various authentication things.

```

require('dotenv').config();
const { google } = require('googleapis');

async function download(fileId) {
  const auth = new google.auth.GoogleAuth({
    scopes: ['https://www.googleapis.com/auth/drive.readonly'],
  });
  const drive = google.drive({ version: 'v3', auth });
  const htmlRes = await drive.files.export({
    fileId: fileId,
    mimeType: 'text/html'
  });
  return htmlRes.data;
}

async function main() {
  console.log(await download(process.argv.length > 2 ? process.argv[2] : process.env['DOC_ID']));
}

main();

```

Then I can wrap a little bit of Emacs Lisp around it.

```

(defun sach-google-doc-download-command
  (list "nodejs" (expand-file-name "~/bin/download-google-doc-html.cjs")))

;;;###autoload
(defun sach-google-doc-html (doc-id)
  (when (string-match "https://docs\\.google\\.com/document/d/\\(\\.+?\\)/" doc-id)
    (setq doc-id (match-string 1 doc-id)))
  (with-temp-buffer
    (apply #'call-process (car sach-google-doc-download-command)
      nil t nil (append (cdr sach-google-doc-download-command) (list doc-id)))
    (buffer-string)))

(require 'dom)
;;;###autoload
(defun sach-google-doc-clean-html (html)
  "Remove links on spaces, replace Google links."
  (let ((dom (with-temp-buffer
                (insert html)
                (libxml-parse-html-region))))
    (dom-search
     dom
     (lambda (o)
       (when (eq (dom-tag o) 'a)
         (when (and (dom-attr o 'href)
                    (string-match "https://\\(www\\.\\)?google\\.com/url\\?q=" (dom-attr o ↵
↵ 'href))))
           (let* ((parsed (url-path-and-query
                           (url-generic-parse-url (dom-attr o 'href))))
                  (params (url-parse-query-string (cdr parsed)))
                  (dom-set-attribute o 'href (car (assoc-default "q" params #'string=))))
             (let ((text (string= (string-trim (dom-text o)) ""))
                   (when (string= text "")
                     (setf (car o) 'span))))
               (when (and
                     (string-match "font-weight:700" (or (dom-attr o 'style) ""))

```

```

        (not (string-match "font-style:normal" (or (dom-attr o 'style) ""))))
        (setf (car o) 'strong))
      (when (dom-attr o 'style)
        (dom-remove-attribute o 'style))))
;; bold text is actually represented as font-weight:700 instead
(with-temp-buffer
  (svg-print dom)
  (buffer-string)))

;;;###autoload
(defun sacha-google-doc-org (doc-id)
  "Return DOC-ID in Org Mode format."
  (pandoc-convert-stdio (sacha-google-doc-clean-html (sacha-google-doc-html doc-id)) "html" ↵
    ↵ "org"))

```

I have lots of sections in that document, including past journal entries, so I want to get a specific section by name.

```

;;;###autoload
(defun sacha-org-get-subtree-by-name (org-text heading-name)
  "Return ORG-TEXT subtree for HEADING-NAME."
  (with-temp-buffer
    (insert org-text)
    (org-mode)
    (goto-char (point-min))
    (while (re-search-forward " " nil t)
      (replace-match " "))
    (goto-char (point-min))
    (let ((org-trust-scanner-tags t))
      (car (delq nil
        (org-map-entries
          (lambda ()
            (when (string= (org-entry-get (point) "ITEM") heading-name)
              (buffer-substring (point) (org-end-of-subtree))))))))))

```

Now I can get the bolded words from a section of my notes, with just a sentence for context. I use pandoc to convert it to Org Mode syntax.

```

(defun sacha-learn-lang-words-for-review-context-function 'sentence-at-point)
(defun sacha-learn-lang-tutor-notes-url nil)
;;;###autoload
(defun sacha-learn-lang-tutor-notes (section-name)
  (sacha-org-get-subtree-by-name
   (sacha-google-doc-org sacha-learn-lang-tutor-notes-url)
   section-name))

;;;###autoload
(defun sacha-learn-lang-words-for-review (section)
  "List the bolded words for review in SECTION."
  (let* ((section (sacha-learn-lang-tutor-notes section))
         results)
    (with-temp-buffer
      (insert section)
      (org-mode)
      (goto-char (point-min))
      (org-map-entries
        (lambda ()
          (org-end-of-meta-data t)

```

```
(unless (looking-at org-heading-regexp)
  (let ((end (save-excursion (org-end-of-subtree))))
    (while (re-search-forward "\\*[^* ].*?\\*" end t)
      (cl-pushnew
        (replace-regexp-in-string
          "[\n ]+" " "
          (funcall sachal-learn-lang-words-for-review-context-function)
          results
          :test 'string=))))))
(nreverse results)))
```

For example, when I run it on my [notes on artificial intelligence](#), this is the list of bolded words and the sentences that contain them.

```
(sachal-learn-lang-words-for-review "Sur l'intelligence artificielle")
```

- Je l'ai aussi utilisée pour faire des recherches.
- Je peux consacrer une petite partie de mon budget à des essais, mais je ne veux pas travailler davantage pour rentabiliser une dépense plus importante.
- Je n'ai pas le temps de concentration nécessaire pour justifier l'investissement dans mon propre matériel, et sinon, les progrès sont trop rapides pour m'engager dans une configuration spécifique.
- J'ai une conscience aiguë des limites cognitives ou physiques à cause des difficultés de santé de ma mère et de ma sur, et de mes expériences avec mes limitations à cause du fait que je suis la personne principalement en charge de ma fille.
- Je lis très vite, mais je n'ai pas assez de patience pour les longs contenus vidéo ou audio.
- Je n'aime pas les textes qui contiennent beaucoup de remplissage.
- Beaucoup de gens ont une réaction forte contre l'IA pour plusieurs raisons qui incluent le battage médiatique excessif dont elle fait l'objet, son utilisation à mauvais escient, et l'inondation de banalité qu'elle produit.
- Je réécris souvent la majorité du logiciel à l'exception d'un ou deux morceaux parce que ce code ne me convient pas.
- Je ne veux pas l'utiliser pour les correctifs que je veux soumettre à d'autres projets parce que le code ne me semble pas correct et je ne veux pas gaspiller le temps d'autres bénévoles.
- J'aime pouvoir lui donner trois dépôts git et des instructions pour générer un logiciel à partir d'un dépôt pour un autre via le troisième dépôt.
- Mais je ne veux pas le publier avant de réécrire et tout comprendre.
- Sans l'IA, je pourrais peut-être apprendre plus lentement avec l'aide d'Internet, qui a beaucoup de ressources comme [Vitrine linguistique](#).
- Je veux profiter davantage, apprendre davantage avec l'aide de vraies personnes, complétée par l'aide de l'IA.
- J'adore les sous-titres simultanés, mais je n'ai pas toujours trouvé une méthode ou un système qui me convienne.

I can then go into the WhisperX transcription JSON file and replay those parts for closer review.

I can also tweak the context function to give me less information. For example, to limit it to the containing phrase, I can do this:

```
;; ##autoload
(defun sacha-split-string-keep-delimiters (string delimiter)
  (when string
    (let (results pos)
      (with-temp-buffer
        (insert string)
        (goto-char (point-min))
        (setq pos (point-min))
        (while (re-search-forward delimiter nil t)
          (push (buffer-substring pos (match-beginning 0)) results)
          (setq pos (match-beginning 0)))
        (push (buffer-substring pos (point-max)) results)
        (nreverse results)))))

(ert-deftest sacha-split-string-keep-delimiters ()
  (should
    (equal (sacha-split-string-keep-delimiters
      "Beaucoup de gens ont une réaction forte contre l'IA pour plusieurs raisons qui ←
    ↪ *incluent* le battage médiatique excessif dont elle fait l'objet, son utilisation à ←
    ↪ mauvais escient, et *l'inondation de banalité* qu'elle produit."
      ", \\| que \\| qui \\| qu'ils? \\| qu'elles? \\| qu'on "
    )
    )))
```

```
;; ##autoload
(defun sacha-learn-lang-words-for-review-phrase-context (&optional s)
  (setq s (replace-regexp-in-string " " " " (or s (sentence-at-point))))
  (string-join
    (seq-keep
      (lambda (s)
        (when (string-match "\\*" s)
          (replace-regexp-in-string "^, " "" s)))
        (sacha-split-string-keep-delimiters s ", \\| parce que \\| que \\| qui \\| qu'ils? \\| ←
    ↪ qu'elles? \\| qu'on \\| pour ")
        " ... "))
    )

(ert-deftest sacha-learn-lang-words-for-review-phrase-context ()
  (should
    (equal (sacha-learn-lang-words-for-review-phrase-context
      "Je peux consacrer une petite partie de mon *budget* à des essais, mais je ne ←
    ↪ veux pas travailler davantage pour rentabiliser une dépense plus importante."
      "Je peux consacrer une petite partie de mon *budget* à des essais"))
    )))
```

```
(let ((sacha-learn-lang-words-for-review-context-function ←
  ↪ 'sacha-learn-lang-words-for-review-phrase-context))
  (sacha-learn-lang-words-for-review "Sur l'intelligence artificielle"))
```

- pour faire des recherches.
- Je peux consacrer une petite partie de mon budget à des essais
- et sinon
- J'ai une conscience aiguë des limites cognitives ou physiques à cause des difficultés de santé de ma mère et de ma sur
- pour les longs contenus vidéo ou audio.

- Je n'aime pas les textes qui contiennent beaucoup de remplissage.
- qui incluent le battage médiatique excessif dont elle fait l'objet ... et l'inondation de banalité
- Je réécris souvent la majorité du logiciel à l'exception d'un ou deux morceaux
- pour les correctifs ... parce que le code ne me semble pas correct et je ne veux pas gaspiller le temps d'autres bénévoles.
- pour un autre via le troisième dépôt.
- Mais je ne veux pas le publier avant de réécrire et tout comprendre.
- je pourrais peut-être apprendre plus lentement avec l'aide d'Internet
- apprendre davantage avec l'aide de vraies personnes, complétée par l'aide de l'IA.
- qui me convienne.
- que la capacité de lui donner plus de contrôle dans une boucle de feedback rapide(??)
- qu'elle me permet de saisir plus d'idées plus vite (avant de les oublier) et d'analyser les transcriptions sans avoir à réécouter tous les enregistrements.
- Beaucoup de raisons peuvent empêcher une personne de taper.
- qui sont souvent mauvais et [qui enlèvent l'expérience](#) de rencontrer d'autres personnes
- J'aime suivre les liens où je peux en apprendre davantage.
- Je veux extraire mes fonctions personnelles dans des bibliothèques de reconnaissance vocale et d'apprentissage des langues ... parce que mon attention est facile à détourner.

Now that I have a function for retrieving the HTML or Org Mode for a section, I can use that to wdiff against my current text to more easily spot wording changes.

```
;; ##autoload
(defun sacha-learn-lang-tutor-notes-wdiff-org ()
  (interactive)
  (let ((section (org-entry-get (point) "ITEM")))
    (sacha-wdiff-strings
     (replace-regexp-in-string
      " " " "
      (sacha-org-subtree-text-without-blocks))
     (replace-regexp-in-string
      org-link-bracket-re
      "\\2"
      (replace-regexp-in-string
       " " " "
       (sacha-learn-lang-tutor-notes section))))))
```

Related:

- `sacha-wdiff-strings` is in [Wdiff](#) (p. 185)
- `sacha-org-subtree-text-without-blocks` is in [Counting words without blocks](#) (p. 288)

Screenshot:



7.1.4 Emacs and French: Focus flycheck-grammalecte on the narrowed part of the buffer

After learning about French spellcheck and grammar checking from [Emacs expliqué à mes enfants](#), I added [flycheck-grammalecte](#) to my config. Nudged by [@lann@mastodon.zaclys.com](#), I finally got around to figuring out why my setup sometimes worked and sometimes didn't. When I checked [flycheck-verify-setup](#), I noticed that [grammalecte](#) kept getting disabled. A little digging around showed me that it was getting disabled because of too many errors. That was because it was trying to work on my whole file instead of just the portion that I [narrowed](#) to with [org-narrow-to-subtree](#) (ooh, just noticed an [org-toggle-narrow-to-subtree](#) command). I like having all of my French journal entries in one file because I can use [consult-line](#) (which I've bound to [M-g l](#)) to quickly look up examples of where else I've used a word. So I needed to define a checker that runs only on the narrowed part of the buffer.

[2026-04-11 Sat]: Now at [learn-lang-flycheck-grammalecte-narrowed-setup](#).

After I use [sacha-flycheck-grammalecte-setup](#), I can use [flycheck-select-checker](#) to select [sacha-grammalecte-narrowed](#) and then use [flycheck-buffer](#) to run it. Then it will underline all the number/gender agreement issues I usually have. It's nice that I can practise editing my text with this script before I run the text through an LLM (also via flycheck) for feedback on wording.

The screenshot shows an Emacs editor window with a buffer named 'journal-fr.org'. The text in the buffer is in French and discusses voice recognition, D&D game profiles, and a D&D game. Below the text, the Flycheck error buffer is visible, showing several warnings from the 'my-grammalecte-narrowed' checker. The errors include: 'Pas de majuscule sur les jours de la semaine.', 'Incohérence. Le verbe devrait être un participe passé.', 'Confusion : employez un participe passé.', 'Conjugaison erronée. Accord avec « Quelques débutants... ».', 'Il faut élider le pronom « je » et l'accoler au verbe.', 'Incohérence avec « d' » : « utilise » est une forme verbale conjuguée.', 'Typographie : points de suspension. Utilisez le caractère dédié.', and 'Accord de genre erroné : « façon » est féminin.'.

7.15 Speech synthesis

gtts-cli libsox-fmt-mp3 gtts-cli "rendez-vous" -l fr | play -t mp3 -

Update [2026-03-08 Sun]: Moved to [learn-lang-tts.el](#) and [learn-lang-phonemes.el](#)

```
(use-package learn-lang-tts :load-path "~/proj/learn-lang"
  :config
  (setq learn-lang-tts-kokoro-cli-executable "~/local/bin/kokoro-tts --model ←
    → /home/sacha/vendor/kokoro-onnx/kokoro-v1.0.onnx --voices ←
    → /home/sacha/vendor/kokoro-onnx/voices-v1.0.bin)))
```

DONE Azure Moved to <https://codeberg.org/sachac/learn-lang/src/branch/main/learn-lang-tts.el>

7.16 Add shadowing with tts to subed-record

emacs:stream

```
(setq learn-lang-subed-record-reference-dir "~/proj/french/reference/")

(with-eval-after-load 'subed-record
  (add-hook 'subed-record-finished-hook 'sacha-subed-record-normalize-current))
```

```
;; ##autoload
(defun sacha-subed-record-normalize-current (file)
  (interactive (list (subed-media-file)))
  (let ((temp-file (make-temp-file file nil (concat "." (file-name-extension file)))))
    (make-process
      :name "normalize"
      :buffer (get-buffer-create "*normalize*")
      :command (list
        (expand-file-name "~/bin/normalize")
        (expand-file-name file)
        temp-file)
      :sentinel
      (lambda (process event)
        (when (string-match "finished" event)
```

```
(rename-file temp-file file t)
(message "Normalized %s" file))))))
```

7.1.7 Make it easy to add reference links

```
(defvar-local sacha-subed-record-references nil)
;; ###autoload
(defun sacha-subed-record-load-references (file &optional skip-insert)
  "Load the references from FILE (media)."
  (interactive (list (read-file-name "Media file: ")
                     current-prefix-arg))
  (dolist (cue (subed-parse-file (concat (file-name-sans-extension file) ".vtt")))
    (push
     (list
      (learn-lang-subed-record-simplify (elt cue 3))
      file
      (elt cue 1)
      (elt cue 2))
     sacha-subed-record-references))
  (unless skip-insert
    (sacha-subed-insert-references)))

;; ###autoload
(defun sacha-subed-record-insert-reference ()
  (interactive)
  (when-let* ((rec (alist-get (learn-lang-subed-record-simplify (subed-subtitle-text))
                             sacha-subed-record-references
                             nil nil #'string=)))
    (subed-record-set-directive
     "#+REFERENCE"
     (format "%s %s --> %s"
              (elt rec 0)
              (subed-msecs-to-timestamp (elt rec 1))
              (subed-msecs-to-timestamp (elt rec 2))))))

;; ###autoload
(defun sacha-subed-insert-references ()
  (interactive)
  (subed-for-each-subtitle (point-min) (point-max) t
    (sacha-subed-record-insert-reference)))
```

7.1.8 Word timestamps

```
;; ###autoload
(defun sacha-learn-lang-word-timestamps ()
  "Add timestamps using the MFA French model."
  (interactive)
  (let ((subed-align-mfa-dictionary "french_mfa")
        (subed-align-mfa-acoustic-model "french_mfa"))
    (subed-align-mfa-set-word-data
     (subed-media-file)
     nil nil
     (lambda (&rest _)
       (subed-word-data-add-word-timestamps))))))
```

7.1.9 AI feedback

```
;; ##autoload
(defun sacha-learn-lang-get-audio-feedback (filename &optional extra-text display)
  (interactive (list (if current-prefix-arg
                        (read-file-name "File: " sacha-learn-lang-practice-dir nil t nil
                                         (lambda (filename) (string-match "\\..m4a$" filename)))
                        (sacha-latest-file sacha-learn-lang-practice-dir ←
                                           ↪ "\\(\\..m4a\\|\\..webm\\)$"))
                    (if (region-active-p)
                        (buffer-substring (region-beginning) (region-end)))
                    t)))
  (let* ((data
         (replace-regexp-in-string
          "\n" ""
          (shell-command-to-string (concat "base64 " (shell-quote-argument filename))))))
    (text
     (concat "Give me feedback in English on this recording of beginner French ←
     ↪ practice. I am a female A0/A1 speaker. Focus first on major mispronunciations, and ←
     ↪ provide English phonetic transcriptions for those words using italicized parenthetical ←
     ↪ notes.

Return your response using Org Mode syntax using only list items, not headings. Do not put ←
     ↪ it in a code block, just return Org Mode text. For example, *bold*. Score it out of 10.

Example output:

- Pronunciation (7 /10)
- travaillé /(trah vay yay)/
"
      (if extra-text
          (concat "\n\n###\n\n" extra-text
                  "")))
    (json-array-type 'vector)
    (json-object-type 'alist)
    (body (json-encode
           `(("contents"
              (("parts" .
                 (((("text" . ,text))
                    (("inline_data" .
                       (("mime_type" . "audio/mp4")
                        ("data" . ,data))))))
              ))))))
      result)
    (setq
     result
     (plz 'post ←
      ↪ "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash:generateContent"
      :headers
      `(("Content-Type" . "application/json")
        ("X-goog-api-key" . ,sacha-gemini-api-key))
      :as #'json-read
      :body body))
    (when display
      (with-current-buffer (get-buffer-create "*Feedback*")
        (org-mode)
        (goto-char (point-min))
        (let* ((text (map-nested-elt result '(candidates 0 content parts 0 text)))
```

```

      (score (when (string-match "Pronunciation (\\(.+\\))" text) (match-string 1 ←
↪ text))))
      (insert "*" " (org-link-make-string
        (concat "audio:"
          (replace-regexp-in-string
            (rx line-start
              (literal (getenv "HOME")))
            "~"
            filename))
        "me:") " "
        (or score "")
        "\n"
        text
        "\n\n"))
      (display-buffer (current-buffer))
      (with-selected-window (get-buffer-window (current-buffer))
        (goto-char (point-min))
        (recenter-top-bottom 0)))
      (map-nested-elt result '(candidates 0 content parts 0 text))))

```

```

;;###autoload
(defun sach-org-copy-clean-version (text)
  "Copy BEG to END without strike-throughs."
  (interactive (list (if (region-active-p) (buffer-substring (region-beginning) (region-end))
    (sentence-at-point))))
  (let ((result
    (mapconcat (lambda (o)
      (if (stringp o)
        o
        (car (org-element-contents o))))
      (seq-remove (lambda (o)
        (and (listp o)
          (member (org-element-type o)
            '(strike-through macro))))
        (org-element-parse-secondary-string
          text '(paragraph bold strike-through macro)))
        " ")))
    (setq result
      (replace-regexp-in-string
        " +" " "
        (replace-regexp-in-string "[\\.,]" "\\1" result)))
    (when (called-interactively-p 'any)
      (message "%s" result)
      (kill-new result))
    result))

```

7.1.10 Using flycheck to display AI grammar feedback from gptel as I learn French

This is now in [learn-lang-flycheck-gptel.el](#) . I can call `learn-lang-flycheck-gptel-setup` to check the buffer for errors using whatever model I've configured in gptel.

7.1.11 Process audio files

```

;;###autoload
(defun sach-audio-clip (source start-time end-time destination text)
  (interactive

```



```
(defun sachal-learn-lang-write-journal-entries-for-subtree ()
  (interactive)
  (org-map-entries
   (lambda ()
     (when (org-entry-get (point) "DATE")
       (let ((text (replace-regexp-in-string "{.+?}" "" ←
        ↪ (sachal-org-subtree-text-without-blocks))))
         (with-temp-file (expand-file-name (concat (org-entry-get (point) "DATE") ".txt")
          "~/proj/french/journal")
           (insert text))))))
   nil 'tree))
```

7.1.13 TODO Load en-fr dictionary

<https://github.com/thierryvolpiatto/dic-en-fr-fr-en>

```
(defvar sachal-learn-lang-en-fr-dictionary-file "~/proj/french/dic-en-fr.iso")
(defvar sachal-learn-lang-dictionary nil)
;;;###autoload
(defun sachal-learn-lang-load-dict ()
  (interactive)
  (with-temp-buffer
    (insert-file-contents sachal-learn-lang-en-fr-dictionary-file)
    (goto-char (point-min))
    (while (looking-at "^# ") (forward-line 1))
    (setq sachal-learn-lang-dictionary
      (seq-keep (lambda (s)
        (when (string-match "^\\(.+?\\) {\\(.+?\\)}\\(?: /\\(.+?\\)/\\)?\\(?: ←
        ↪ (\\(.+?\\))\\)?\\(?: SEE: .+?\\)? ::\\(?: \\(.+?\\) *\\({.+}.*\\)?\\)?$" s)
          (let ((headword (match-string 1 s))
                (head-type (match-string 2 s))
                (def (match-string 4 s))
                (see (match-string 5 s))
                (translation (match-string 6 s))
                (types (match-string 7 s)))
            (cons
             (propertize (format "%s {%- %s - %s :: %s"
                                headword
                                head-type
                                (or see translation)
                                (or def "")
                                (or types ""))
                          'gender)
             (cond
              ((null types) nil)
              ((string-match "^{m}" types) "m")
              ((string-match "^{f}" types) "f"))
             translation))))
              (split-string (buffer-substring (point) (point-max)) "\\n"))
      (seq-take sachal-learn-lang-dictionary 10)))

;;;###autoload
(defun sachal-learn-lang-consult-en-fr ()
  (interactive)
  (unless sachal-learn-lang-dictionary (sachal-learn-lang-load-dict))
  (insert
   (consult--read
    (consult--dynamic-collection
```



```

(lambda (input)
  (let (match-start
        match-any
        exact
        (search (regexp-quote input)))
    (seq-map (lambda (o)
              (setf (car o)
                    (propertize
                     (car o)
                     'face
                     (list
                      :background
                      (pcase (get-text-property 0 'gender (car o))
                        ('nil nil)
                        ("m" (modus-themes-get-color-value 'bg-blue-subtle))
                        ("f" (modus-themes-get-color-value 'bg-magenta-subtle))))))
            (cond
             ((string-match (concat "^" search " - ") (car o))
              (push o exact))
             ((string-match (concat "^" search) (car o))
              (push o match-start))
             ((string-match search (car o))
              (push o match-any))))
      sachal-learn-lang-dictionary)
    (append
     (nreverse exact)
     (nreverse match-start)
     (nreverse match-any)
     nil))))
:sort nil
:lookup #'consult--lookup-cdr)))

```

<https://github.com/emacs-mirror/wordreference>

7.1.14 Conjugation

- <http://rali.iro.umontreal.ca/rali/?q=en/node/1238>
- <https://github.com/RosaeNLG/rosaenlg>

```

;; ##autoload
(defun sachal-learn-lang-conjugate-grammalecte (input)
  "Query Grammalecte for INPUT."
  (interactive (list (sachal-learn-lang-lexique-complete-word)))
  (grammalecte-conjugate-verb input))

(defvar sachal-learn-lang-verbe-db <
  <- "~/vendor/verbe-conjugaison-academie-francaise/output/verbs.db")

;; ##autoload
(defun sachal-learn-lang-conjugate (input &optional all-forms)
  (interactive (list (sachal-learn-lang-lexique-complete-word)
                    (null current-prefix-arg)))
  (let* ((db (sqlite-open sachal-learn-lang-verbe-db))
         (lemme (elt (car (sachal-learn-lang-lexique-lookup-db-exact input)) 1))
         (value
          (consult--read
           (if all-forms

```

```

        (mapcar (lambda (row)
                  (cons (string-join
                        (list
                          (elt row 0)
                          (elt row 1)
                          (elt row 2)
                          (elt row 4)
                          (elt row 3))
                        " - ")
                        (elt row 0)))
                (sqlite-select db
                              "SELECT
conjugaison, voix, mode, temps, personne
FROM verbes v
JOIN conjugaisons c
ON v.id = c.verbe_id WHERE v.infinitif = ?
ORDER BY temps, mode, personne" (list lemme)))
        (mapcar (lambda (row)
                  (cons (string-join
                        (list
                          (elt row 0)
                          (elt row 4)
                          (elt row 3))
                        " - ")
                        (elt row 0)))
                (sqlite-select db
                              "SELECT
conjugaison, voix, mode, temps, personne
FROM verbes v
JOIN conjugaisons c
ON v.id = c.verbe_id WHERE v.infinitif = ?
AND temps in (?, ?)
AND mode=?
ORDER BY temps, mode, personne"
                              (list input "present" "passe_compose" ←
                                  ↪ "indicatif"))))
        :prompt "Verbe: "
        :lookup 'consult--lookup-cdr)))
(sqlite-close db)
(when (called-interactively-p 'any)
  (when (word-at-point)
    (delete-region (save-excursion
                     (skip-syntax-backward "w")
                     (point))
                   (save-excursion
                     (skip-syntax-forward "w")
                     (point)))))
(insert value))
value))

```

7.1.15 Looking up words via wordreference

There's also a [wordreference](#).

```

(defvar sachal-learn-lang-wordreference-cache nil)

;;;###autoload

```

```

(defun sacha-learn-lang-wordreference-completing-read ()
  (interactive)
  (consult--read
   (consult--dynamic-collection
    (lambda (input)
      (message "input: %s" input)
      (with-current-buffer
       (url-retrieve-synchronously
        (concat "https://www.wordreference.com/autocomplete?dict=enfr&query="
                 (url-hexify-string input)))
       (set-buffer-multibyte t)
       (goto-char (point-min))
       (re-search-forward "^$" nil t)
       (prog1
        (mapcar (lambda (row)
                   (let ((fields (split-string row "\t")))
                     (propertize
                      (format "%s (%s)"
                               (car fields)
                               (cadr fields))
                      'consult--candidate
                      fields)))
                  (split-string
                   (string-trim (buffer-substring (point) (point-max))) "\n"))
        (kill-buffer (current-buffer))))))
    :sort nil
    :initial (symbol-name (symbol-at-point))
    :history 'sacha-learn-lang-wordreference-lookup-history
    :prompt "Word: "
    :category 'word))

(defvar-keymap sacha-learn-lang-wordreference-keymap
  "v" #'sacha-spookfox-scroll-down
  "V" #'sacha-spookfox-scroll-up
  "c" #'sacha-learn-lang-conjugate-last-word
  "l" #'sacha-learn-lang-wordreference-lookup)

(defun sacha-learn-lang-wordreference-lookup-history nil)
;; ###autoload
(defun sacha-learn-lang-wordreference-lookup (word)
  (interactive (list (sacha-learn-lang-wordreference-completing-read)))
  (let (language)
    (if (string-match " (\\(en\\|fr\\))" word)
        (setq language (match-string 1 word)
              word (replace-match "" nil t word 0)))
    (setq word (replace-regexp-in-string "^#" "" word))
    (browse-url
     (format "https://www.wordreference.com/%s/%s"
             (if (string= language "fr")
                 "fren"
                 "enfr")
             (url-hexify-string word))))
  (set-transient-map sacha-learn-lang-wordreference-keymap t))

;; ###autoload
(defun sacha-learn-lang-conjugate-last-word ()
  (interactive)
  (grammalecte-conjugate-verb
   (replace-regexp-in-string " (\\.+)" ""

```

```

(car sacha-learn-lang-wordreference-lookup-history))))

;; ###autoload
(defun sacha-learn-lang-wordreference-conjugate (word)
  (interactive (list (sacha-learn-lang-wordreference-completing-read)))
  (browse-url (concat "https://www.wordreference.com/conj/frverbs.aspx?v="
    (url-hexify-string (if (listp word) (car word) word))))))

;; ###autoload
(defun sacha-learn-lang-reverso (s)
  (interactive (list (if (region-active-p)
    (buffer-substring (region-beginning) (region-end))
    (word-at-point))))
  (browse-url (concat "https://www.reverso.net/text-translation#sl=fra&tl=eng&text="
    (url-hexify-string s)))))

```

7.1.16 Lexique

<http://www.lexique.org/databases/Lexique383/>

```
(defvar sacha-learn-lang-lexique-db "~/proj/french/lexique.db" "SQLite3 DB")
```

How to load the TSV into SQLite:

```

import pandas as pd
import sqlite3
df = pd.read_csv('lexique383.tsv', sep='\t')
conn = sqlite3.connect('lexique.db')
df.to_sql('lexique', conn, if_exists='replace', index=False)
conn.close()

```

Adding an unaccented version for easier lookup:

```

ALTER TABLE lexique ADD COLUMN ortho_flat TEXT;

-- Update it (you can use your Python logic to strip accents)
-- Or use SQLite's built-in REPLACE for common French chars:
UPDATE lexique SET ortho_flat =
  REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(LOWER(ortho), 'é', 'e'), 'è', 'e'), 'ê', 'e'), 'à', 'a'), 'ç', 'c'), 'ô', 'o'), 'î', 'i');

-- Index the flat column
CREATE INDEX idx_ortho_flat ON lexique(ortho_flat);

```

Emacs Lisp functions for finding things:

```

(defvar sacha-learn-lang-csv-path "/home/sacha/proj/french/french_vocabulary_list.csv")
(defvar sacha-learn-lang-known-lemmas nil)
;; ###autoload
(defun sacha-learn-lang-lexique-lookup-db-exact (input)
  "Query the Lexique SQLite database for INPUT."
  (let ((db (sqlite-open sacha-learn-lang-lexique-db)))
    (prog1 (sqlite-select db
      "SELECT ortho, lemme, genre, nombre, phon, syll, infover FROM lexique
      WHERE ortho=? ORDER BY freqfilms2 DESC LIMIT 1"
      (list input))
      (sqlite-close db))))

```

```

        (sqlite-close db))))

;; ###autoload
(defun sacha-learn-lang-lexique-lookup-db-flat (input)
  "Query the Lexique SQLite database for INPUT."
  (let ((db (sqlite-open sacha-learn-lang-lexique-db)))
    (prog1 (sqlite-select db
      "SELECT ortho, lemme, genre, nombre, phon, syll, infover FROM lexique
      WHERE ortho_flat LIKE ? OR ortho LIKE ? ORDER BY freqfilms2 DESC LIMIT 50"
      (list (concat (downcase input) "%")
            (concat (downcase input) "%")))
      (sqlite-close db))))

;; ###autoload
(defun sacha-learn-lang-lexique-lookup-db-lemma (input)
  "Query the Lexique SQLite database for INPUT."
  (let ((db (sqlite-open sacha-learn-lang-lexique-db)))
    (prog1 (sqlite-select db
      "SELECT ortho, lemme, genre, nombre, infover, phon, syll, ↵
      ↵ ortho_flat FROM lexique
      WHERE lemme LIKE ? ORDER BY freqfilms2 DESC LIMIT 50"
      (list (concat input "%")))
      (sqlite-close db))))

;; ###autoload
(defun sacha-learn-lang-lexique-complete-word ()
  (interactive)
  (let* ((selection
    (consult--read
      (consult--dynamic-collection
        (lambda (input)
          (modus-themes-with-colors
            (mapcar (lambda (row)
              (let ((word (nth 0 row))
                    (gender (nth 2 row))
                    (number (nth 3 row))
                    (ipa (sacha-learn-lang-lexique-to-ipa (nth 5 row)))
                    (infover (nth 6 row)))
                ;; Format the string for the completion buffer
                (cons
                  (propertize
                    (format "%-20s [%s] (%s)" word ipa
                      (string-join
                        (delq nil (list gender number infover))
                        ", "))
                    'consult--candidate word
                    'face
                    `(:background
                      ,(pcase gender
                        ("m" bg-blue-subtle)
                        ("f" bg-magenta-subtle))))
                  word))))
            (sacha-learn-lang-lexique-lookup-db-flat input))))))
    :prompt "French word: "
    :initial (word-at-point)
    :sort nil
    :lookup #'consult--lookup-cdr
    :category 'french-word)))

```

```

(when selection
  (when (called-interactively-p 'any)
    (when (word-at-point)
      (delete-region (save-excursion
                       (skip-syntax-backward "w")
                       (point))
                     (save-excursion
                       (skip-syntax-forward "w")
                       (point))))
      (insert selection))
    selection)))

```

7.1.17 **STARTED** Write a completion-at-point function for French

```

;; ###autoload
(defun sacha-learn-lang-lexique-completion-at-point ()
  (let ((bounds (bounds-of-thing-at-point 'word)))
    (when bounds
      (list (car bounds)
            (cdr bounds)
            (mapcar (lambda (row)
                      (let ((word (nth 0 row))
                            (gender (nth 2 row))
                            (number (nth 3 row))
                            (ipa (sacha-learn-lang-lexique-to-ipa (nth 5 row)))
                            (infover (nth 6 row)))
                        word))
                    (sacha-learn-lang-lexique-lookup-db-flat
                     (buffer-substring-no-properties
                      (car bounds)
                      (cdr bounds))))
              :exclusive 'no))))
  (with-eval-after-load 'org
    (add-hook 'org-mode-hook
      (lambda ()
        (when (and (buffer-file-name) (string-match "journal-fr\\|french" ↵
        ↵ (buffer-file-name)))
          (add-hook 'completion-at-point-functions ↵
        ↵ 'sacha-learn-lang-lexique-completion-at-point))))))

```

7.1.18 Highlight and count new words in journal entries

```

;; ###autoload
(defun sacha-learn-lang-load-known-lemmas ()
  "Parse the CSV and return a hash table of lemma -> info."
  (let* ((known (make-hash-table :test 'equal))
         (data (cdr (pcsv-parse-file sacha-learn-lang-csv-path))))
    (mapc
     (lambda (o)
       (unless (gethash (elt o 2) known)
         (puthash (elt o 2) o known))
       (unless (gethash (elt o 0) known)
         (puthash (elt o 0) o known)))
     data)
    (setq sacha-learn-lang-known-lemmas known)))

```



```

'modus-themes-fg-cyan-faint
'keymap
sacha-learn-lang-overlay-map
))
(csv-date (car (gethash lemma known-lemmas)))
(is-new (and (or (null csv-date)
                 (and subtree-date (not (string> subtree-date
                                                csv-date)))))
        (not (gethash lemma seen-so-far))
        (not (member lemma sacha-learn-lang-ignore))))
(when is-new
  (puthash lemma (list subtree-date word lemma) seen-so-far)
  (puthash lemma (list subtree-date word lemma) ←
    ↪ sacha-learn-lang-known-lemmas)
  (puthash word (list subtree-date word lemma) ←
    ↪ sacha-learn-lang-known-lemmas)
  (setq count-new (1+ count-new))
  (let ((ov (make-overlay (match-beginning 0) (match-end 0))))
    (face (cond
           ((string-equal gender "m") 'modus-themes-subtle-blue)
           ((string-equal gender "f") 'modus-themes-subtle-magenta)
           (t 'modus-themes-subtle-green))))
    (overlay-put ov 'sacha-learn-lang-highlight t)
    (overlay-put ov 'word word)
    (overlay-put ov 'evaporate t)
    (when sacha-learn-lang-show-pronunciation
      (overlay-put ov 'after-string (sacha-learn-lang-lexique-to-ipa ←
    ↪ syll))))
    (overlay-put ov 'face face))))))
(org-back-to-heading)
(let ((ov (make-overlay (line-beginning-position) (line-end-position))))
  (overlay-put ov 'after-string (format " + %d = %d" count-new count-words))
  (overlay-put ov 'sacha-learn-lang-highlight t))
(when (called-interactively-p 'any)
  (message "%d total words, %d new lemmas" count-words count-new))
(cons count-new count-words)))
(let ((data (org-map-entries #'sacha-learn-lang-highlight-new-words-in-subtree ←
    ↪ "DATE={.}" 'tree)))
  (org-back-to-heading)
  (let ((ov (make-overlay (line-beginning-position) (line-end-position))))
    (overlay-put ov 'evaporate t)
    (overlay-put ov 'after-string
      (format " + %d = %d"
        (apply '+ (mapcar 'car data))
        (apply '+ (mapcar 'cdr data)))))
    (overlay-put ov 'sacha-learn-lang-highlight t))))))
;; ##autoload
(defun sacha-learn-lang-remove-new-word-highlights ()
  "Remove all word highlights created by `sacha-learn-lang-highlight-new-words-in-subtree'."
  (interactive)
  (let ((beg (save-excursion (org-back-to-heading t) (point)))
        (end (save-excursion (org-end-of-subtree t) (point))))
    (remove-overlays beg end 'sacha-learn-lang-highlight t)))

```

7.1.19 Doublecheck with Google Translate

Now in [learn-lang-translate-dwim](#).

7.1.20 Correct encoding errors

```
;; ##autoload
(defun sacha-learn-lang-repair-french-encoding-full ()
  "Repair double and single encoded UTF-8 sequences, including uppercase."
  (interactive)
  (save-excursion
    (let ((case-fold-search t)
          (pairs '((("fÃÂ" . "é") ("f%ÃÂ" . "É")
                    ("fÃÂ" . "à") ("f€ÃÂ" . "À")
                    ("Ã" . "ê")
                    ("Ã" . "ê")
                    ("Ã" . "ç")
                    ("Ã" . "ç")
                    ("Ã" . "ç")
                    ("Ã" . "ù")
                    ("fÃÂ" . "è") ("f^ÃÂ" . "È")
                    ("fÃÂ" . "ç") ("f+ÃÂ" . "Ç")
                    ("Ã" . "é") ("&Ã" . "É")
                    ("Ã" . "à") ("€Ã" . "À")
                    ("ã" . "'')))))
      (dolist (pair pairs)
        (goto-char (point-min))
        ;; 'nil' for literal search, 't' for case-sensitivity
        (while (search-forward (car pair) nil t)
          (replace-match (cdr pair) t))))))
```

7.2 Spellcheck

7.3 gif-screencast

Animated GIFs make it easy to demonstrate things, and [gif-screencast](#) makes it easy to record a screencast with each command getting its own GIF frame. In my config, I have `s-s` (super-s, or the Windows key + s) bound to `gif-screencast-start-or-stop`. I like adding animated GIFs to my blog posts and videos.

Not all interfaces can display animated GIFs. For example, some people read my blog in [elfeed](#) or in a console Emacs and can see only the first frame of a GIF. It makes sense to select the a frame to use as the fallback for the animated GIF, much like videos can have thumbnails.

When I stop gif-screencast, I want to pick a frame and use it as the first frame. Let's see if this works...

```
;; ##autoload
(defun sacha-gif-screencast-start-or-stop-and-choose-thumbnail ()
  "Start a screencast or pause recording."
  (interactive)
  (if gif-screencast-mode
      (progn
        (gif-screencast-toggle-pause)
        (dired gif-screencast-screenshot-directory)
        (revert-buffer)
        (dired gif-screencast-screenshot-directory)
        (image-dired gif-screencast-screenshot-directory))
      (gif-screencast)))
```

Actually, what I want to do is a more general case: be able to delete frames, copy frames to the beginning, eventually move frames around, etc., and then generate the GIF based on the contents

of that directory.

```
;;;###autoload
(defun sach-gif-screencast-copy-image-to-first-frame (file)
  (interactive (list (dired-get-filename)))
  ;; Determine the timestamp of the first file in this directory
  (copy-file
   file
   (expand-file-name
    (format-time-string
     "screen-%F-%T-%3N.png"
     (time-subtract
      (sach-gif-screencast-timestamp-from-filename
       (car (directory-files gif-screencast-screenshot-directory nil
                              ↪ ".png"))
       (seconds-to-time 0.001)))
      gif-screencast-screenshot-directory)))

;;;###autoload
(defun sach-gif-screencast-timestamp-from-filename (file)
  (setq file (replace-regexp-in-string "^screen-" "" (file-name-base file)))
  (time-add (date-to-time (format "%s %s" (substring file 0 10) (substring file 11 19)))
            (float-time (/ (string-to-number (substring ↪
            ↪ file 20 23)) 1000.0))))
(cl-assert
 (string= (format-time-string "test-%F-%T-%3N" (sach-gif-screencast-timestamp-from-filename ↪
            ↪ "screen-2024-09-20-13:18:08-024.png"))
          "test-2024-09-20-13:18:08-024"))

;;;###autoload
(defun sach-gif-screencast-update-frames-from-directory ()
  (interactive)
  (let* ((files (directory-files gif-screencast-screenshot-directory nil ".png"))
         (start-time (sach-gif-screencast-timestamp-from-filename ↪
            ↪ (car files)))
         (setq gif-screencast--frames
              (mapcar (lambda (o)
                        ↪
            ↪ (make-gif-screencast-frame
                        ↪ :timestamp ↪
            ↪ (sach-gif-screencast-timestamp-from-filename o)
                        ↪ :filename o))
                     files))
         (gif-screencast-mode 0)
         (gif-screencast--finish)))
```

Someday it could be pretty cool to have an SVG where I can see the frames on a timeline and then drag them around. In the meantime, let's see how this works out.

```
(use-package gif-screencast
  :bind
  ("s-S" . sach-gif-screencast-start-or-stop-and-choose-thumbnail)
  :config
  (setq gif-screencast-output-directory sach-recordings-dir))

(use-package giffy
  :vc (:url "https://github.com/larsmagne/giffy")
  :defer t)
```

7.4 Sentences end with a single space

In my world, sentences end with a single space. This makes sentence navigation commands work for me.

```
(setq sentence-end-double-space nil)
```

7.5 Writeroom

```
(use-package writeroom-mode
  :defer t
  :commands writeroom-mode
  :config
  (setq writeroom-global-effects (remove 'writeroom-set-fullscreen
    ↩ ↩ ↩
    ↩ writeroom-global-effects)))
```

7.6 Try redacting

emacs:config

```
;;;###autoload
(defun sach-redact (s)
  "Replace S with x characters."
  (make-string (length s) ?x))

;;;###autoload
(defun sach-redact-region (beg end &optional func)
  "Redact from BEG to END."
  (interactive "r")
  (let ((overlay (make-overlay beg end)))
    (overlay-put overlay 'redact t)
    (overlay-put overlay 'evaporate t)
    (overlay-put overlay 'display
      (cond
        ((functionp func)
         (funcall func))
        ((stringp func)
         func)
        (t (make-string (- ↩ ↩
          ↩ end beg) ?x))))))

;;;###autoload
(defun sach-redact-regexp-replacement (regexp replacement &optional beg end)
  "Redact buffer content matching regexp."
  (interactive (list (read-regexp "Redact regexp: " 'regexp-history-last)
    ↩
    ↩ (read-string "Replacement (ex: \\1 \\,(sach-redact \\2)): ")
    (setq beg (or beg (point-min)))
    (setq end (or end (point-max)))
    (when (stringp replacement)
      (setq replacement (query-replace-compile-replacement replacement t)))
    (save-excursion
      (goto-char beg)
      (while (re-search-forward regexp end t)
        (sach-redact-region
```

```

                (match-beginning 0) (match-end 0)
                (with-temp-buffer
                  (insert (match-string 0))
                  (goto-char (point-min))

                  )
                (replace-regexp-in-string regexp replacement (match-string 0))))))

;; ###autoload
(defun sacha-redact-regexp (regexp &optional beg end func)
  "Redact buffer content matching regexp."
  (interactive (list (string-trim (read-regexp "Redact regexp: " 'regexp-history-last))))
  (save-excursion
    (goto-char (or beg (point-min)))
    (while (re-search-forward regexp (or end (point-max)) t)
      (sacha-redact-region (match-beginning 0) (match-end 0) func))))

;; ###autoload
(defun sacha-unredact ()
  (interactive)
  (mapc 'delete-overlay
    (seq-filter (lambda (overlay) (overlay-get overlay 'redact))
      (overlays-in (point-min) (point-max)))))

;; ###autoload
(defun sacha-redact-email-string (s)
  (replace-regexp-in-string
    "\\([-+~a-zA-Z0-9][-+_.~: a-zA-Z0-9]*\\)@\\([-a-zA-Z0-9]+[-.a-zA-Z0-9]*\\)"
    (lambda (sub)
      (concat
        (make-string (length (match-string 1 sub)) ?x)
        "@"
        (make-string (length (match-string 2 sub)) ?x)))
    s))

;; ###autoload
(defun sacha-redact-emails (&rest _)
  (interactive)
  (sacha-redact-regexp
    "\\([-+~a-zA-Z0-9][-+_.~: a-zA-Z0-9]*\\)@\\([-a-zA-Z0-9]+[-.a-zA-Z0-9]*\\)"
    nil nil
    (lambda () (sacha-redact-email-string (match-string 0)))))

;; ###autoload
(defun sacha-redact-emacsconf-org ()
  (interactive)
  (sacha-redact-regexp-replacement
    "\\(^:EMAIL:[ \\t]+\\)\\((.+\\)"
    "\\1 \\,(sacha-redact \\2)"
  ))

;; ###autoload
(defun sacha-redact-tabulated-list-in-rectangle (regexp beg end)
  ;; tabulated columns use substrings with display properties
  ;; so we should skip any characters that have text-property-any 'display
  (interactive (list (read-regexp "Redact regexp: " 'regexp-history-last)
    (min (point) (mark))

```

```

↪ (point) (mark)))                                     (max ↪
  (apply-on-rectangle
    (lambda (start-col end-col)
      (let ((start-pos (and (move-to-column start-col) (point)))
            (end-pos (and (move-to-column end-col) (point)))
            display-prop)
        (save-restriction
          (narrow-to-region start-pos end-pos)
          (goto-char start-pos)
          (setq display-prop (text-property-search-forward 'display))
          (if display-prop
              (while display-prop
                (sacha-redact-regexp regexp ↪
                ↪ start-pos (prop-match-beginning display-prop))      (setq start-pos (prop-match-end ↪
                ↪ display-prop))                                         (setq display-prop ↪
                ↪ (text-property-search-forward 'display)))             (sacha-redact-regexp regexp start-pos end-pos))))))
    beg end))

;; ##autoload
(defun sacha-redact-regexp-in-rectangle (regexp beg end)
  (interactive (list (read-regexp "Redact regexp: " 'regexp-history-last)
                    ↪
                    ↪ (point) (mark))                                     (min ↪
                    ↪ (point) (mark)))                                   (max ↪
                    (apply-on-rectangle (lambda (start-col end-col)
                      ↪
                      ↪ (sacha-redact-regexp regexp                               ↪
                      ↪
                      ↪ (move-to-column start-col) (point))             (and ↪
                      ↪
                      ↪ (move-to-column end-col) (point))))             (and ↪
                    ↪ end))                                             beg ↪

```

```

(with-eval-after-load 'notmuch
  (advice-add
    #'notmuch-show
    :after #'sacha-redact-emails))

```

7.7 **DONE** Audio braindump workflow tweaks: Adding Org Mode hyperlinks to recordings based on keywords audio:writing

[2023-12-24 Sun] Added a quick video!

<https://sachachua.com/blog/2023/12/audio-braindump-workflow-tweaks-adding-org-mode-hyperlinks-to-recordings-based-on-keywords/2023-12-22-audio-workflow.webm>

nil

Audio recording is handy for capturing thoughts as I wait, walk around, or do chores. But my wireless earbuds don't have a good mic, I rarely got back to reviewing the wall of text, and I don't trust speech recognition to catch all my words.

Here's a new brain-dumping workflow that I've been experimenting with, though. I use a lapel mic to record in my phone. Google Recorder gives me an audio file as well as a rough transcript right away.

nil

I copy those with Synthing.

If I use keywords like "start" or "stop" along with things like "topic", "reminder", or "summary", then I can put those on separate lines automatically ([sacha-audio-braindump-prepare-alignment-breaks](#)).

```
...
News. Miscellaneous little tasks that he doing. I do want to
finish that blog post about the playlist Just so that it's out.
Something else that people can, you know, refer to or that I can refer
to. Uh, And at some point I want to think about, This second brain
stuff.
So, right now, What's my current state? Uh,
START CHAPTER second brain STOP CHAPTER
Right now, I dumped everything into originally. In my inbox, if I come
across an interesting website. As usually in my phone. So then I share
it. As. Something links at those or four none. Uh, into my inbox.
...
```

I use [subed-align](#) to get the timestamps, and add the headings.

```
00:20:18.680 --> 00:20:24.679
So, right now, What's my current state? Uh,

NOTE CHAPTER: second brain

00:20:24.680 --> 00:20:30.719
START CHAPTER second brain STOP CHAPTER
```

I can then create an Org Mode TODO item with a quick hyperlinked summary as well as my transcript.

I can jump to the audio if there are misrecognized words.

nil

I can use [subed-waveform](#) to tweak the start and end times. ([subed-waveform-show-current](#), then left-clicking to set the start or right-clicking to set the end, or using keybindings to adjust the start/stop).

Someday I'll write code to send sections to a better speech recognition engine or to AI. In the meantime, this is pretty good.

Here's how the code works:

7.7.1 Recognizing keyword phrases

There are several things I want to do while dictating.

- I want to mark different topics so that it's easy to find the section where I was talking about something.

* **TODO** Review braindump from
2023-12-11T20.05 :braindump:
:PROPERTIES:...

[VTT](#) - [Text](#) - [Audio](#)

- [SUMMARY: Workflow and kenban for ideas \(VTT\)](#)
- [CHAPTER: second brain \(VTT\)](#)
- [SUMMARY: emacs called harvesting \(VTT\)](#)

* Transcript

There's brain on being the raw text
and then there is doing A little bit
of revision so that it's ready to
post. And that's actually a tricky one
for me, too. Because,
I attempted to use that time instead
to explore the next interesting idea.
I accumulate all these little bits.
which means? Yeah, probably. Another

U: ** - **Inbox.org**

[<up>](#) [previous-](#)

Figure 9: Summary with headings and links

- I might want to set tags or priorities, or even schedule something (today, tomorrow, next week, next month).
- I can also use commands to trigger different things, like sending the section to a better speech recognition engine.

By analyzing the text, I might be able to make my own command system.

So far, for starting keywords, I can use "start", "begin", or "open". I pair that with one of these part keywords:

- "section", "chapter", "topic", "summary": I use these pretty interchangeably at the moment. I want them to make a new Org heading.
- "next steps": could be handy for being able to quickly see what to do next
- "reminder":
- "interruption": don't know what I'll use this for yet, but it might be useful to note this.
- "tag", "keyword": maybe use this to add tags to the current section?

Then the code can extract the text until the matching "stop/close/end <part>", assuming it happens within 50 words or so. ([sacha-audio-braindump-close-keyword-distance-words](#))

Sometimes keywords get misrecognized. "Begin summary" sometimes becomes "again summary" or "the game summary". I could try "open" and "close". Commercial dictation programs like Dragon NaturallySpeaking use "open" and "close" for punctuation, so that would probably work fine. "Start" works well, but "end" doesn't because it can be confused with "and".

Sometimes an extra word sneaks in, either because I say it or because the speech recognition tries too hard to guess. "Begin reminder" ends up as "Begin a reminder." I changed from using regular expressions that searched for just start-keyword + part-keyword to one that looked for the start of the keyword phrase and then looked for the next keyword within the next X words. ([sacha-audio-braindump-scan-for-part-keyword](#))

```
(defvar sacha-audio-braindump-open-keywords '("start" "begin" "open"))
(defvar sacha-audio-braindump-close-keywords '("stop" "end" "close"))
(defvar sacha-audio-braindump-part-keywords '("summary" "chapter" "topic"
↵
↵                                     "section"
↵
↵                                     "action" "idea" "journal" "reminder"
↵
↵                                     "command" "interruption" "note"
↵
↵                                     "next step" "next steps" "tags" "tag" "keywords"
↵ "keyword"))

(defvar sacha-audio-braindump-part-keyword-distance-words 2 "Number of words to scan for
↵ part keyword.")
(defvar sacha-audio-braindump-close-keyword-distance-words 50 "number of words to scan for
↵ stop keyword.
Put the keywords on the same line if found.")
;; ##autoload
(defun sacha-audio-braindump-scan-for-part-keyword (before-part &optional part-keywords
↵ within-distance before-distance)
    "Look for BEFORE-PART followed by PART-KEYWORDS.
There might be WITHIN-DISTANCE words between BEFORE-PART and PART-KEYWORDS,
and the pair might be within BEFORE-DISTANCE from point.
Distances are in words.
Return (start end before-part part) if found, nil otherwise."
```



```

    (setq before-part (pcase before-part
      ↪
      ↪
      ↪
      ↪ ('nil (append sach-audio-braindump-open-keywords sach-audio-braindump-close-keywords))
      ↪
      ↪
      ↪ before-part)))
    (if (stringp before-part) (setq before-part (list before-part)))
    (setq part-keywords (or part-keywords sach-audio-braindump-part-keywords))
    (when (stringp part-keywords) (setq part-keywords (list part-keywords)))
    (setq within-distance (or within-distance ↪
      ↪ sach-audio-braindump-part-keyword-distance-words))
    (setq before-distance (if (eq before-distance t)
      ↪
      ↪ (point-max)
      ↪
      ↪ (or before-distance sach-audio-braindump-close-keyword-distance-words)))
    (let (result
          start end
          (before-point (save-excursion (forward-word before-distance) ↪
      ↪ (point)))
          before-word
          part-word)
      (save-excursion
        (when (looking-at (regexp-opt before-part))
          (setq before-word (match-string 0) start (match-beginning 0))
          (when (re-search-forward (regexp-opt part-keywords) ↪
      ↪ (save-excursion (forward-word within-distance) (point)) t)
            (setq result (list start (match-end 0) before-word ↪
      ↪ (match-string 0))))))
        (while (and (not result)
          ↪ (re-search-forward ↪
      ↪ (regexp-opt before-part) before-point t))
          (setq before-word (match-string 0) start (match-beginning 0))
          (when (re-search-forward (regexp-opt part-keywords) ↪
      ↪ (save-excursion (forward-word within-distance) (point)) t)
            (setq result (list start (match-end 0) before-word ↪
      ↪ (match-string 0))))))
        (when result (goto-char (elt result 1)))
        result)))

(ert-deftest sach-audio-braindump-scan-for-part-keyword ()
  (with-temp-buffer
    (insert "some text start a reminder hello world stop there and do something ↪
  ↪ stop reminder more text")
    (goto-char (point-min))
    (let ((result (sach-audio-braindump-scan-for-part-keyword 'start nil)))
      (expect (elt result 2) :to-equal "start")
      (expect (elt result 3) :to-equal "reminder"))
    (let ((result (sach-audio-braindump-scan-for-part-keyword 'stop "reminder")))
      (expect (elt result 2) :to-equal "stop")
      (expect (elt result 3) :to-equal "reminder"))))

```

7.7.2 Splitting the lines based on keywords and oopses

Now I can use that to scan through the text. I want to put commands on their own lines so that `subed-align` will get the timestamp for that segment and so that the commands are easier to parse.

I also want to detect "oops" and split things up so that the start of that line matches my correction after the "oops". I use `sacha-subed-split-oops` for that, which I should write about in another post. By putting the oops fragment on its own line, I can use `subed-align` to get a timestamp for just that segment. Then I can either use `flush-lines` to get rid of anything with "oops" in it. I can even remove the subtitle and use `subed-record-compile-media` to compile audio/video without that segment, if I want to use the audio without rerecording it.

And the way I can help is by jotting words down in a mind map,
typing her sentences. Oops
typing, her sentences And generating, follow-up questions.

I also all-caps the keyword phrases so that they're easier to see when skimming the text file.

```
;;;###autoload
(defun sacha-audio-braindump-prepare-alignment-breaks ()
  "Split lines in preparation for forced alignment with aeneas.

Split \"oops\" so that it's at the end of the line and the
previous line starts with roughly the same words as the next
line, for easier removal.

Add a linebreak before \"begin/start\" followed by
`sacha-audio-braindump-part-keywords'.

Add a linebreak after \"stop\" followed by
`sacha-audio-braindump-part-keywords'.

Look for begin keyword ... stop keyword with at most
`sacha-audio-braindump-part-keyword-distance-words' between them and put them on one
line. If begin or stop has been misrecognized, try the best guess."
  (interactive)
  (let ((case-fold-search t) result close-result)
    (sacha-split-oops)
    ;; break "begin/start keyword"
    (goto-char (point-min))
    (while (setq result (sacha-audio-braindump-scan-for-part-keyword 'start nil <-
  => nil t))
      (goto-char (car result))
      (delete-region (car result) (elt result 1))
      (insert "\n" (upcase (concat (elt result 2) " " (elt result 3))) "\n"))
    ;; break stop
    (goto-char (point-min))
    (while (setq result (sacha-audio-braindump-scan-for-part-keyword 'stop nil <-
  => nil t))
      (goto-char (car result))
      (delete-region (car result) (elt result 1))
      (insert (upcase (concat (elt result 2) " " (elt result 3))) "\n"))
    ;; try to get start and end sections on one line
    (goto-char (point-min))
    (while (setq result (sacha-audio-braindump-scan-for-part-keyword 'start nil <-
  => nil t))
      (goto-char (elt result 1))
      (setq stop-result (sacha-audio-braindump-scan-for-part-keyword 'stop <-
  => (elt result 3)))
```

```

        (if stop-result
            (progn
                (goto-char (car stop-result))
                (while (re-search-backward " *\n+ *" (car ←
↪ result) t)
                    (replace-match " ")))
            ;; no stop keyword; is the keyword around? maybe it was just ←
↪ unrecognized
            (if (re-search-forward (elt result 3)
↪
                (save-excursion
↪
                    (forward-word ←
↪ sacha-audio-braindump-close-keyword-distance-words)
↪
                    (point))
↪
                t)
                (save-excursion
                    (goto-char (match-beginning 0))
                    (save-excursion
                        (insert " STOP "))
                    (while (re-search-backward " *\n+ *" ←
↪ (car result) t)
                        (replace-match " ")))
                    (when (looking-at "\n+ *"
                        (replace-match " "))))))
            ;; Check for stops without starts
            (goto-char (point-min))
            (while (setq result (sacha-audio-braindump-scan-for-part-keyword 'stop nil ←
↪ nil t))
                (goto-char (car result))
                (save-excursion
                    (unless (re-search-backward (elt result 3) ←
↪ (line-beginning-position) t)
                        (when (re-search-backward
                            (elt result 3)
                            (save-excursion ←
↪ (backward-word sacha-audio-braindump-close-keyword-distance-words)
↪
                                (point))
                                t)
                            (replace-match (concat "\nSTART " (elt ←
↪ result 3))))))
                    (goto-char (cadr result)))
                ;; remove empty lines
                (goto-char (point-min))
                (when (looking-at "\n+") (replace-match ""))
                (while (re-search-forward "\n\n+" nil t)
                    (replace-match "\n"))
                (goto-char (point-min))
                (while (re-search-forward " *\n+ *" nil t)
                    (replace-match "\n"))))
(ert-deftest sacha-audio-braindump-prepare-alignment-breaks ()
  (with-temp-buffer
    (insert "some text start a reminder hello world stop there and do something ←
↪ stop reminder more text")
    (goto-char (point-min))

```

```

(sacha-audio-braindump-prepare-alignment-breaks)
(expect (buffer-string) :to-equal
      "some text
START REMINDER hello world stop there and do something STOP REMINDER
more text"))))

```

7.7.3 Preparing the VTT subtitles

`subed-align` gives me a VTT subtitle file with timestamps and text. I add NOTE comments with the keywords and make `subed:` links to the timestamps using the `ol-subed.el` that I just added.

```

;; ##autoload
(defun sacha-audio-braindump-get-subtitle-note-based-on-keywords (sub-text)
  (let ((case-fold-search t))
    (when (string-match (concat "^"
      ↪
      ↪ (regexp-opt sacha-audio-braindump-open-keywords) ↪
      ↪
      ↪ " \\\(" (regexp-opt ↪
      ↪ sacha-audio-braindump-part-keywords) "\\) \\\(\\.+?\\)\\(" "
      ↪
      ↪ (regexp-opt sacha-audio-braindump-close-keywords) " " ↪
      ↪
      ↪ (regexp-opt sacha-audio-braindump-part-keywords) ↪
      ↪ "\\)?\$")
      ↪
      ↪ sub-text)
      (concat (match-string 1 sub-text) ": " (match-string 2 sub-text))))))
(ert-deftest sacha-audio-braindump-get-subtitle-note-based-on-keywords ()
  (expect (sacha-audio-braindump-get-subtitle-note-based-on-keywords "BEGIN NEXT STEPS ↪
  ↪ . Think about how dictation helps me practice slower speed. CLOSE NEXT STEPS")
    :to-equal "NEXT STEPS: . Think about how dictation ↪
  ↪ helps me practice slower speed.")
  (expect (sacha-audio-braindump-get-subtitle-note-based-on-keywords "START SUMMARY ↪
  ↪ hello world STOP SUMMARY")
    :to-equal "SUMMARY: hello world")
  (expect (sacha-audio-braindump-get-subtitle-note-based-on-keywords "START CHAPTER ↪
  ↪ hello world again")
    :to-equal "CHAPTER: hello world again")
  )
)

```

7.7.4 Formatting the subtitles into Org Mode subtrees

The last step is to take the list of subtitles and format it into the subtree.

```

;; todo: sort the completion? ↪
  ↪ https://emacs.stackexchange.com/questions/55502/list-files-in-directory-in-reverse-order-of-date
;;
;; ##autoload
(defun sacha-audio-braindump-insert-subtitles-as-org-tree (vtt-filename)
  (interactive (list (read-file-name "VTT: " (expand-file-name "./" sacha-phone-recording-dir) ↪
    ↪ nil t nil
                      (lambda (s) (string-match "\\..vtt$" s)))))
  (let* ((subtitles
    (mapcar (lambda (sub)
      (unless (elt sub 4)

```

```

                (setf (elt sub 4)
                      (sacha-audio-braindump-get-subtitle-note-based-on-keywords (elt sub 3))))
        (sub)
        (subed-parse-file vtt-filename)))
    (start-date (sacha-audio-braindump-get-file-start-time vtt-filename))
    chapters tags
    start-of-entry)
(setq start-of-entry (point))
(insert (format "* TODO Review braindump from %s :braindump:\n\n" (file-name-base (vtt-filename)))
        (org-entry-put (point) "CREATED"
                        (concat "[" (format-time-string
                                      (cdr org-timestamp-formats)
                                      (sacha-audio-braindump-get-file-start-time
                                       (file-name-nondirectory vtt-filename))) "]")))

(insert
 (format "%s - %s - %s\n"
         (org-link-make-string (concat "file:" (file-name-sans-extension vtt-filename))
                               "VTT")
         (org-link-make-string (concat "file:" (file-name-sans-extension vtt-filename))
                               ".txt")
         (org-link-make-string (concat "file:" (file-name-sans-extension vtt-filename))
                               ".m4a")
         "Audio"))))

(save-excursion
 (insert "** Transcript\n")
 ;; add each subtitle; add an ID in case we change the title
 (mapc
  (lambda (sub)
    (when (elt sub 4)
      (let ((note (sacha-audio-braindump-get-subtitle-note-based-on-keywords (elt sub 3))))
        (insert (concat "*** "
                          note " "
                          (org-link-make-string
                           (format "subed:%s::%s"
                                   vtt-filename
                                   (sacha-msecs-to-timestamp (elt sub 1)))
                           "VTT")
                          "\n\n")))
        (org-entry-put (point) "CREATED"
                        (concat "[" (format-time-string
                                      (cdr org-timestamp-formats)
                                      (time-add start-date
                                                 (seconds-to-time (/ (elt sub 1) 1000.0)))) "]")))

        (org-entry-put (point) "START" (sacha-msecs-to-timestamp (elt sub 2)))
        (when (elt sub 4)
          (when (string-match "command: .*recognize" (elt sub 4))
            (save-excursion
              ;; TODO: scope this to just the section someday
              (goto-char start-of-entry)
              (org-set-tags (append (list "recognize") (org-get-tags)))))
          (when (string-match "command: .*outline" (elt sub 4))
            (save-excursion
              (goto-char start-of-entry)
              (org-set-tags (append (list "outline") (org-get-tags))))))

```

```

      (when (string-match "^time" (elt sub 4))
        (insert "[" (org-format-time-string (cdr org-timestamp-formats)
                                              (time-add start-date (seconds-to-time (/ ←
↪ (elt sub 1) 1000))))
          "]" \n"))
      (when (string-match "command: .+\\(high\\|low\\)" (elt sub 4))
        (save-excursion
          (goto-char start-of-entry)
          (org-priority (if (string= (downcase (match-string 1)) "high") ?A ?C)))
        (when (string-match "\\(?:tags?\\|keywords?\\): \\(\\.+\\)" (elt sub 4))
          (save-excursion
            (goto-char start-of-entry)
            (org-set-tags (append (split-string (match-string 1) " ") (org-get-tags))))))
        (add-to-list 'chapters
          (format "- %s (%s)"
            (org-link-make-string (concat "id:" (org-id-get-create))
                                  note)
            (org-link-make-string
              (format "subed:%s::%s"
                vtt-filename
                (sacha-msecs-to-timestamp (elt sub 1)))
              "VTT")))))
      (insert (elt sub 3) "\n")
      subtitles))
    (when chapters
      (insert (string-join (nreverse chapters) "\n") "\n"))))

```

```

;; ##autoload
(defun sacha-file-start-time (filename &optional base-date)
  "Return the local time based on FILENAME."
  (setq filename (file-name-base filename))
  (cond
    ((string-match ←
      ↪ "^\\([0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]\\)[-T]\\([0-9][0-9][\\.\\-][0-9][0-9]\\(?:[\\-\\.\\-][0-9][0-9]\\)\\)"
      ↪ filename)
      (date-to-time (concat (match-string 1 filename) "T"
        (replace-regexp-in-string "[\\-\\.]" ":" (match-string 2 filename)))))
    ((string-match "^\\(?:Copy of \\)?\\([ ]?[^ ]\\)[^ ]+ at \\([0-9]+\\)-\\([0-9]+\\)" ←
      ↪ filename)
      (let* ((day (match-string 1 filename))
              (hour (match-string 2 filename))
              (min (match-string 3 filename))
              (changed-time (or base-date (file-attribute-modification-time
                (file-attributes filename))))
              (decoded-time (decode-time changed-time)))
        ;; get the day on or before changed-time
        (if (string= (format-time-string "%a" changed-time) day)
          (encode-time (append
            (list
              0
              (string-to-number min)
              (string-to-number hour))
            (seq-drop decoded-time 3)))
          ;; synchronized maybe within the week after
          (let ((org-read-date-prefer-future nil))
            (org-read-date t t
              (concat "--" day " " hour ":" min)
              nil changed-time)))))))

```

```
(ert-deftest sach-file-start-time ()
  (should
    (equal (format-time-string "%Y-%m-%d %H:%M:%S"
      (sacha-file-start-time "2024-01-05-09-46-59.flv"))
      "2024-01-05 09:46:59"))
  (should
    (equal (format-time-string "%Y-%m-%d %H:%M:%S"
      (sacha-file-start-time "2024-01-08T12.49.vtt"))
      "2024-01-08 12:49:00"))
  (should
    (equal (format-time-string "%Y-%m-%d %H:%M:%S"
      (sacha-file-start-time "Sunday at 15-30.vtt"
        (date-to-time "2023-01-12")))
      "2023-01-08 15:30:00"))
  (should
    (time-equal-p (sacha-file-start-time "Sunday at 12-49.txt")
      (org-read-date t t "-sun 12:49")))))
```

```
(defalias 'sacha-audio-braindump-get-file-start-time #'sacha-file-start-time)
```

7.7.5 Process a single transcript from the raw text file

So now we put that all together: rename the file using the calculated start time, prepare the alignment breaks, align the file to get the timestamps, and add the subtree to an Org file.

```
(defvar sacha-audio-braindump-file "~/sync/orgzly/braindump.org")

;; ###autoload
(defun sacha-audio-braindump-make-todo (text-file &optional force)
  "Add TEXT-FILE as a TODO."
  (interactive (list (buffer-file-name) current-prefix-arg))
  ;; rename the files to use the timestamps
  (unless (string-match "[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]"
    ↪ (file-name-base text-file))
    ↪ (setq text-file (sacha-audio-braindump-rename-files-based-on-time text-file)))
  (let* ((recording (concat (file-name-sans-extension text-file) ".m4a"))
    (start (sacha-audio-braindump-get-file-start-time text-file))
    (vtt (concat (file-name-sans-extension text-file) ".vtt"))
    chapters
    (title (concat "Review braindump " text-file))
    existing)
    ;; check if already exists
    (with-current-buffer (find-file-noselect sacha-audio-braindump-file)
      (save-excursion
        (goto-char (point-min))
        (setq existing (org-find-exact-headline-in-buffer title))))
    (if (and existing (not force))
      (progn
        (message "Going to existing heading")
        (org-goto-marker-or-bmk existing))
      (if (or (null sacha-audio-braindump-last-processed-time)
        ↪ (time-less-p ↪
        ↪ sacha-audio-braindump-last-processed-time start))
        (customize-save-variable ↪
        ↪ 'sacha-audio-braindump-last-processed-time start)))
```

```

(find-file text-file)
(sacha-audio-braindump-prepare-alignment-breaks)
(save-buffer)
(when (file-exists-p vtt) (delete-file vtt))
(when (get-file-buffer vtt) (kill-buffer (get-file-buffer vtt)))
(subed-align recording text-file "VTT")
(when (get-file-buffer vtt) (kill-buffer (get-file-buffer vtt)))
(find-file sacha-audio-braindump-file)
(goto-char (point-min))
(if existing
    (progn
      (org-goto-marker-or-bmk existing)
      (delete-region (point) (org-end-of-subtree)))
    (org-next-visible-heading 1))
(sacha-audio-braindump-insert-subtitles-as-org-tree vtt)))

```

7.7.6 Process multiple files

I want to process multiple files in one batch.

```

;;;###autoload
(defun sacha-audio-braindump-process (files &optional force)
  (interactive (list (cond
    ↪
    ↪ (dired-get-marked-files) ↪
    ↪ (dired-get-marked-files) ↪
    ↪ ((derived-mode-p 'dired-mode) ↪
    ↪ (list (dired-get-filename))) ↪
    ↪ ((string-match "\\\\.txt$" (buffer-file-name)) ↪
    ↪ (list (buffer-file-name)) ↪
    ↪ (read-file-name "Transcript: ")) (t ↪
    ↪ current-prefix-arg)) ↪
    (mapc (lambda (f)
      (when (string-match "txt" f)
        (sacha-audio-braindump-make-todo f force))) ↪
    ↪ files))

```

It would be nice to have it automatically keep track of the latest one that's been processed, maybe via `customize-save-variable`. This still needs some tinkering with.

```

(defcustom sacha-audio-braindump-last-processed-time nil
  "The timestamp of the last processed transcript."
  :group 'sacha
  :type '(repeat integer))

;;;###autoload
(defun sacha-audio-braindump-process-since-last ()
  (interactive)

```



```

    (let ((files
            (seq-filter
              (lambda (f)
                (or (null ←
                    sach-audio-braindump-last-processed-time)
                    (time-less-p ←
                    sach-audio-braindump-last-processed-time
                    (sacha-audio-braindump-get-file-start-time f))))
              (directory-files sach-audio-braindump-dir 'full " ←
              at ←
              [0-9][0-9]-[0-9][0-9]\\\.txt\\|^([0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]T[0-9][0-9]\\\.([0-9][0-9]\\\.txt"))))
            (mapc (lambda (f)
                    (sacha-audio-braindump-make-todo f)
                    (let ((start ←
                        (sacha-audio-braindump-get-file-start-time f)))
                      (if (time-less-p ←
                          sach-audio-braindump-last-processed-time start)
                          (setq ←
                          sach-audio-braindump-last-processed-time start))))
                    files))
          (customize-save-variable 'sacha-audio-braindump-last-processed-time ←
          sach-audio-braindump-last-processed-time))

;; ##autoload
(defun sach-audio-braindump-new-filename (text-file &optional base-date)
  (if (string-match "[0-9][0-9][0-9][0-9]" text-file)
      text-file ; no change, already uses date
      (let* ((base (file-name-base text-file))
              (start (sacha-audio-braindump-get-file-start-time ←
              base base-date))
              (rest (if (string-match "[0-9][0-9][0-9][0-9]" base)
                          (match-string 2 base)
                          ""))
              (new-base (format-time-string "%Y-%m-%dT%H.%M" ←
              start)))
        (concat new-base rest "." (file-name-extension text-file))))

(ert-deftest sach-audio-braindump-new-filename ()
  (should
    (equal (sacha-audio-braindump-new-filename "Wednesday at 18-58.txt" (date-to-time ←
    "2023-01-01"))
            "2022-12-28T18.58.txt"))
  (should
    (equal (sacha-audio-braindump-new-filename "Wednesday at 18-58 extra text.txt" ←
    (date-to-time "2023-01-01"))
            "2022-12-28T18.58 extra text.txt")))

;; ##autoload
(defun sach-audio-braindump-rename-files-based-on-time (text-file)
  "Rename TEXT-FILE based on date. Return the new text file."
  (interactive (list (if (derived-mode-p 'dired-mode) (dired-get-filename)
    (buffer-file-name))))
  (if (string-match "[0-9][0-9][0-9][0-9]" text-file)
      text-file ; no change, already uses date
      (let ((new-name (sacha-audio-braindump-new-filename (file-name-nondirectory ←

```

```

↪ text-file))))
      (if (file-exists-p (expand-file-name new-name
↪
↪
↪ (file-name-directory text-file)))
          (error "%s already exists" new-base)
          (dolist (ext '(".txt" ".m4a" ".vtt"))
            (if (file-exists-p (concat (file-name-sans-extension ↪
↪ text-file) ext))
                (rename-file (concat ↪
↪ (file-name-sans-extension text-file) ext)
↪
↪ (expand-file-name (concat (file-name-sans-extension new-name) ext)
↪
↪
↪ (file-name-directory text-file))))))
      (expand-file-name new-name
↪
↪ (file-name-directory text-file))))))

```

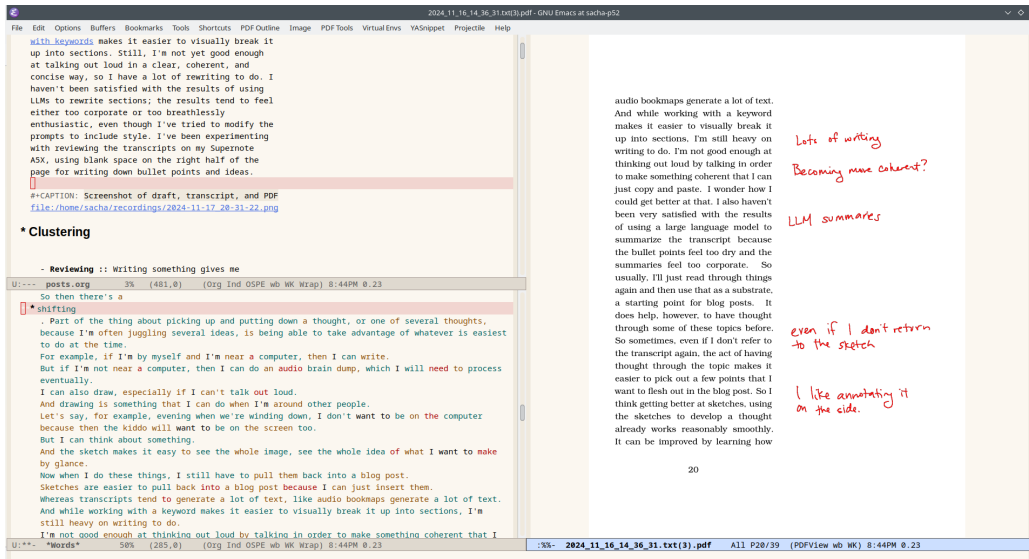
7.7.7 Ideas for next steps

- Make the commands process things even more automatically.
- Experiment with just sending everything to OpenAI Whisper instead of conditionally sending it based on the keywords (which might not be recognized).
- See if I want to reuse more sentences or move them around.
- Find out where people who have thought about dictation keywords have their notes; probably don't have to reinvent the wheel here

7.8 **DONE** Updating my audio braindump workflow to take advantage of WhisperX speechtotext.org

I get word timestamps for free when I transcribe with [WhisperX](#), so I can skip the Aeneas alignment step. That means I can update my [previous code for handling audio braindumps](#) . Breaking the transcript up into sections Also, I recently updated [subed-word-data](#) to [colour words based on their transcription score](#), which draws my attention to things that might be uncertain.

Here's what it looks like when I have the post, the transcript, and the annotated PDF.



Here's what I needed to implement `sacha-audio-braindump-from-whisperx-json` (plus some code from my previous audio braindump workflow):

```
;; ##autoload
(defun sacha-whisperx-word-list (file)
  (let* ((json-object-type 'alist)
         (json-array-type 'list))
    (seq-mapcat (lambda (seg)
                  (alist-get 'words seg))
                (alist-get 'segments seg)
                (json-read-file file))))

;; (seq-take (sacha-whisperx-word-list (sacha-latest-file "~/sync/recordings" "\\json") 10)
;; ##autoload
(defun sacha-whisperx-insert-word-list (words)
  "Inserts WORDS with text properties."
  (require 'subed-word-data)
  (mapc (lambda (word)
          (let ((start (point)))
            (insert
              (alist-get 'word word))
            (subed-word-data--add-word-properties start (point) word)
            (insert " ")))
        words))

;; ##autoload
(defun sacha-audio-braindump-turn-sections-into-headings ()
  (interactive)
  (goto-char (point-min))
  (while (re-search-forward "START SECTION \\.+\\. STOP SECTION" nil t)
    (replace-match
      (save-match-data
        (format
          "\n** %s\n"
          (save-match-data (string-trim (replace-regexp-in-string
            "^[,\\.]\|\\[,\\.]" " " (match-string 1))))))
      nil t)
    (let ((prop-match (save-excursion (text-property-search-forward
```

```

↳ 'subed-word-data-start))))
      (when prop-match
        (org-entry-put (point) "START" (format-seconds ↳
↳ "%02h:%02m:%02s" (prop-match-value prop-match))))))

;;###autoload
(defun sacha-audio-braindump-split-sentences ()
  (interactive)
  (goto-char (point-min))
  (while (re-search-forward "[a-z]\\." nil t)
    (replace-match (concat (string-trim (match-string 0)) "\\n") )))

;;###autoload
(defun sacha-audio-braindump-restructure ()
  (interactive)
  (goto-char (point-min))
  (sacha-subed-fix-common-errors)
  (org-mode)
  (sacha-audio-braindump-prepare-alignment-breaks)
  (sacha-audio-braindump-turn-sections-into-headings)
  (sacha-audio-braindump-split-sentences)
  (goto-char (point-min))
  (sacha-remove-filler-words-at-start))

;;###autoload
(defun sacha-audio-braindump-from-whisperx-json (file)
  (interactive (list (read-file-name "JSON: " "~/sync/recordings/" nil nil nil (lambda ↳
↳ (f) (string-match "\\..json\\" f)))))
  ;; put them all into a buffer
  (with-current-buffer (get-buffer-create "*Words*")
    (erase-buffer)
    (fundamental-mode)
    (sacha-whisperx-insert-word-list (sacha-whisperx-word-list file))
    (sacha-audio-braindump-restructure)
    (goto-char (point-min))
    (switch-to-buffer (current-buffer))))

;;###autoload
(defun sacha-audio-braindump-process-text (file)
  (interactive (list (read-file-name "Text: " "~/sync/recordings/" nil nil nil (lambda ↳
↳ (f) (string-match "\\..txt\\" f)))))
  (with-current-buffer (find-file-noselect file)
    (sacha-audio-braindump-restructure)
    (save-buffer)))

;; (sacha-audio-braindump-from-whisperx-json (sacha-latest-file "~/sync/recordings" "\\..json"))

```

Ideas for next steps:

- I can change my processing script to split up the Whisper TXT into sections and automatically make the PDF with nice sections.
- I can add reminders and other callouts. I can style them, and I can copy reminders into a different section for easier processing.
- I can look into extracting PDF annotations so that I can jump to the next highlight or copy highlighted text.

7.9 Markdown

```
(use-package markdown-mode
  :if sacha-laptop-p
  :mode ("\\.\\.\\(njk\\.\\.md\\.\\.\\)" . markdown-mode))
```

7.10 Screenshot

Based on https://www.reddit.com/r/emacs/comments/idz35e/emacs_27_can_take_svg_screenshots_of_itself/

```
(defvar sacha-screenshot-hook nil "Functions to run after a screenshot. Called with the ↵
  ↵ filename as the arg.")

;;###autoload
(defun sacha-screenshot (&optional filename current-screen caption)
  "Save a screenshot of the current frame as an SVG image.
Saves to a temp file and puts the filename in the kill ring.
Prompt for a caption afterwards."
  (interactive (list nil t))
  (if current-screen
      (setq filename (sacha-screenshot-current-screen filename))
      (setq filename (or filename
                        (expand-file-name
                         (format-time-string "%Y-%m-%d-%H-%M-%S.svg")
                         sacha-recordings-dir))))
  (with-temp-file filename
    (insert (x-export-frames nil 'svg))))
  (when (called-interactively-p 'any)
    (unless caption
      (save-window-excursion
        (with-current-buffer (find-file-noselect filename) (display-buffer (current-buffer)))
        (setq caption (read-string "Caption: "))))
    (when (and caption (not (string= caption "")))
      (let ((new-filename (concat
                          (file-name-sans-extension filename)
                          " " caption
                          "." (file-name-extension filename))))
        (rename-file filename new-filename t)
        (setq filename new-filename)))
      (kill-new filename)
      (message filename))
    (run-hook-with-args 'sacha-screenshot-hook filename)
    filename)

;;###autoload
(defun sacha-screenshot-current-screen (&optional filename)
  (interactive)
  (let ((new-file
        (or filename
            (expand-file-name
             (format-time-string "%Y-%m-%d-%H-%M-%S.png")
             sacha-recordings-dir))))
    (call-process
     "spectacle"
     nil nil nil
     "-b" "-m" "-n" "-o" new-file)
    new-file))
(defun sacha-save-screenshot (filename)
```

```
(org-capture-string filename "S"))
```

```
(add-hook 'sacha-screenshot-hook 'sacha-save-screenshot)
(keymap-global-set "C-c s" #'sacha-screenshot)
(keymap-global-set "s-s" #'sacha-screenshot)
```

7.11 Avoiding weasel words

```
(use-package artbollocks-mode
  :if sacha-laptop-p
  :defer t
  :load-path "~/elisp/artbollocks-mode"
  :config
  (progn
    (setq artbollocks-weasel-words-regex
      (concat "\\b" (regexp-opt
        '("one of the"
          "should"
          "just"
          "sort of"
          "a lot"
          "probably"
          "maybe"
          "perhaps"
          "I think"
          "really"
          "pretty"
          "nice"
          "action"
          "utilize"
          "leverage") t) "\\b")))
    ;; Don't show the art critic words, or at least until I figure
    ;; out my own jargon
    (setq artbollocks-jargon nil)))
```

7.12 Emacs: Cycle through different paragraph formats: all on one line, wrapped, max one sentence per line, one sentence per line

[2025-09-30 Tue]: Add `move-to-left-margin` to work around bug when using `fill-paragraph-sem1f` at the end of a paragraph.

I came across [Schauderbasis - reformat paragraph](#) via [@EFLS@mastodon.social](#). Now I want `M-q` to cycle through different ways of wrapping text:

- all on one line
- at most one sentence per line (don't even try to keep it within `fill-column`).
- at most one sentence per line (although still wrapping at `fill-column`)
- according to `fill-column`

[https://sachachua.com/blog/2025/09/emacs-cycle-through-different-paragraph-formats-all-on-one-line-wrapped-max-one-sentence-per-line-one-sentence-per-line/sacha-reformat-paragraph-or-region.webm?caption=Screencast cycling through different paragraph formats](https://sachachua.com/blog/2025/09/emacs-cycle-through-different-paragraph-formats-all-on-one-line-wrapped-max-one-sentence-per-line-one-sentence-per-line/sacha-reformat-paragraph-or-region.webm?caption=Screencast%20cycling%20through%20different%20paragraph%20formats)

Now that [semantic linefeeds are part of core Emacs](#) (as of 2025-06-14), the code for cycling through different paragraph formats can be pretty short. Most of it is actually just the logic for cycling through different commands. That might come in handy elsewhere. There's an [unfill](#) package as well, but since the code for unfilling a paragraph is very simple, I'll just include that part.

Note that `fill-paragraph-semelf` pays attention to `sentence-end-double-space`, and it doesn't handle comments yet. I also have some code to check if I'm in a comment and skip those filling methods if needed.

This might encourage me to write shorter sentences. I can move sentences around with `M-Shift-up` and `M-Shift-down` in Org Mode, which is pretty handy. Also, one sentence per line makes diffs easier to read. But wrapped text is annoying to edit in Orgzly Revived on my phone, because the wrapping makes a very ragged edge on a narrow screen. I might unwrap things that I want to edit there. With a little bit of tweaking to skip source blocks, I can narrow to the subtree, select my whole buffer, and cycle the formatting however I like.

```
(defvar sacha-repeat-counter '())
  "How often `sacha-repeat-next' was called in a row using the same command.
  This is an alist of (cat count list) so we can use it for different functions.")

;;;###autoload
(defun sacha-unfill-paragraph ()
  "Replace newline chars in current paragraph by single spaces.
  This command does the inverse of `fill-paragraph'."
  (interactive)
  (let ((fill-column most-positive-fixnum))
    (fill-paragraph)))

;;;###autoload
(defun sacha-fill-paragraph-semelf-long ()
  (interactive)
  (let ((fill-column most-positive-fixnum))
    (fill-paragraph-semelf)))

;;;###autoload
(defun sacha-repeat-next (category &optional element-list reset)
  "Return the next element for CATEGORY.
  Initialize with ELEMENT-LIST if this is the first time."
  (let* ((counter
          (or (assoc category sacha-repeat-counter)
              (progn
                (push (list category -1)
                      sacha-repeat-counter)
                (assoc category
                      sacha-repeat-counter))))
         (setf (elt (cdr counter) 0)
               (mod
                (if reset 0 (1+ (elt (cdr counter) 0)))
                (length (elt (cdr counter) 1)))))
        (elt (elt (cdr counter) 1) (elt (cdr counter) 0))))

;;;###autoload
(defun sacha-in-prefixed-comment-p ()
  (or (member 'font-lock-comment-delimiter-face (face-at-point nil t))
      (member 'font-lock-comment-face (face-at-point nil t))
      (save-excursion
```


Sometimes I use [writeroom-mode](#) to make the lines look even narrower, with lots of margin on the side.

Related:

- [Text Editor Judo: One Sentence Per Line - The Daily Macro](#) - in Org Mode, one sentence per line lets you drag sentences around with M-Shift-up and M-Shift-down
- [Writing one sentence per line | Derek Sivers \(HN\)](#)
- [Semantic Linefeeds](#)
- [Fill paragraph using semantic linefeeds](#)

7.13 Visual line

```
(global-visual-line-mode)
(add-hook 'minibuffer-mode-hook (lambda () (visual-line-mode -1)))
```

7.14 Unicode

```
(defmacro sach-insert-unicode (unicode-name)
  `(lambda () (interactive)
    (insert-char (cdr (assoc-string ,unicode-name (ucs-names))))))
(keymap-global-set "C-x 8 s" (sach-insert-unicode "ZERO WIDTH SPACE"))
(keymap-global-set "C-x 8 S" (sach-insert-unicode "SNOWMAN"))
```

7.15 Clean up spaces

```
(keymap-global-set "M-SPC" 'cycle-spacing)
```

7.16 Expand

```
(setq save-abbrevs 'silently)
(keymap-global-set "M-/" 'hippie-expand)
```

From <https://github.com/purcell/emacs.d/blob/master/lisp/init-auto-complete.el> - Exclude very large buffers from dabbrev

```
;; ###autoload
(defun sanityinc/dabbrev-friend-buffer (other-buffer)
  (< (buffer-size other-buffer) (* 1 1024 1024)))
(setq dabbrev-friend-buffer-function 'sanityinc/dabbrev-friend-buffer)
```

```
(setq hippie-expand-try-functions-list
  '(yas-hippie-try-expand
    try-expand-all-abbrevs
    try-complete-file-name-partially
    try-complete-file-name
    try-expand-dabbrev
    try-expand-dabbrev-from-kill
    try-expand-dabbrev-all-buffers
```

```
try-expand-list
try-expand-line
try-complete-lisp-symbol-partially
try-complete-lisp-symbol))
```

7.17 Write about keybindings

```
; ; hmm, doesn't quite work for looking things up yet. I basically want a programmatic ↵
↳ where-is for a specific keymap
(defvar sachal-keybinding-maps '(subed-mode-map subed-waveform-minor-mode-map ↳
↳ subed-waveform-svg-map))
; ;; ##autoload
(defun sachal-copy-keybinding (symbol)
  (interactive (list (find-function-read)))
  (when (listp symbol)
    (setq symbol (car symbol)))
  (let (result keys)
    (map-keymap
     (lambda (event def)
       (cond ((and (symbolp def))
              (push (list def event) result))
             ((and (listp def) (eq 'keymap (car def)))
              (apply 'append
                     (map-keymap
                      ↵
                      ↵
                      ↵
                      (push (list def event) result))))
            (def))))))
      subed-mode-map)
  (setq keys (assoc-default symbol result))
  (when keys
    (kill-new (key-description keys))
    (message "%s" (key-description keys))))
```

7.18 Transcripts from my phone

audio:editing

```
(defvar sach-audio-braindump-dir "~/sync/Phone")

;;###autoload
(defun sach-open-latest-braindump ()
  (interactive)
  (find-file (sach-latest-file sach-audio-braindump-dir "\\*.txt"))
  (kill-new (buffer-string)))

;;###autoload
(defun sach-insert-latest-braindump ()
  (interactive)
  (insert-file-contents (sach-latest-file sach-audio-braindump-dir "\\*.txt")))

;;###autoload
(defun sach-audio-braindump-dired ()
  (interactive)
  (dired sach-audio-braindump-dir "-lt"))
```

```
(defalias 'sacha-phone-dired #'sacha-audio-braindump-dired)
```

7.19 Speech recognition

audio:SpeechRecognition

```
(use-package caser
  :bind
  ("M-D" . caser-dashcase-dwim))
```

7.19.1 Map lang-gptel feedback from the logbook to KwizIQ topics

ai

```
;; ##autoload
(defun sacha-org-collect-logbook-contents ()
  "Collect contents of all LOGBOOK drawers in the current subtree.
Returns them concatenated as a string."
  (save-excursion
    (org-back-to-heading t)
    (let ((subtree-end (save-excursion (org-end-of-subtree t)))
          contents
          elem)
      (while (re-search-forward "[^\\t]*:LOGBOOK:[^\\t]*$" subtree-end t)
        (setq elem (org-element-at-point))
        (push (buffer-substring-no-properties
              (org-element-contents-begin elem)
              (org-element-contents-end elem))
              contents))
      (string-join (nreverse contents) "\n"))))

;; ##autoload
(defun sacha-org-get-subtree (link)
  (save-window-excursion
    (save-excursion
      (org-link-open-from-string link)
      (buffer-substring-no-properties (point) (progn (org-end-of-subtree) (point))))))

;; ##autoload
(defun sacha-learn-lang-gptel-analyze-feedback ()
  (interactive)
  (with-current-buffer (get-buffer-create "*Feedback*")
    (erase-buffer)
    (org-mode))
  (gptel-request
    (json-encode
      `(("feedback on previous mistakes" . ,(sacha-org-collect-logbook-contents))
        ("topic links" . ,(sacha-org-get-subtree ←
        ↪ "[[file:~/sync/orgzly/organizer.org::#kwiziq-a2]]")
        ("prompt" . "Analyze the feedback on previous mistakes. Map them to the different ←
        ↪ topics and create a frequency table where column A has a link to the topic and column ←
        ↪ B has the number of errors in that category. For anything that doesn't match, ←
        ↪ summarize them in a separate list called Other. Also create a 10-item quiz covering ←
        ↪ the most important points. Hide answers like this: [[answer:the answer goes ←
        ↪ here][__]] Use Org Mode syntax.")))
    :callback (lambda (response info)
      (with-current-buffer (get-buffer-create "*Feedback*")
        (insert response)
        (goto-char (point-min))
```

```
(pop-to-buffer (current-buffer))))))
```

Hmm, I think it messes up ROT13, so I need a different way of obscuring answers. Maybe it can do Org Mode links. I might need to do that myself somehow.

Example:

Fréquence des erreurs par catégorie

Thématique (KwizIQ)	Nombre d'erreurs
Nouns & Articles	18
Adjectives & Adverbs	14
Verbs Tenses & Conjugation	12
Prepositions & Conjunctions	11
Pronouns	9
Idioms & Structures	7
Numbers, Time & Date	8

Other

- Terminologie Technique : Confusion entre API (Interface) et AFI (Phonétique).
- Anglicismes : "Possiblement" au lieu de "peut-être", "retour" au lieu de "commentaire".
- Orthographe : "Minutieux", "fonctionnalité", "nécessaire".
- Typographie : Utilisation de l'espace comme séparateur de milliers (31 000) au lieu de la virgule ou du point.

Quiz de révision (10 questions) Hmm, let me try filling in this quiz...

1. Complétez : "J'évite que ma voix ne se fatigue (fatiguer)." (Subjonctif présent)
2. Quel est l'antonyme de "plus de" dans une comparaison de quantité ? moins de
3. Corrigez : "C'est une bonne occasion de documenter mon processus."
4. Traduisez "251 hours" en lettres (attention au genre d'heure). deux cents cinquante une heure
5. Corrigez : "Ces souvenirs me rappellent certains moments" (Ajoutez le pronom manquant). ??
6. Choisissez : On joue du piano mais on joue au tennis.
7. Quelle est la forme correcte : "Une fois que j'ai fini mes enregistrements, je les écoute" (Habitude présente).
8. Corrigez l'accord : "La transcription est précise."
9. Traduisez "I have a lot to do" en utilisant "rien ne me presse". Rien ne me presse de faire
10. Quelle est l'abréviation correcte pour l'alphabet phonétique en français ? AFI

7.19.2 Using whisper.el to convert speech to text and save it to the currently clocked task in Org Mode or elsewhere speech

- [2026-01-30 Fri]: Major change: I switched to [my fork](#) of natrys/whisper.el so that I can specify functions that change the window configuration etc.
- [2026-01-13 Tue]: Change main function to `sacha-whisper-run`, use seq-reduce to go through the functions.

- [2026-01-09 Fri]: Added code for automatically capturing screenshots, saving text, working with a list of functions.
- [2026-01-08 Thu]: Added demo, fixed some bugs.
- [2026-01-04 Sun]: Added note about difference from MELPA package, fixed :vc

I want to get my thoughts into the computer quickly, and talking might be a good way to do some of that. [OpenAI Whisper](#) is reasonably good at recognizing my speech now and [whisper.el](#) gives me a convenient way to call [whisper.cpp](#) from Emacs with a single keybinding. (Note: This is not the same [whisper](#) package as the one on MELPA.) Here is how I have it set up for reasonable performance on my Lenovo P52 with just the CPU, no GPU.

I've bound `<f9>` to the command `whisper-run`. I press `<f9>` to start recording, talk, and then press `<f9>` to stop recording. By default, it inserts the text into the buffer at the current point. I've set `whisper-return-cursor-to-start` to `nil` so that I can keep going.

```
(use-package whisper
  :vc (:url "https://github.com/natrys/whisper.el")
  :load-path "~/vendor/whisper.el"
  :config
  (setq whisper--mode-line-recording-indicator "🔊")
  (setq whisper-quantize "q4_0")
  (setq whisper-install-directory "~/vendor")
  (setq whisper--install-path (concat
    (expand-file-name (file-name-as-directory whisper-install-directory))
    "whisper.cpp/"))
  ;; Get it running with whisper-server-mode set to nil first before you switch to 'local.
  ;; If you change models,
  ;; (whisper-install-whispercpp (whisper--check-install-and-run nil "whisper-start"))
  (setq whisper-server-mode 'local)
  (setq whisper-return-cursor-to-start nil)
  ;(setq whisper--ffmpeg-input-device ←
    ↪ "alsa_input.usb-Blue_Microphones_Yeti_Stereo_Microphone_REV8-00.analog-stereo")
  (setq whisper--ffmpeg-input-device "VirtualMicSink.monitor")
  (setq whisper-language "en")
  (setq whisper-recording-timeout 3000)
  (setq whisper-before-transcription-hook nil)
  (setq whisper-use-threads (1- (num-processors)))
  (setq whisper-transcription-buffer-name-function 'whisper--simple-transcription-buffer-name)
  (add-hook 'whisper-after-transcription-hook 'sacha-subed-fix-common-errors-from-start -100)
  :bind
  ((("<f9>" . whisper-run)
   ("C-<f9>" . sacha-whisper-run)
   ("S-<f2>" . whisper-run)
   ("S-<f9>" . sacha-whisper-replay)
   ("M-<f9>" . sacha-whisper-toggle-language))))
```

Let's see if we can process "Computer remind me to...":

```
(defvar sacha-whisper-org-reminder-template "t")

;;;###autoload
(defun sacha-whisper-org-process-reminder ()
  (let ((text (buffer-string))
        reminder)
    (when (string-match "computer[,\\.]? reminds? me to \\((.+\\)" text)
      (setq reminder (match-string 1 text))
      (save-window-excursion
```

```

(with-current-buffer (if (markerp whisper--marker) (marker-buffer whisper--marker) ←
  ↪ (current-buffer))
  (when (markerp whisper--marker) (goto-char whisper--marker))
  (org-capture nil sachawhisper-org-reminder-template)
  (insert reminder)
  (org-capture-finalize)))
(erase-buffer)))

```

```

(with-eval-after-load 'whisper
  (add-hook 'whisper-after-transcription-hook 'sachawhisper-org-process-reminder 50))

```

Disk space is inexpensive and backups are great, so let's save each file using the timestamp.

```

(defvar sachawhisper-dir "~/recordings/whisper/")
;;;autoload
(defun sachawhisper-set-temp-filename ()
  (setq whisper--temp-file (expand-file-name
                            (format-time-string "%Y-%m-%d-%H-%M-%S.wav")
                            sachawhisper-dir)))

```

```

(with-eval-after-load 'whisper
  (add-hook 'whisper-before-transcription-hook #'sachawhisper-set-temp-filename))

```

The technology isn't quite there yet to do real-time audio transcription so that I can see what it understands while I'm saying things, but that might be distracting anyway. If I do it in short segments, it might still be okay. I can replay the most recently recorded snippet in case it's missed something and I've forgotten what I just said.

```

;;;autoload
(defun sachawhisper-replay (&optional file)
  "Replay the last temporary recording."
  (interactive (list
    (when current-prefix-arg
      (read-file-name "File: " sachawhisper-dir)))))
  (setq whisper--temp-file (or file whisper--temp-file))
  (mpv-play whisper--temp-file))

;;;autoload
(defun sachawhisper-insert-retry (&optional file)
  (interactive (list
    (when current-prefix-arg
      (read-file-name "File: " sachawhisper-dir)))))
  (whisper--cleanup-transcription)
  (setq whisper--marker (point-marker))
  (setq whisper--temp-file (or file whisper--temp-file))
  (whisper--transcribe-audio))

```

Il peut aussi comprendre le français.

```

;;;autoload
(defun sachawhisper-toggle-language ()
  "Set the language explicitly, since sometimes auto doesn't figure out the right one."
  (interactive)
  (setq whisper-language (if (string= whisper-language "en") "fr" "en"))
  ;; If using a server, we need to restart for the language

```

```
(when (process-live-p whisper--server-process) (kill-process whisper--server-process))
(message "%s" whisper-language))
```

I could use this with `org-capture`, but that's a lot of keystrokes. My shortcut for `org-capture` is `C-c r`. I need to press at least one key to set the template, `<f9>` to start recording, `<f9>` to stop recording, and `C-c C-c` to save it. I want to be able to capture notes to my currently clocked in task without having an Org capture buffer interrupt my display.

To clock in, I can use `C-c C-x i` or my `! speed command` (p. 196). Bonus: the modeline displays the current task to keep me on track, and I can use `org-clock-goto` (which I've bound to `C-c j`) to jump to it.

Then, when I'm looking at something else and I want to record a note, I can press `<f9>` to start the recording, and then `C-<f9>` to save it to my currently clocked task along with a link to whatever I'm looking at. (Update: Ooh, now I can save a screenshot too.)

```
;; ##autoload
(defun sacha-whisper-reset (text)
  (setq sacha-whisper-skip-annotation nil)
  (remove-hook 'whisper-insert-text-at-point #'sacha-whisper-org-save-to-clocked-task)
  text)
```

```
;; Only works with my tweaks to whisper.el
;; https://github.com/sachac/whisper.el/tree/whisper-insert-text-at-point-function
(with-eval-after-load 'whisper
  (setq whisper-insert-text-at-point
    '(sacha-whisper-handle-commands
      sacha-whisper-save-text
      sacha-whisper-save-to-file
      sacha-whisper-maybe-expand-snippet
      sacha-speech-input-quantified-track
      sacha-whisper-maybe-type
      sacha-whisper-maybe-type-with-hints
      sacha-whisper-insert
      sacha-whisper-reset)))
```

```
(defvar sacha-whisper-last-annotation nil "Last annotation so we can skip duplicates.")
(defvar sacha-whisper-skip-annotation nil)
(defvar sacha-whisper-target-markers nil "List of markers to send text to.")

;; ##autoload
(defun sacha-whisper-insert (text)
  (let ((markers
    (cond
      ((null sacha-whisper-target-markers)
        (list whisper--marker)) ; current point where whisper was started
      ((listp sacha-whisper-target-markers)
        sacha-whisper-target-markers)
      ((markerp sacha-whisper-target-markers)
        (list sacha-whisper-target-markers)))))
    (orig-point (point))
    (orig-buffer (current-buffer)))
    (when text
      (mapcar (lambda (marker)
        (with-current-buffer (marker-buffer marker)
          (save-restriction
            (widen)
```

```

      (when (markerp marker) (goto-char marker))
      (when (and (derived-mode-p 'org-mode) (org-at-drawer-p))
        (insert "\n"))
      (whisper--insert-text
       (concat
        (if (looking-back "[ \t\n]\\|^")
            ""
            " ")
        (string-trim text)))
      ;; Move the marker forward here
      (move-marker marker (point))))
    markers)
  (when sachawhisper-target-markers
    (goto-char orig-point))
  nil)))

```

```

;;;###autoload
(defun sachawhisper-maybe-type (text)
  "If Emacs is not the focused app, simulate typing TEXT.
Add this function to `whisper-insert-text-at-point'."
  (when text
    (if (frame-focus-state)
        text
        (make-process :name "xdotool" :command
                      (list "xdotool" "type"
                            text)))
    nil)))

```

```

;;;###autoload
(defun sachawhisper-clear-markers ()
  (interactive)
  (setq sachawhisper-target-markers nil))

;;;###autoload
(defun sachawhisper-use-current-point (&optional add)
  (interactive (list current-prefix-arg))
  (if add
      (push (point-marker) sachawhisper-target-markers)
      (setq sachawhisper-target-markers (list (point-marker)))))

;;;###autoload
(defun sachawhisper-run-at-point (&optional add)
  (interactive (list current-prefix-arg))
  (sachawhisper-clear-markers)
  (whisper-run))

```

```

(keymap-global-set "<f9>" #'sachawhisper-run-at-point)
(keymap-global-set "<kp-1>" #'whisper-run)

```

```

;;;###autoload
(defun sachawhisper-jump-to-marker ()
  (interactive)
  (with-current-buffer (marker-buffer (car sachawhisper-target-markers))
    (goto-char (car sachawhisper-target-markers)))))

```



```

;;;###autoload
(defun sacha-whisper-use-currently-clocked-task (&optional add)
  (interactive (list current-prefix-arg))
  (save-window-excursion
    (save-restriction
      (save-excursion
        (org-clock-goto)
        (org-end-of-meta-data)
        (org-end-of-subtree)
        (if add
          (push (point-marker) sacha-whisper-target-markers)
          (setq sacha-whisper-target-markers (list (point-marker))))))))

;;;###autoload
(defun sacha-whisper-run (&optional skip-annotation)
  (interactive (list current-prefix-arg))
  (require 'whisper)
  (add-hook 'whisper-insert-text-at-point #'sacha-whisper-org-save-to-clocked-task -10)
  (whisper-run)
  (when skip-annotation
    (setq sacha-whisper-skip-annotation t)))

;;;###autoload
(defun sacha-whisper-save-text (text)
  "Save TEXT beside `whisper--temp-file'."
  (when text
    (let ((link (org-store-link nil)))
      (with-temp-file (concat (file-name-sans-extension whisper--temp-file) ".txt")
        (when link
          (insert link "\n"))
        (insert text)))
      text))

;;;###autoload
(defun sacha-whisper-org-save-to-clocked-task (text)
  (when text
    (save-window-excursion
      (with-current-buffer (if (markerp whisper--marker) (marker-buffer whisper--marker) ↵
        ↵ (current-buffer))
        (when (markerp whisper--marker) (goto-char whisper--marker))
        ;; Take a screenshot maybe
        (let* ((link (and (not sacha-whisper-skip-annotation)
                          (org-store-link nil)))
              (region (and (region-active-p) (buffer-substring (region-beginning) ↵
        ↵ (region-end))))
          (screenshot-filename
            (when (or
                  (null link)
                  (not (string= sacha-whisper-last-annotation link))
                  (not (frame-focus-state)))) ; not in focus, take a screenshot
              (sacha-screenshot-current-screen (concat (file-name-sans-extension ↵
        ↵ whisper--temp-file) ".png")))))
        (if (org-clocking-p)
          (save-window-excursion
            (save-restriction
              (save-excursion
                (org-clock-goto)
                (org-end-of-subtree)
                (unless (bolp)

```

```

      (insert "\n"))
    (insert "\n")
    (if (and link (not (string= sachawhisper-last-annotation link)))
      (insert
        (if screenshot-filename
          (concat "(" (org-link-make-string
            (concat "file:" screenshot-filename)
            "screenshot") ") ")
          "")
        link
        "\n")
      (when screenshot-filename
        (insert (org-link-make-string
          (concat "file:" screenshot-filename)
          "screenshot")
          "\n")))
    (when region
      (insert "#+begin_example\n" region "\n#+end_example\n"))
    (insert text "\n")
    (setq sachawhisper-last-annotation link))
  (run-at-time 0.5 nil (lambda (text) (message "Added clock note: %s" text)) text))
;; No clocked task, prompt for a place to capture it
(kill-new text)
(setq org-capture-initial text)
(call-interactively 'org-capture)
;; Delay the window configuration
(let ((config (current-window-configuration)))
  (run-at-time 0.5 nil
    (lambda (text config)
      (set-window-configuration config)
      (message "Copied: %s" text))
    text config))))))

```

```

(with-eval-after-load 'org
  (add-hook 'org-clock-in-hook #'sachawhisper-org-clear-saved-annotation))

```

```

;;;###autoload
(defun sachawhisper-org-clear-saved-annotation ()
  (setq sachawhisper-org-last-annotation nil))

```

Here's an idea for a function that saves the recognized text with a timestamp.

```

(defvar sachawhisper-notes "~/sync/stream/narration.org")
;;;###autoload
(defun sachawhisper-save-to-file (text)
  (when text
    (let ((link (org-store-link nil)))
      (with-current-buffer (find-file-noselect sachawhisper-notes)
        (goto-char (point-max))
        (insert "\n\n" (format-time-string "%H:%M ") text "\n" (if link (concat link "\n") ""))
        (save-buffer)
        (run-at-time 0.5 nil (lambda (text) (message "Saved to file: %s" text)) text)))
    text))

```

And now I can redo things if needed:

```
;;###autoload
(defun sacha-whisper-redo ()
  (interactive)
  (setq whisper--marker (point-marker))
  (whisper--transcribe-audio))
```

I think I've just figured out my Pipewire setup so that I can record audio in OBS while also being able to do speech to text, without the audio stuttering. [qpwgraph](#) was super helpful for visualizing the Pipewire connections and fixing them.

```
systemctl --user restart pipewire pipewire-pulse wireplumber
sleep 3
pactl load-module module-null-sink \
  sink_name="VirtualMicSink" sink_properties=device.description=VirtualMicSink
pactl load-module module-null-sink \
  sink_name="CombinedSink" sink_properties=device.description=CombinedSink
if pactl list short sources | grep -i pci-0000; then
  pactl load-module module-loopback \
    source="alsa_input.pci-0000_00_1f.3.analog-stereo" \
    sink="VirtualMicSink" \
    latency_msec=200 \
    adjust_time=3 \
    source_output_properties="node.description='SysToVMic' node.name='SysToVMic' ↔
    ↔ media.name='SysVToMic'" \
    sink_input_properties="node.description='SysToVMic' node.name='SysToVMic' ↔
    ↔ media.role='filter'"
  pactl load-module module-loopback \
    source="alsa_output.pci-0000_00_1f.3.analog-stereo.monitor" \
    sink="CombinedSink" \
    node_name="SystemOutToCombined" \
    source_output_properties="node.description='SysOutToCombined' ↔
    ↔ node.name='SysOutToCombined'" \
    sink_input_properties="node.description='SysOutToCombined' node.name='SysOutToCombined' ↔
    ↔ media.role='filter'" \
    latency_msec=200 adjust_time=3
fi
if pactl list short sources | grep -i yeti; then
  pactl load-module module-loopback \
    source="alsa_input.usb-Blue_Microphones_Yeti_Stereo_Microphone_REV8-00.analog-stereo" \
    sink="VirtualMicSink" \
    latency_msec=200 \
    adjust_time=3 \
    source_output_properties="node.description='YetiToVMic' node.name='YetiToVMic' ↔
    ↔ media.name='YetiToVMic'" \
    sink_input_properties="node.description='YetiToVMic' node.name='YetiToVMic' ↔
    ↔ media.role='filter'"
  pactl load-module module-loopback \
    ↔
    ↔ source="alsa_output.usb-Blue_Microphones_Yeti_Stereo_Microphone_REV8-00.analog-stereo.monitor" ↔
    ↔ \
    sink="CombinedSink" \
    source_output_properties="node.description='YetiOutToCombined' ↔
    ↔ node.name='YetiOutToCombined' media.name='YetiOutToCombined'" \
    sink_input_properties="node.description='YetiOutToCombined' node.name='YetiOutToCombined' ↔
    ↔ media.role='filter'" \
    latency_msec=200 adjust_time=3
fi
```

```

pactl load-module module-loopback \
  source="VirtualMicSink.monitor" \
  sink="CombinedSink" \
  source_output_properties="node.description='VMicToCombined' node.name='VMicToCombined' ↔
  ↳ media.name='VMicToCombined'" \
  sink_input_properties="node.description='VMicToCombined' node.name='VMicToCombined' ↔
  ↳ media.role='filter'" \
  latency_msec=200 adjust_time=3

pactl load-module module-null-sink \
  sink_name="ExtraSink1" sink_properties=device.description=ExtraSink1

pactl load-module module-loopback \
  source="ExtraSink1.monitor" \
  sink="CombinedSink" \
  source_output_properties="node.description='ExtraSink1ToCombined' ↔
  ↳ node.name='ExtraSink1ToCombined' media.name='ExtraSink1ToCombined'" \
  sink_input_properties="node.description='ExtraSink1ToCombined' ↔
  ↳ node.name='ExtraSink1ToCombined' media.role='filter'" \
  latency_msec=200 adjust_time=3

if pactl load-module module-remap-source \
  source_name="VirtualMic" \
  master="VirtualMicSink.monitor" \
  source_properties="device.description='VirtualMic'" 2>/dev/null; then
  pactl set-default-source VirtualMic
else
  pactl set-default-source VirtualMicSink.monitor
fi

```

Here's a demo:

https://sachachua.com/blog/2026/01/using-whisper-el-to-capture-text-to-speech-in-emacs/2026-01-08_1.17.22.webm?caption=Screencastofusingwhisper.eltodospeech-to-textintothecurrentbuffer,clocked-intask,orotherfunctioncaptions=t

nil

And then I define a global shortcut in KDE that runs:

```
/home/sacha/bin/xdotool-emacs key --clearmodifiers F9
```

So now I can dictate into other applications or save into Emacs. Which suggests of course that I should get it working with C-f9 as well, if I can avoid the keyboard shortcut loop...

7.19.3 Emacs and whisper.el: Trying out different speech-to-text backends and models emacs:SpeechRecognition

I was curious about [parakeet](#) because I heard that it was faster than Whisper on the [HuggingFace leaderboard](#). When I installed it and got it running on my laptop (CPU only, no GPU), it seemed like my results were a little faster than whisper.cpp with the large model, but much slower than whisper.cpp with the base model. The base model is decent for quick dictation, so I got curious about other backends and other models.

In order to try [natrys/whisper.el](#) with other backends, I needed to work around how whisper.el validates the model names and sends requests to the servers. Here's the quick and dirty code for doing so, in case you want to try it out for yourself.

```
(defvar sachawhisper-url-format "http://s:%d/transcribe")
;;;###autoload
(defun sachawhisper--transcribe-via-local-server ()
  "Transcribe audio using the local whisper server."
  (message "[%] Transcribing via local server")
  (whisper--setup-mode-line :show 'transcribing)
  (whisper--ensure-server)
  (setq whisper--transcribing-process
    (whisper--process-curl-request
      (format sachawhisper-url-format whisper-server-host whisper-server-port)
      (list "Content-Type: multipart/form-data")
      (list (concat "file=@" whisper--temp-file)
            "temperature=0.0"
            "temperature_inc=0.2"
            "response_format=json"
            (concat "model=" whisper-model)
            (concat "language=" whisper-language))))))
;;;###autoload
(defun sachawhisper--check-model-consistency () t)
```

I moved my server code to [speech-input-speeches.el](#).

```
(with-eval-after-load 'whisper
  (advice-add 'whisper--transcribe-via-local-server :override ←
    ↪ #'sachawhisper--transcribe-via-local-server)
  (advice-add 'whisper--check-model-consistency :override ←
    ↪ #'sachawhisper--check-model-consistency)
  (advice-add 'whisper--ensure-server :override #'speech-input-speeches-server-start)
)
```

Then I have this function for trying things out.

```
(defun sachatest-whisper-api (url &optional args)
  (with-temp-buffer
    (apply #'call-process "curl" nil t nil "-s"
      url
      (append (mapcan
        (lambda (h) (list "-H" h))
        (list "Content-Type: multipart/form-data"))
        (mapcan
        (lambda (h) (list "-F" h))
        (list (concat "file=@" whisper--temp-file)
              "temperature=0.0"
              "temperature_inc=0.2"
              "response_format=verbose_json"
              (concat "language=" whisper-language))))
        args))
    (message "%s %s" (buffer-string) url)))
```

Here's the audio file. It is around 10 seconds long. I run the benchmark 3 times and report the average time.

: [Download](#)

```
(let ((times '3))
  (mapcar
    (lambda (group)
```

```

(let ((whisper--temp-file "/home/sacha/recordings/whisper/2026-01-19-14-17-53.wav"))
  ;; warm up the model
  (eval (cadr group))
  (list
    (format "%.3f"
      (/ (car
          (benchmark-call (lambda () (eval (cadr group))) times))
         times))
    (car group))))
'
("parakeet"
 (sacha-test-whisper-api
  (format "http://%s:%d/v1/audio/transcriptions" whisper-server-host 5092)))
("whisper.cpp base-q4_0"
 (sacha-test-whisper-api
  (format "http://%s:%d/inference" whisper-server-host 8642)))
("speaches whisper-base"
 (sacha-test-whisper-api
  (format "http://%s:%d/v1/audio/transcriptions" whisper-server-host 8001)
  (list "-F" "model=Systran/faster-whisper-base"))))
("speaches whisper-base.en"
 (sacha-test-whisper-api
  (format "http://%s:%d/v1/audio/transcriptions" whisper-server-host 8001)
  (list "-F" "model=Systran/faster-whisper-base.en"))))
("speaches whisper-small"
 (sacha-test-whisper-api
  (format "http://%s:%d/v1/audio/transcriptions" whisper-server-host 8001)
  (list "-F" "model=Systran/faster-whisper-small"))))
("speaches whisper-small.en"
 (sacha-test-whisper-api
  (format "http://%s:%d/v1/audio/transcriptions" whisper-server-host 8001)
  (list "-F" "model=Systran/faster-whisper-small.en"))))
("speaches lorneluo/whisper-small-ct2-int8"
 (sacha-test-whisper-api
  (format "http://%s:%d/v1/audio/transcriptions" whisper-server-host 8001)
  (list "-F" "model=lorneluo/whisper-small-ct2-int8"))))
; needed export TORCH_FORCE_NO_WEIGHTS_ONLY_LOAD=1
("whisperx-server Systran/faster-whisper-small"
 (sacha-test-whisper-api
  (format "http://%s:%d/transcribe" whisper-server-host 8002))))
)

```

3.694	parakeet
2.484	whisper.cpp base-q4_0
1.547	speaches whisper-base
1.425	speaches whisper-base.en
4.076	speaches whisper-small
3.735	speaches whisper-small.en
2.870	speaches lorneluo/whisper-small-ct2-int8
4.537	whisperx-server Systran/faster-whisper-small

All the results above were CPU-only, no GPU acceleration.

I tried it with:

- [parakeet](#)
- [whisper.cpp](#) (as [whisper.el](#) sets it up)

- [speaches](#), which is a front-end for [faster-whisper](#), and
- [whisperx-server](#), which is a front-end for [whisperx](#)

[2026-04-05 Sun] Update: After a lot of rebooting, I figured out how to get the Nvidia P1000 GPU on my Lenovo P52 to work for up to the Whisper medium models. Large models don't fit in the 4GB of VRAM it has. Here are the benchmarking results:

```
(let ((times '3))
  (mapcar
    (lambda (group)
      (let ((whisper--temp-file "/tmp/2026-01-19-14-17-53.wav"))
        ;; warm up the model
        (eval (cadr group))
        (list
          (format "%.3f"
            (/ (car
              (benchmark-call (lambda () (eval (cadr group))) times))
              times))
          (car group))
        ))
      '("speaches whisper-tiny"
        (sacha-test-whisper-api
          (format "http://s:%d/v1/audio/transcriptions" "localhost" 8000)
          (list "-F" "model=Systran/faster-whisper-tiny.en"))))
        ("speaches whisper-base"
          (sacha-test-whisper-api
            (format "http://s:%d/v1/audio/transcriptions" "localhost" 8000)
            (list "-F" "model=Systran/faster-whisper-base.en"))))
        ("speaches whisper-small"
          (sacha-test-whisper-api
            (format "http://s:%d/v1/audio/transcriptions" "localhost" 8000)
            (list "-F" "model=Systran/faster-whisper-small.en"))))
        ("speaches whisper-medium"
          (sacha-test-whisper-api
            (format "http://s:%d/v1/audio/transcriptions" "localhost" 8000)
            (list "-F" "model=Systran/faster-whisper-medium.en"))))))
```

```
0.596  speaches whisper-tiny
0.940  speaches whisper-base
2.909  speaches whisper-small
8.740  speaches whisper-medium
```

That probably means I can upgrade to using the small model on a regular basis. For large models, I can use the command-line tool, since the overhead of loading the model is probably small compared to the processing time. I could also run two Speaches servers (one on CPU and one on GPU), but I don't think I'll need that level yet.

I like how speaches lets me switch models on the fly, so maybe I can use small.en generally and switch to a different model when I want to try dictating in French. Here's how I've set it up to use the server.

```
(defvar sacha-speech-input-model-aliases
  '(("small" . "Systran/faster-whisper-small.en")
    ("medium" . "Systran/faster-whisper-medium.en")
    ("base" . "Systran/faster-whisper-base.en")
    ("tiny" . "Systran/faster-whisper-tiny.en"))
```

```

("large" . "Systran/faster-whisper-large-v2"))))

(defun sacha-speech-input-set-model (model-name)
  "Change the speech recognition model to MODEL-NAME.
Use `sacha-speech-input-model-aliases' for aliases."
  (interactive (list (speech-input-speeches-read-model-name)))
  (when (assoc-default model-name sacha-speech-input-model-aliases #'string=)
    (setq model-name (assoc-default model-name sacha-speech-input-model-aliases #'string=)))
  (setq whisper-model model-name)
  (setq speech-input-model model-name))

```

```

(setq whisper-server-port 8000
      whisper-model "Systran/faster-whisper-small.en"
      sacha-whisper-url-format "http://%s:%d/v1/audio/transcriptions")

```

Benchmark notes: I have a Lenovo P52 laptop (released 2018) with an Intel Core i7-8850H (6 cores, 12 threads; 2.6 GHz base / 4.3 GHz turbo) with 64GB RAM and an SSD. It has a soldered-on Nvidia P1000 with 4GB of VRAM.

7.19.4 Queuing multiple transcriptions with `whisper.el` speech recognition `SpeechRecognition:emacs`

I want to be able to talk out loud and have the ideas go into Emacs. I can do this in a number of different ways:

1. I briefly demonstrated a step-by-step approach with [natrys/whisper.el with a single file](#). I press a keyboard shortcut to start the recording, another shortcut to stop the recording, and it transcribes it in the background. But the way `whisper.el` is set up is that if I press the keyboard shortcut to start recording again it will offer to interrupt the transcription process, which is not what I want. I want to just keep talking and have it process results as things come in.
2. I'm also experimenting with [Google Chrome's web speech API to do continuous speech recognition](#) (p. 166), which I can get into Emacs using a web socket.
3. What I've just figured out is how to layer a semi-continuous interface for speech recognition on top of `whisper.el` so that while it's processing in the background, I can just press a keyboard shortcut (I'm using numpad 9 to call `sacha-whisper-continue`) to stop the previous recording, queue it for processing, and start the next recording. If I use this keyboard shortcut to separate my thoughts, then Whisper has a much easier time making sense of the whole sentence or paragraph or whatever, instead of trying to use the sliding 30 second context window that many streaming approaches to speech recognition try to use.

Question: Did you fix the keyboard delay you've got while speech catches what you're saying?

Sometimes, when the speed recognition kicks in, my computer gets busy. When my computer gets really busy, it doesn't process my keystrokes in the right order, which is very annoying because then I have to delete the previous word and retype it. I haven't sorted that out yet, but it seems like I probably have to lower the priority on different processes. On the plus side, as I mentioned, if I dictate things instead of typing them, then I don't run into that problem at all.

Also, other notes on delays: The continuous speech recognition via Google Chrome shows up fairly quickly, but it's not very precise, and it doesn't have punctuation. Even if there's a little bit of a delay, as long as I press the `sacha-whisper-continue` shortcut after each thought, then I can get that text into my Emacs buffer using the nicer transcription from my selected model. There is going to be a bit of a delay for that one because it gets processed at the end of the thought. Also, I need to start thinking in complete sentences instead of just adding one cause after the other as my brain goes on all of these tangents. I think it's pretty promising. There's the continuous speech recognition via

Google Chrome if I don't mind the lower accuracy and lack of punctuation, and I can still get the pretty version on the other side.

Why talk out loud? I liked the [Bookclub Tapas](#) presentation that Maddie Sullivan did at EmacsConf 2025. Talking out loud helps me be a lot more verbose about what I'm saying, compared to typing things out or even like having to switch to my notes or interrupting my screen with an Org capture buffer. Of course I want to clean that up for putting into a blog post, but given that my life still sometimes has random interruptions from a kiddo who must have my attention at that very minute, having that kind of record that I can at least try to reread afterwards to reconstruct what I was thinking about sounds like it might be helpful.

Still, making sense out loud is hard. I'm not actually used to talking to people that much now. This is probably a good reason for me to experiment with streaming more. Then I get the practice in talking out loud, there are backup recordings, and people can ask questions when things are unclear.

Of course, sometimes the text doesn't quite make sense because of the speech recognition errors. I can usually figure it out from the context. I save the audio as well so that I can go back and listen to it again if I really need to.

Anyway, here's the code for sending the current recording to whisper in the background and starting another recording. It assumes a lot about how things are set up. For example, I'm only testing this with a local speeches server instead of whisper.cpp. You might need to look at my other speech related configuration blog posts and sections in order to make sense of it.

```
(defvar sacha-whisper--queue nil)
;;;###autoload
(defun sacha-whisper-continue (&optional arg)
  "Send what we've got so far for transcription and then continue recording.
Call with \\[universal-argument] to signal that we can stop."
  (interactive "P")
  (require 'whisper)
  (if arg
      (sacha-whisper-done)
      (setq whisper--marker (point-marker) whisper--point-buffer (current-buffer))
      (when (process-live-p whisper--recording-process)
        ;; queue only if the last one is not asking for the same file
        (unless
            (string=
             (plist-get
              (car
               (last sacha-whisper--queue))
              :file)
             whisper--temp-file)
            (add-to-list
             'sacha-whisper--queue
             (list :file whisper--temp-file
                   :buffer
                   (format "%s" (file-name-base whisper--temp-file)))
             t))
        ;; Remove the sentinel; handle results ourselves
        (set-process-sentinel whisper--recording-process
                              (lambda (process event)
                                (sacha-whisper-process-queue)))
        (interrupt-process whisper--recording-process))
      (run-hooks 'whisper-before-transcription-hook)
      (whisper--setup-mode-line :show 'recording)
      (whisper--record-audio)))
;;;###autoload
```

```

(defun sacha-whisper-discard ()
  "Ignore the previous recording."
  (interactive)
  (when (process-live-p whisper--recording-process)
    ;; Remove the sentinel; handle results ourselves
    (set-process-sentinel whisper--recording-process
      (lambda (process event)
        (when (file-exists-p whisper--temp-file)
          (delete-file whisper--temp-file))
        (sacha-whisper-process-queue)))
    (interrupt-process whisper--recording-process)))

;;;###autoload
(defun sacha-whisper-discard-and-continue ()
  "Ignore the previous recording and continue."
  (interactive)
  (if (process-live-p whisper--recording-process)
      (progn
        ;; Remove the sentinel; handle results ourselves
        (set-process-sentinel whisper--recording-process
          (lambda (process event)
            (sacha-whisper-process-queue)
            (sacha-whisper-continue)))
        (interrupt-process whisper--recording-process))
      (sacha-whisper-continue)))

;;;###autoload
(defun sacha-whisper-done ()
  (interactive)
  (when (process-live-p whisper--recording-process)
    (add-to-list
      'sacha-whisper--queue
      (list :file whisper--temp-file
            :buffer
            (format "%s*" (file-name-base whisper--temp-file))))
    t)
  ;; Remove the sentinel; handle results ourselves
  (set-process-sentinel whisper--recording-process
    (lambda (process event)
      (sacha-whisper-process-queue)))
  (whisper--setup-mode-line :hide 'recording)
  (interrupt-process whisper--recording-process)))

;;;###autoload
(defun sacha-whisper-process-queue-result ()
  "Process the first part of the queue that already has results."
  (while (plist-get (car sacha-whisper--queue) :results)
    (let ((o (pop sacha-whisper--queue)))
      (unless sacha-whisper-target-markers
        (setq whisper--marker (point-marker)
              whisper--point-buffer (current-buffer)))
      (with-current-buffer (plist-get o :buffer)
        (erase-buffer)
        (insert (plist-get o :results))))
    ;; Only works with my fork: ↵
    ↵ https://github.com/sachac/whisper.el/tree/whisper-insert-text-at-point-function
    (whisper--handle-transcription-output nil (plist-get o :buffer))))

;;;###autoload

```

```

(defun sacha-whisper-process-queue ()
  (let (o)
    (while (setq o (seq-find (lambda (o) (and (plist-get o :file)
                                              (not (plist-get o :process))
                                              (not (plist-get o :results))))
                          sacha-whisper--queue))
      (let* ((headers (list "Content-Type: multipart/form-data"))
             (params (list (concat "file=@"
                                   (plist-get o :file))
                           "temperature=0.0"
                           "temperature_inc=0.2"
                           "response_format=json"
                           (concat "model=" whisper-model)
                           (concat "language=" whisper-language)))
             (url (format sacha-whisper-url-format whisper-server-host whisper-server-port))
             (command `("curl" "-s"
                        ,url
                        ,@ (mapcan (lambda (h) (list "-H" h)) headers)
                        ,@ (mapcan (lambda (p) (list "-F" p)) params))))
        (with-current-buffer (get-buffer-create (plist-get o :buffer))
          (erase-buffer))
        (plist-put
         o :process
         (make-process
          :name "whisper-curl"
          :command command
          :buffer (plist-get o :buffer)
          :coding 'utf-8
          :sentinel
          (lambda (process event)
            (with-current-buffer (process-buffer process)
              (let ((current sacha-whisper--queue-item))
                (when (and (get-buffer (plist-get current :buffer))
                          (string-equal "finished\n" event))
                  (with-current-buffer (plist-get current :buffer)
                    (goto-char (point-min))
                    (plist-put current :results
                               (or
                                (condition-case nil
                                  (gethash "text" (json-parse-buffer))
                                  (error ""))
                                "error"))))))
              (sacha-whisper-process-queue-result))))
          (plist-put o :command (string-join command " "))
          (with-current-buffer (process-buffer (plist-get o :process))
            (setq-local sacha-whisper--queue-item o))))))
  (defvar-local sacha-whisper--queue-item nil)

;; ##autoload
(defun sacha-whisper-reprocess-queue ()
  (interactive)
  (setq whisper--marker (point-marker) whisper--point-buffer (current-buffer))
  (mapc (lambda (o)
    (when (process-live-p (plist-get o :process))
      (kill-process (plist-get o :process)))
    (when (get-buffer (plist-get o :buffer))
      (kill-buffer (plist-get o :buffer)))
    (plist-put o :process nil)
    (plist-put o :results nil))
    sacha-whisper--queue-item))

```

```

      sachawhisper--queue)
    (sachawhisper-process-queue))

;;;###autoload
(defun sachawhisper-clear-queue ()
  (interactive)
  (mapc (lambda (o)
    (when (process-live-p (plist-get o :process))
      (kill-process (plist-get o :process)))
    (when (get-buffer (plist-get o :buffer))
      (kill-buffer (plist-get o :buffer)))
    (plist-put o :process nil)
    (plist-put o :results nil))
    sachawhisper--queue)
  (setq sachawhisper--queue nil))

(defvar-keymap sachawhisper-simulated-continuous-mode-map
  :doc "Keymap for sachawminor-mode."
  "S-<f2>" #'sachawhisper-continue
  )

(define-minor-mode sachawhisper-simulated-continuous-mode
  "Simulate continuous speech recognition by queuing."
  :lighter "W"
  (if sachawhisper-simulated-continuous-mode
    (message "Start speaking ... ")
    (message "All done.")
    (sachawhisper-done)))

```

```

(keymap-global-set "<kp-9>" #'sachawhisper-continue)
(keymap-global-set "<kp-8>" #'sachawhisper-discard-and-continue)
(keymap-global-set "C-<kp-9>" #'sachawhisper-done)
(with-eval-after-load 'sachawhisper
  (define-key sachawhisper-simulated-continuous-mode-map [remap whisper-run] ↔
    ↪ #'sachawhisper-continue))

```

7.19.5 Using Silero voice activity detection to automatically queue multiple transcriptions with `na-trys/whisper.el` SpeechRecognition:emacs

[2026-03-27 Fri]: Moved code to [sachac/speech-input](#).

I can [queue multiple transcriptions with whisper.el](#) so that they get processed sequentially with backup audio. It catches up when I pause to think. Now I want to use [Silero voice activity detection](#) to do that kind of segmentation for me automatically.

First, I need a Python server that can print out events when it notices the start or stop of a speech segment. If I print out the timestamps, I might be able to cross-reference it someday with interesting things. For now, even just paying attention to the end of a segment is enough for what I want to do.

nil

I've made the code to [speech-input-vad.el](#), so now I can use `speech-input-vad-start` to start monitoring speech events.

```
(use-package speech-input
```

```
:load-path "~/proj/speech-input/"
:preface (load "~/proj/speech-input/speech-input-autoloads.el" nil t)
)
```

Because I added Pulse properties to the process environment, I can easily use [epwgraph](#) to rewire the input so that it gets the input from my VirtualMicSink instead of the default system audio device. (Someday I'll figure out how to specify that as the input automatically.)

Then I can start this process from Emacs:

```
(with-eval-after-load 'speech-input-vad
  (add-hook 'speech-input-vad-on-end-functions #'sacha-whisper-maybe-continue))
```

```
;;###autoload
(defun sacha-whisper-maybe-continue ()
  (when (process-live-p whisper--recording-process)
    (sacha-whisper-continue)))
```

Now I can press my shortcut for `sacha-whisper-continue` to start the process. As I keep talking, it will continue to record. When I pause for more than a second between sentences, then it will send that chunk to the server for transcription without me having to press another button, while still listening for more speech.

How is this different from the streaming approach that many real-time speech recognition services offer? I think this gives me a bit more visibility into and control of the process. For my personal use, I don't need to have everything processed as quickly as possible, and I'm not trying to replicate live captions. I just want to be able to look back over the last five minutes to try to remember what I was talking about. I usually have a lot of quiet time as I think through my next steps, and it's fine to have it catch up then. I also like that I can save time-stamped audio files for later processing, divided according to the speech segments. Those might be a little bit easier to work with when I get around to compositing them into a video.

7.19.6 Slowly building speech-based commands for Emacs

I'd like to be able to dictate to Org Mode more fluidly, which includes being able to break thoughts up into paragraphs or add to lists. Here's a possible starting point that works with my other experiments in [speech recognition](#)

```
(defvar sacha-whisper-commands
  '(("scroll up" . scroll-down-command)
    ("scrolling up" . scroll-down-command)
    ("page up" . scroll-down-command)
    ("scroll down" . scroll-up-command)
    ("scroll down" . scroll-up-command)
    ("page down" . scroll-up-command)
    ("next page" . scroll-up-command)
    ("close other windows" . delete-other-windows)
    ("run the buffer" . eval-buffer)
    ("mark buffer" . mark-whole-buffer)
    ("mark paragraph" . mark-paragraph)
    ("expand" . expand-region)
    ("start emacs news" . sacha-workflow-emacs-news-start)
    ("update emacs calendar" . sacha-workflow-emacs-calendar-update)
  )
  "Commands for speech recognition.")
```

```

;;;###autoload
(defun sacha-whisper-handle-commands (text)
  ;; Let's do commands at the beginning of a speech segment for now
  (if (string-match (concat "^" (regexp-opt (mapcar 'car sacha-whisper-commands)) "\\>")
    text)
    (progn
      (while (string-match (concat "\\(" (regexp-opt (mapcar 'car sacha-whisper-commands)) "↵"
        ↵ "\\>[\\.,\\?]? *")
          text)
        (let* ((match (match-string 1 text))
          (func (assoc-default (downcase match) sacha-whisper-commands #'string=)))
          (when func
            (message "Command: %s" match)
            (setq text (replace-match "" nil nil text))
            (cond
              ((commandp func)
               (call-interactively func))
              ((functionp func)
               (funcall func))))))
        text)
      text))

(defvar sacha-whisper-replacements
  '((" *\\<start \\(list\\|next\\) item\\>[\\.,] *" . "\n- ")
    (" *\\<start check ?box\\>[\\.,] *" . "\n- [ ] ")
    (" *start paragraph[\\.,]? *" . "\n\n")))

;;;###autoload
(defun sacha-whisper-process-replacements ()
  (goto-char (point-min))
  (when (looking-at " ") (replace-match ""))
  (let ((case-fold-search t))
    (cond
      ((re-search-forward " *okay[\\.,]? stop recording" nil t)
       (when (process-live-p whisper--recording-process)
         (replace-match "")
         (message "Stopping.")
         (sacha-whisper-done))))))
  (dolist (rep sacha-whisper-replacements)
    (goto-char (point-min))
    (while (re-search-forward (car rep) nil t)
      (replace-match (cdr rep))))
  (goto-char (point-max))
  (insert " "))

```

```

(with-eval-after-load 'whisper
  (add-hook 'whisper-after-transcription-hook 'sacha-whisper-process-replacements 70))

```

7.19.7 Okay, track

```

(defvar sacha-quantified-common-categories
  '(("Emacs" . "Discretionary - Productive - Emacs")
    ("Child care" . "Childcare")
    ("French" . "Discretionary - French")
    ("Brigade" . "Discretionary - Productive - Bike Brigade")
    ("Consulting" . "E1 Gen")))

```

```
;;###autoload
(defun sacha-speech-input-quantified-track (text)
  "Start tracking time."
  (if (and text
        (string-match "^ok\\(?::ay\\)?[,\\.]? track \\(\\.+\\)" text))
      (let ((category
              (speech-input-match-in-list
               (match-string 1 text)
               (mapcar 'car sacha-quantified-common-categories))))
          (message "Tracking %s" category)
          (quantified-track
           (assoc-default category sacha-quantified-common-categories #'string=)
           nil)
          text))
      text))
```

7.19.8 Using speech recognition for on-the-fly translations in Emacs and faking in-buffer completion for the results emacs:speech

[2026-03-31 Tue]: I added the recognized text so that I can confirm what was translated. I also moved sacha-type-with-hint to [learn-lang-type-with-hint](#).

When I'm writing a journal entry in French, I sometimes want to translate a phrase that I can't look up word by word using a dictionary. Instead of switching to a browser, I can use an Emacs function to prompt me for text and either insert or display the translation.

```
;;###autoload
(defun sacha-learn-lang-en-to-fr (text &optional display-only)
  (interactive (list (read-string "Text: ") current-prefix-arg))
  (let* ((url "https://translation.googleapis.com/language/translate/v2")
         (params `(("key" . ,(getenv "GOOGLE_API_KEY"))
                    ("q" . ,text)
                    ("source" . "en")
                    ("target" . "fr")
                    ("format" . "text")))
         (query-string (mapconcat
                        (lambda (pair)
                          (format "%s=%s"
                                    (url-hexify-string (car pair))
                                    (url-hexify-string (cdr pair))))
                        params
                        "&"))
         (full-url (concat url "?" query-string)))
    (let* ((response (plz 'get full-url :as #'json-read))
           (data (alist-get 'data response))
           (translations (alist-get 'translations data))
           (first-translation (car translations))
           (translated-text (alist-get 'translatedText first-translation)))
      (when (called-interactively-p 'any)
        (if display-only
            (message "%s" translated-text)
            (insert translated-text)))
      translated-text)))
```

I think it would be even nicer if I could use speech synthesis, so I can keep it a little more separate from my typing thoughts. I want to be able to say "Okay, translate ..." or "Okay, ... in French"

to get a translation. I've been using [my fork of natrys/whisper.el](#) for [speech recognition](#) in English, and I like it a lot. By adding a function to `whisper-after-transcription-hook`, I can modify the intermediate results before they're inserted into the buffer.

```
;; ##autoload
(defun sacha-whisper-translate ()
  (goto-char (point-min))
  (let ((case-fold-search t))
    (when (re-search-forward "okay[,\\.]? translate[,\\.]? \\(\\.+\\)\\|okay[,\\.]? \\(\\.+?\\) in ↵
    ↵ French" nil t)
      (let* ((s (or (match-string 1) (match-string 2)))
             (translation (save-match-data (sacha-learn-lang-en-to-fr s))))
        (replace-match
         (propertize translation
                      'type-hint translation
                      'type-original s
                      'help-echo s))))))
```

```
(with-eval-after-load 'whisper
  (add-hook 'whisper-after-transcription-hook 'sacha-whisper-translate 70))
```

But that's too easy. I want to actually type things myself so that I get more practice. Something like an autocomplete suggestion would be handy as a way of showing me a hint at the cursor. The usual completion-at-point functions are too eager to insert things if there's only one candidate, so we'll just fake it with an overlay. This code works only with my `whisper.el` fork because it supports using a list of functions for `whisper-insert-text-at-point`.

```
;; ##autoload
(defun sacha-whisper-maybe-type-with-hints (text)
  "Add this function to `whisper-insert-text-at-point'."
  (let* ((hint (and text (org-find-text-property-in-string 'type-hint text)))
         (original (and text (org-find-text-property-in-string 'type-original text))))
    (if hint
        (progn
          (learn-lang-type-with-hint hint original)
          nil)
        text)))
```

Here's a demonstration of me saying "Okay, this is a test, in French.":

<https://sachachua.com/blog/2026/02/using-speech-recognition-for-on-the-fly-translations-in-emacs-and-faking-in-buffer-completion-for-the-results/2026-02-27-translate-and-type-with-hints.webm?caption=ScreenCast> of using speech recognition to translate into French and provide a hint when typing

Since we're faking in-buffer completion here, maybe we can still get away with considering this as an entry for [Emacs Carnival February 2026: Completion ? =\)](#)

7.19.9 Expanding yasnippets by voice in Emacs and other applications

[Yasnippet](#) is a template system for Emacs. I want to use it by voice. I'd like to be able to say things like "Okay, define interactive function" and have that expand to a matching snippet in Emacs or other applications. Here's a quick demonstration of expanding simple snippets:

https://sachachua.com/blog/2026/03/expanding-yasnippet-by-voice/2026-03-04_12.33.19.webm?caption=ScreenCastofexpandingasnippetbyvoiceinEmacsandinothertapplicationscptions=t

nil

This is handled by the following code:

```
(declare-function 'subed-word-data-compare-normalized-string-distance "subed-word-data")

;;;###autoload
(defun sacha-whisper-maybe-expand-snippet (text)
  "Add to `whisper-insert-text-at-point'."
  (if (and text
        (string-match
         "^ok\\(?::ay\\)?[,\\.]? \\((.+\\)" text))
      (let* ((name
              (downcase
               (string-trim
                (replace-regexp-in-string "[,\\.]" "" (match-string 1 text))))
             (matching
              (seq-find (lambda (o)
                         (subed-word-data-compare-normalized-string-distance
                          name
                          (downcase (yas--template-name o))))
                        (yas--all-templates (yas--get-snippet-tables)))))
            (if matching
                (progn
                 (if (frame-focus-state)
                     (progn
                      (yas-expand-snippet matching)
                      nil)
                     ;; In another application
                     (with-temp-buffer
                      (yas-minor-mode)
                      (yas-expand-snippet matching)
                      (buffer-string))))
                  text))
        text))
    text))
```

This code relies on [my fork of whisper.el](#), which lets me specify a list of functions for `whisper-insert-text-at-point`. (I haven't asked for upstream review yet because I'm still testing things, and I don't know if it actually works for anyone else yet.) It does approximate matching on the snippet name using a function from [subed-word-data.el](#) which just uses `string-distance`. I could probably duplicate the function in my config, but then I'd have to update it in two places if I come up with more ideas.

The code for inserting into other functions is defined in [sacha-whisper-maybe-type](#), which is very simple:

```
;;;###autoload
(defun sacha-whisper-maybe-type (text)
  "If Emacs is not the focused app, simulate typing TEXT.
Add this function to `whisper-insert-text-at-point'."
  (when text
    (if (frame-focus-state)
        text
        (make-process :name "xdotool" :command
                      (list "xdotool" "type"
                            text))
        nil)))
```

Someday I'd like to provide alternative names for snippets. I also want to make it easy to fill in

snippet fields by voice. I'd love to be able to answer minibuffer questions from `yas-choose-value`, `yas-completing-read`, and other functions by voice too. Could be fun!

Related:

- [My snippets repository](#)
- [My topic page for speech recognition](#)

7.19.10 Streaming speech recognition into Emacs using Google Chrome Web Speech API

```
(defvar sachaspeech-chrome-ws nil "Websocket for connecting.")
(defvar sachaspeech-chrome-url "ws://127.0.0.1:8000/ws" "Websocket URL to connect to for ↔
↳ captions.")
;;;###autoload
(defun sachaspeech-chrome-connect ()
  (interactive)
  (unless (websocket-p sachaspeech-chrome-ws)
    (setq sachaspeech-chrome-ws
      (websocket-open
        sachaspeech-chrome-url
        :on-message #'sachaspeech-chrome-handle))))
;;;###autoload
(defun sachaspeech-chrome-reconnect ()
  (interactive)
  (sachaspeech-chrome-disconnect)
  (sachaspeech-chrome-connect))

(defvar sachaspeech-functions '(sachaspeech-display)
  "Functions to run with the info as an argument.
The info is an alist with 'type and 'content.
The functions are called in sequence, with the first function getting the info
from the websocket message and the other functions getting the results of the
previous functions.")

(defvar-local sachaspeech-previous-final nil)

(deface sachaspeech-chrome-caption-current
  '((t :height 200))
  "Display current caption.")
;;;###autoload
(defun sachaspeech-display (info)
  (let-alist info
    (with-current-buffer (get-buffer-create
      (format "*Captions - %s*"
        .session))
      (when (and (string= .type "TEMP")
        sachaspeech-previous-final)
        (goto-char (point-max))
        (delete-region
          (line-beginning-position)
          (line-end-position))
        (insert (propertize sachaspeech-previous-final
          'face `(:foreground ,(modus-themes-get-color-value 'fg-dim)))
          "\n")
        (setq sachaspeech-previous-final nil))
      (when (string= .type "TEMP")
        (goto-char (point-max))
```

```

      (delete-region (line-beginning-position) (line-end-position))
      (insert .type (propertize .content 'face 'sacha-chrome-caption-current)))
    (when (string= .type "FINAL")
      (unless (string= sacha-speech-previous-final .content)
        (goto-char (point-max))
        (set-text-properties
         (line-beginning-position)
         (line-end-position)
         (list
          'face `(:foreground ,(modus-themes-get-color-value 'fg-dim))))
        (insert "\n"))
      (setq sacha-speech-previous-final .content)
      (goto-char (point-max))
      (delete-region (line-beginning-position) (line-end-position))
      (insert (propertize .content 'face 'sacha-chrome-caption-current))))))
info)

```

```

;;;###autoload
(defun sacha-speech-chrome-handle (_ frame)
  (let* ((info (json-parse-string (websocket-frame-text frame)
                                   :object-type 'alist)))
    (seq-reduce (lambda (prev cur)
                  (funcall cur prev))
                sacha-speech-functions info)))

;;;###autoload
(defun sacha-speech-chrome-disconnect ()
  (interactive)
  (websocket-close sacha-speech-chrome-ws)
  (setq sacha-speech-chrome-ws nil))

(defvar sacha-speech-chrome-recognition-server-process nil)
(defvar sacha-speech-chrome-dir "~/proj/emacs-web-speech")
;;;###autoload
(defun sacha-chrome-ensure-speech-recognition-server ()
  (interactive)
  (unless (process-live-p sacha-speech-chrome-recognition-server-process)
    (let ((default-directory sacha-speech-chrome-dir))
      (setq sacha-speech-chrome-recognition-server-process
            (make-process
             :name "live-captioning"
             :buffer "*live-captioning*"
             :command (list (expand-file-name ".venv/bin/python3") "app.py"))
            (sit-for 1))))

;;;###autoload
(defun sacha-chrome-stop-speech-recognition-server ()
  (interactive)
  (when (process-live-p sacha-speech-chrome-recognition-server-process)
    (kill-process sacha-speech-chrome-recognition-server-process)))

(defvar-local sacha-speech-chrome-session nil)
(defvar-local sacha-speech-chrome-user-dir nil)
(defvar-local sacha-speech-chrome-lang "en-US")
;;;###autoload
(defun sacha-speech-chrome-new-session (&optional id lang local-only)
  (interactive (list
                (file-name-base

```

```

        (make-temp-name
         (expand-file-name "chrome-"
                           (temporary-file-directory))))
        sacha-speech-chrome-lang
        current-prefix-arg))
(sacha-chrome-ensure-speech-recognition-server)
(let* ((base-id (file-name-base id))
      (user-dir
       (if (file-exists-p
            (expand-file-name
             id
             (temporary-file-directory)))
           (make-temp-file "chrome-" t)
           ;; small race condition, but this is fine
           (expand-file-name
            id
            (temporary-file-directory)))))
  process
  (process-environment
   (append
    (list
     (format
      "PULSE_SOURCE=%s"
      sacha-speech-input)
     (format
      "PULSE_PROP=node.description='%s' media.name='%s' node.name='%s'"
      base-id base-id base-id))
    process-environment)))
;; Hook it up to sacha-speech-input by default
(setq process
  (make-process
   :name "chrome"
   :buffer "*chrome*"
   :command (list
             "google-chrome"
             "--disable-fre"
             "--no-default-browser-check"
             "--no-first-run"
             (concat "--user-data-dir=" (shell-quote-argument user-dir))
             (format "http://127.0.0.1:8000/?session=%s&lang=%s&local=%s"
                     (url-hexify-string base-id)
                     lang
                     (if local-only "1" ""))
             (concat "--class=" (shell-quote-argument base-id)))
   :sentinel
   (lambda (process event)
     ;; Clean up afterwards
     (cond
      ((string-match "finished\\|deleted\\|exited\\|failed\\|core dumped" event)
       (with-current-buffer (process-buffer process)
        (when sacha-speech-chrome-user-dir
         (delete-directory sacha-speech-chrome-user-dir t))))))
   (with-current-buffer (process-buffer process)
    (setq-local sacha-speech-chrome-user-dir user-dir))
   (switch-to-buffer (format "*Captions - %s*" base-id))
   (setq-local sacha-speech-chrome-session base-id)
   (sacha-speech-chrome-connect)))

```

Now I get to figure out how to use the buffers.

```
;;;###autoload
(defun sacha-speech-sessions ()
  (seq-keep (lambda (o)
    (with-current-buffer o
      (when sacha-speech-session
        (cons sacha-speech-session o))))
    (buffer-list)))

;;;###autoload
(defun sacha-speech-clear-all ()
  (interactive)
  (dolist (session (sacha-speech-sessions))
    (sacha-speech-clear session)))

;;;###autoload
(defun sacha-speech-clear (session)
  (interactive (list (sacha-speech-select-session)))
  (with-current-buffer (cdr session)
    (erase-buffer)
    (setq-local sacha-speech-previous-final nil)))

;;;###autoload
(defun sacha-speech-select-session (&optional prompt)
  (let ((sessions (sacha-speech-sessions)))
    (if (= (length sessions) 1)
      (car sessions)
      (assoc
        (completing-read
          (or prompt "Session: ")
          (mapcar 'car sessions))
        sessions))))

(defvar-local sacha-speech-input "VirtualMicSink:input")

;;;###autoload
(defun sacha-speech-rewire (&optional id input)
  "Unhook it from all input and reconnect it to `sacha-speech-input'.
Call with \\[universal-argument] to specify the input."
  (interactive (list (sacha-speech-select-session)
    (if current-prefix-arg
      (epwgraph-complete-logical-node-name)
      sacha-speech-input)))
  (with-current-buffer (cdr id)
    (setq input (or input sacha-speech-input))
    (setq-local sacha-speech-input input)
    (let* ((node-name (concat (car id) ":input"))
      (session-ports (epwgraph-get-ports-with-logical-name
        node-name))
      (new-ports (if (stringp input)
        (epwgraph-get-ports-with-logical-name input)
        input))
      (old-incoming (epwgraph-get-incoming-links session-ports)))
      (epwgraph-disconnect-all-inputs-for-logical-node session-ports)
      (epwgraph-connect-logical-nodes
        (epwgraph--map-channels new-ports session-ports))))))

;;;###autoload
```

```

(defun sacha-speech-get-text-and-clear (session)
  (let (text)
    (with-current-buffer (cdr session)
      (setq text (buffer-substring-no-properties (point-min) (point-max)))
      (erase-buffer)
      (setq-local sacha-speech-previous-final nil))
    text))

;;;###autoload
(defun sacha-speech-insert-at-point (session)
  (interactive (list (sacha-speech-select-session)))
  (insert (sacha-speech-get-text-and-clear session)))

;;;###autoload
(defun sacha-speech-save-to-clocked-task (session)
  (interactive (list (sacha-speech-select-session)))
  (save-window-excursion
    (let ((link (org-store-link nil)))
      (org-clock-goto)
      (org-end-of-subtree)
      (unless (bolp)
        (insert "\n"))
      (insert "\n")
      (when link (insert link "\n"))
      (insert (sacha-speech-get-text-and-clear session) "\n"))))

```

Oooh, let's add it to an EmacsConf Etherpad...

```

(defvar sacha-speech-etherpads nil "Alist of (session . pad-id)")
;; (setq sacha-speech-etherpads '(("chrome-VgjMhu" . "test")))

;;;###autoload
(defun sacha-speech-append-to-etherpad (info)
  (when (and info (string= (assoc-default 'type info) "FINAL"))
    (let-alist info
      (when-let* ((pad-id (assoc-default .session sacha-speech-etherpads #'string=)))
        (emacsconf-pad-append-text pad-id (concat "\n" .content))))
    info)

;;;###autoload
(defun sacha-speech-link-etherpad (session pad-id)
  (interactive (list
    (sacha-speech-select-session)
    (read-string "Pad ID: ")))
  (add-to-list 'sacha-speech-etherpads
    (cons (concat "#" (car session))
          pad-id)))

;;;###autoload
(defun sacha-speech-unlink-etherpad (pad-id)
  (interactive (list (completing-read "Pad: " (mapcar 'cdr sacha-speech-etherpads))))
  (setq sacha-speech-etherpads
    (seq-remove (lambda (o)
      (string= (cdr o) pad-id))
      sacha-speech-etherpads)))

```

```

(with-eval-after-load 'sacha-speech
  (add-to-list 'sacha-speech-functions #'sacha-speech-append-to-etherpad))

```

And let's do the same with IRC:

```
(defvar sacha-speech-erc nil "Alist of (session . channel)")
;; (setq sacha-speech-erc '("#chrome-HP7k8I" . "#emacsconf-test"))

;;;autoload
(defun sacha-speech-send-to-erc (info)
  (when (and info (string= (assoc-default 'type info) "FINAL"))
    (let-alist info
      (when-let* ((channel (assoc-default .session sacha-speech-erc #'string=)))
        (emacsconf-erc-with-channels (list channel)
          (erc-send-message (string-trim .content))))))
  info)

;;;autoload
(defun sacha-speech-link-erc (session channel)
  (interactive (list
    (sacha-speech-select-session)
    (read-string "Channel: ")))
  (add-to-list 'sacha-speech-erc
    (cons (concat "#" (car session))
      channel)))

;;;autoload
(defun sacha-speech-unlink-channel (channel)
  (interactive (list (completing-read "Channel: " (mapcar 'cdr sacha-speech-erc))))
  (setq sacha-speech-erc
    (seq-remove (lambda (o)
      (string= (cdr o) channel))
      sacha-speech-erc)))
```

```
(with-eval-after-load 'sacha-speech
  (add-to-list 'sacha-speech-functions #'sacha-speech-send-to-erc))
```

Ooooh! Let's run it through my common edits process as well!

```
;;;autoload
(defun sacha-speech-fix-common-errors (info)
  (with-temp-buffer
    (insert (alist-get 'content info))
    (goto-char (point-min))
    (sacha-subed-fix-common-errors-from-start)
    (setf (alist-get 'content info) (buffer-string)))
  info)
```

```
(with-eval-after-load 'sacha-speech
  (add-hook 'sacha-speech-functions #'sacha-speech-fix-common-errors -100))
```

And let's try it with the markers...

```
;;;autoload
(defun sacha-speech-insert-at-markers (info)
  (when (and sacha-whisper-target-markers info)
    (sacha-whisper-insert (alist-get 'content info))))
```

```
(add-hook 'sacha-speech-functions #'sacha-speech-insert-at-markers 100)
```

speech and subed-record

```
(defvar sacha-speech-timestamp-adjust-before 1000)
(defvar sacha-speech-timestamp-adjust-after 300)

;;;###autoload
(defun sacha-speech-subed-record-convert-timestamp (s)
  "Convert S into a relative number of milliseconds based on `subed-record-filename'."
  (floor (* (float-time (time-subtract (date-to-time s) subed-record-start-time)) 1000.0)))

;;;###autoload
(defun sacha-speech-subed-record-distance (s1 s2)
  (/
    (* 1.0
      (string-distance (downcase (replace-regexp-in-string "[^A-Za-z]"
                                                             ""
                                                             s1))
                       (downcase (replace-regexp-in-string "[^A-Za-z]"
                                                             ""
                                                             s2)))))
    (max (length s1)
          (length s2))))

;;;###autoload
(defun sacha-speech-subed-record-close-enough (s1 s2)
  "Return t if it's close enough."
  (< (sacha-speech-subed-record-distance s1 s2) 0.3))

;;;###autoload
(defun sacha-speech-subed-record-update (info)
  (let ((start-ms (- (sacha-speech-subed-record-convert-timestamp
                     (alist-get 'start info))
                     sacha-speech-timestamp-adjust-before))
        (stop-ms (+ (sacha-speech-subed-record-convert-timestamp
                     (alist-get 'end info))
                     sacha-speech-timestamp-adjust-after)))
    (subed-set-subtitle-time-start start-ms)
    (subed-set-subtitle-time-stop stop-ms)
    (subed-set-subtitle-comment
     (concat
      (if (subed-subtitle-comment)
          (concat (string-trim (replace-regexp-in-string
                               ↵ ↵
                               "#\\++AUDIO: .*\\((\\n\\|\\$\\|\\)?" ""
                               (subed-subtitle-comment)))
                  "\\n")
          ""))
      (format "#+AUDIO: %s" subed-record-filename)))
    (message "%.1f %s"
              (sacha-speech-subed-record-distance
               (alist-get 'content info)
               (subed-subtitle-text))
              (alist-get 'content info))))

(defvar sacha-speech-subed-ignore nil "Ignore the GTTS-CLI output.")
;;;###autoload
```



```

(defun sacha-speech-subed-record-process (info)
  (let ((text (alist-get 'content info))
        (current (subed-subtitle-text)))
    (cond
      ((sacha-speech-subed-record-close-enough text current)
       (sacha-speech-subed-record-update info)
       (subed-forward-subtitle-text)
       (sacha-learn-lang-say-current-subtitle
        (lambda ()
          (setq sacha-speech-subed-ignore nil))))
      ;; Check previous
      ((sacha-speech-subed-record-close-enough
        text
        (save-excursion
          (subed-backward-subtitle-text)
          (subed-subtitle-text)))
       (save-excursion
        (subed-backward-subtitle-text)
        (sacha-speech-subed-record-update info)))
      ;; Check next
      ((sacha-speech-subed-record-close-enough
        text
        (save-excursion
          (subed-forward-subtitle-text)
          (subed-subtitle-text)))
       (save-excursion
        (subed-forward-subtitle-text)
        (sacha-speech-subed-record-update info)))
      (t
       (sacha-speech-subed-record-update info))))))
;; ###autoload
(defun sacha-speech-subed-record (info)
  (when (and (string= (alist-get 'type info) "FINAL")
             (derived-mode-p 'subed-mode)
             (boundp 'subed-record-start-time)
             subed-record-start-time
             (not sacha-speech-subed-ignore))
    (sacha-speech-subed-record-process info))
  info)

```

```

(with-eval-after-load 'sacha-speech
  (add-to-list 'sacha-speech-functions #'sacha-speech-subed-record))

```

7.19.11 Try speeches with realtime

```

(defvar sacha-speeches-process nil)
(defvar sacha-speeches-dir "~/vendor/speeches")
;; ###autoload
(defun sacha-speeches-start ()
  (interactive)
  (unless (process-live-p sacha-speeches-process)
    (let ((default-directory sacha-speeches-dir))
      (setq sacha-speeches-process
            (make-process
             :name "speeches-bridge"
             :buffer "*speeches-output*" ; Standard output buffer

```

```

:command '("bash" "-c" "rec -q -t raw -r 16000 -c 1 -b 16 -e signed-integer - | ↵
↵ uv run python3 stream.py")
:filter #'sacha-speaches-filter
:sentinel (lambda (proc event)
  (when (memq (process-status proc) '(exit signal))
    (message "Speaches process finished: %s" event)))
:stderr "*speaches-stderr*" ; Separate buffer for Python errors/logs
:noquery t)))
(message "Speaches started.")))

;;;###autoload
(defun sacha-speaches-filter (proc string)
  "Accumulate STRING and call processor on complete JSON lines."
  (let ((moving-point (process-mark proc))
        results)
    (with-current-buffer (process-buffer proc)
      (save-excursion
        (goto-char moving-point)
        (insert string)
        (set-marker (process-mark proc) (point))
        (goto-char (point-min))
        (while (search-forward "\n" nil t)
          (let ((line (buffer-substring (point-min) (1- (point)))))
            (delete-region (point-min) (point))
            (unless (string-empty-p (string-trim line))
              (condition-case err
                (let ((json-obj (json-parse-string line :object-type 'alist)))
                  (push json-obj results))
                (error (message "JSON parse error: %s" err)))))))
          (mapc #'sacha-speaches-process-logic
                (nreverse results))))))

;;;###autoload
(defun sacha-speaches-process-logic (o)
  "Handle the parsed ALIST from the Speaches bridge."
  (let ((type (cdr (assoc 'type o))))
    (cond
      ((string= type "conversation.item.input_audio_transcription.completed")
       (let ((text (cdr (assoc 'transcript o))))
         (with-current-buffer (get-buffer-create "*speaches*")
           (goto-char (point-max))
           (unless (bolp) (insert "\n"))
           (insert text)))
         (t (prin1 o))))))

```

7.19.12 TOBLOG Using Emacs Lisp to send audio files to Deepgram and format VTTsemas:speech

I've been experimenting with Deepgram's API for speech recognition because it can handle larger files than OpenAI Whisper's API, so I don't have to worry about chunking my files into 15-minute segments. It also supports diarization, which means identifying different speakers. That's handy for things like the EmacsConf Q&A sessions, which involve multiple people.

I think the built-in VTT formatter doesn't handle speaker identification, so I wrote some Emacs Lisp to send an audio file for recognition, save the JSON, and format the results as a VTT subtitle file. I also split the captions a little closer to the way I like to do them, starting a new subtitle if the line exceeds `sacha-deepgram-length-threshold` or `sacha-deepgram-time-threshold`, or if we're after a punctuated word and the current subtitle is more than halfway to the length threshold. Someday

I'll figure out how to get it to split on prepositions.

```
(defvar sacha-deepgram-length-threshold 45 "Number of characters.")
(defvar sacha-deepgram-time-threshold 10 "Number of seconds since the first word.")

;;;###autoload
(defun sacha-deepgram-recognize-audio (audio-file &optional diarize)
  "Send AUDIO-FILE to Deepgram, save the JSON, and create a VTT.
If DIARIZE is non-nil, identify speakers."
  (require 'subed)
  (interactive (list (if (auth-info-password (car (auth-source-search :host ↵
↵ "https://api.deepgram.com"))))

↵
↵ (read-file-name "Audio file: ")
↵
↵ (error "Please put deepgram API key in auth sources.))))
  (with-current-buffer (get-buffer-create "*Deepgram*")
    (erase-buffer)
    (unless (string-match "\\(opus\\|wav\\|mp3\\|\\)$" audio-file)
      (if (file-exists-p (concat (file-name-sans-extension audio-file) ↵
↵ ".opus"))
          (setq audio-file (concat (file-name-sans-extension ↵
↵ audio-file) ".opus"))
          (call-process "ffmpeg" nil t t "-i" (expand-file-name audio-file)
                        "-ac" ↵
↵ "1" "-y"
↵
↵ (expand-file-name (concat (file-name-sans-extension audio-file) ".opus"))
          (setq audio-file (concat (file-name-sans-extension audio-file) ↵
↵ ".opus"))))
    (unless (file-exists-p (expand-file-name (concat (file-name-sans-extension ↵
↵ audio-file) ".json")))
      (call-process
        "curl" nil t t "--request" "POST" "--header"
        (concat "Authorization: Token " (auth-info-password (car ↵
↵ (auth-source-search :host "https://api.deepgram.com"))))
        "--header" (concat "Content-Type: " (mailcap-file-name-to-mime-type ↵
↵ audio-file))
        "--data-binary" (concat "⌘" (expand-file-name audio-file))
        "--url"
        (concat
          ↵
          ↵ "https://api.deepgram.com/v1/listen?punctuate=true&model=whisper-large&smart_format=true&utterances=true"
          (if diarize
              "&diarize=true"
              ""))
          "-o"
          (expand-file-name (concat (file-name-sans-extension audio-file) ↵
↵ ".json"))))
      (sacha-deepgram-convert-json-to-vtt (concat (file-name-sans-extension ↵
↵ audio-file) ".json"))
      (find-file (concat (file-name-sans-extension audio-file) ".vtt")))

;;;###autoload
(defun sacha-emacsconf-extract-deepgram-recognize-qa-for-talk (talk)
  "Send the QA (or main) Opus file for TALK to Deepgram.
Save the results as JSON and VTT."
  (interactive (list (emacsconf-complete-talk-info)))
  (setq talk (emacsconf-resolve-talk talk))
```

```

    (if (or (emacsconf-talk-file talk "--answers--original.json")
            (emacsconf-talk-file talk "--original.json"))
        (message "Files already exist for %s" (plist-get talk :slug))
        (if-let ((file
                  (or (emacsconf-talk-file talk
                  ↪ "--answers--original.opus")
                      ↪
                  ↪ (emacsconf-talk-file talk "--original.opus"))))
            (sacha-deepgram-recognize-audio file)
            (error "No file to recognize for %s" (plist-get talk :slug)))))

;;###autoload
(defun sacha-deepgram-parse (json-file)
  "Convert JSON-FILE into a list of subtitles."
  (let* ((json-object-type 'alist)
         (json (json-read-file json-file))
         (words
          (assoc-default
           'words
           (aref (assoc-default 'alternatives (aref (let-alist ↪
           ↪ json .results.channels) 0)) 0)))
         (halfway-length (/ sacha-deepgram-length-threshold 2))
         subtitles
         current
         current-length
         last-speaker
         last-text
         current-text)
    (dolist (speaker (seq-group-by (lambda (o) (assoc-default 'speaker o)) words))
      (setf current-length 0 current nil)
      (dolist (word (cdr speaker))
        (let-alist word
          ;; determine whether we are adding to the existing one.
          ;; start a new one if length > length-threshold
          ;; or time > time-threshold
          (when (or (> (+ (length .punctuated_word)
          ↪
          ↪ current-length)
                      ↪
          ↪ sacha-deepgram-length-threshold)
                (and (car ↪
          ↪ current)
                     ↪
                     (> .start (+ (assoc-default 'start (car current))
                     ↪
                     ↪
                     ↪ sacha-deepgram-time-threshold))))
            ;; copy the previous subtitle
            (push current subtitles)
            (setf current nil current-length 0))
          (push word current)
          (setf current-length (+ (length .punctuated_word) ↪
          ↪ current-length 1))
          (when (and (string-match "[,\\.?.]" .punctuated_word)
                     (> ↪
          ↪ current-length halfway-length))
            (push current subtitles)
            (setf current nil current-length 0))))
      (when current (push current subtitles)))

```

```

        (seq-keep
          (lambda (entry)
            (setq current-text
                  (mapconcat (lambda (w) (assoc-default 'punctuated_word w))
                             (reverse entry) " "))
            (when (not (string= (downcase current-text) (or last-text "")))
              (setq last-text (downcase current-text))
              (list nil
                    (* (assoc-default 'start (car (last entry))) nil 0) 1000)
                    (* (assoc-default 'end (car entry)) nil 0) 1000)
            ;; add speaker tag?
            (concat
              (if (and (assoc-default 'speaker (car entry))
                        (or (null last-speaker)
                            (not (eq last-speaker (assoc-default 'speaker (car entry))))))
                  (progn
                    (setq last-speaker (assoc-default 'speaker (car entry)))
                    (format "[Speaker %d]: " (assoc-default 'speaker (car entry)))
                    current-text)
                ""))
            (sort subtitles
                  (lambda (a b)
                    ;; sort by time
                    (< (assoc-default 'start (car a) nil 0)
                       (assoc-default 'start (car b) nil 0)))))
    (lambda (a b)
      ;; sort by time
      (< (assoc-default 'start (car a) nil 0)
         (assoc-default 'start (car b) nil 0)))))

;; ###autoload
(defun sacha-deepgram-convert-json-to-vtt (json-file &optional force)
  (interactive (list (read-file-name "JSON: ") current-prefix-arg))
  "Convert JSON-FILE into a VTT."
  (subed-create-file
   (concat (file-name-sans-extension json-file) ".vtt")
   (sacha-deepgram-parse json-file)
   force))

(defconst deepgram-whisper-large-per-min 0.0048)
;; ###autoload
(defun sacha-deepgram-cost (file)
  (interactive "FFile: ")
  (let* ((whisper-large-per-min deepgram-whisper-large-per-min)
         (nova2-streaming-per-min 0.0059)
         (duration (/ (ceiling (/ (compile-media-get-file-duration-ms file) 1000.0)) 60))
         (msg (format "%.1f minutes: USD %.2f batch, USD %.2f streaming"
                      duration
                      (* duration whisper-large-per-min)
                      (* duration nova2-streaming-per-min))))
    msg))

```

```

↳ duration nova2-streaming-per-min))))
    (when (called-interactively-p 'any)
      (message "%S" msg)
      (kill-new msg))
    (list
      duration
      (* duration whisper-large-per-min)
      (* duration nova2-streaming-per-min))))

```

7.19.13 TOBLOG Rerecognize this audio and reprocess it

```

;;;###autoload
(defun sacha-audio-braindump-reprocess (audio-file)
  (interactive
    (list
      (let ((default (cond
        ↳
        ↳ (save-excursion
        ↳
        ↳ (org-back-to-heading)
        ↳ (when ↳
        ↳ (re-search-forward "\\[Audio\\]" nil (save-excursion (org-end-of-subtree)))
        ↳
        ↳ (org-element-property :path (org-element-context))))))
        ↳
        ↳ ((file-exists-p (concat (file-name-sans-extension (buffer-file-name)) ".m4a"))
        ↳ (concat ↳
        ↳ (file-name-sans-extension (buffer-file-name)) ".m4a")))))
        ↳ (read-file-name (if default (format "Audio (%s): " default)
        ↳
        ↳ "Audio: ")
        ↳ nil ↳
        ↳ default))))
    (save-window-excursion
      (unless (file-exists-p (concat (file-name-sans-extension audio-file) ".json"))
        (sacha-deepgram-recognize-audio audio-file))
      (with-temp-file (concat (file-name-sans-extension audio-file) ".txt")
        (insert
          (subed-subtitle-list-text
            (sacha-deepgram-parse (concat (file-name-sans-extension ↳
        ↳ audio-file) ".json"))))
        (goto-char (point-min))
        (sacha-audio-braindump-prepare-alignment-breaks))
      (with-current-buffer (find-file-noselect (concat (file-name-sans-extension ↳
        ↳ audio-file) ".txt"))
        (subed-align audio-file (concat (file-name-sans-extension audio-file) ↳
        ↳ ".txt") "VTT"))
      (find-file sacha-audio-braindump-braindump-file)
      (goto-char (point-min))
      (sacha-audio-braindump-insert-subtitles-as-org-tree (concat (file-name-sans-extension ↳
        ↳ audio-file) ".vtt"))))

```

7.19.14 Gladia

```

;;;###autoload

```



```

↪      (> .start (+ (assoc-default 'start (car current)))
↪
↪
↪      sach-deepgram-time-threshold))))
↪
↪      ;; copy the previous subtitle
↪      (push current subtitles)
↪      (setq current nil current-length 0))
↪      (push word current)
↪      (setq current-length (+ (length .punctuated_word)
↪      current-length 1))
↪      (when (and (string-match "[,\\.?.]" .punctuated_word)
↪      (>
↪      current-length halfway-length))
↪      (push current subtitles)
↪      (setq current nil current-length 0))))
↪      (when current (push current subtitles)))
↪      (seq-keep
↪      (lambda (entry)
↪      (setq current-text
↪      (mapconcat (lambda (w) (assoc-default
↪      'punctuated_word w))
↪      (nreverse entry) " "))
↪      (when (not (string= (downcase current-text) (or last-text "")))
↪      (setq last-text (downcase current-text))
↪      (list nil
↪      (* (assoc-default 'start (car entry)
↪      nil 0) 1000)
↪      (* (assoc-default 'end (car (last
↪      entry)) nil 0) 1000)
↪      ;; add speaker tag?
↪      (concat
↪      (if (and (assoc-default
↪      'speaker (car entry))
↪      (or (null last-speaker)
↪      (not (eq last-speaker (assoc-default 'speaker (car entry))))))
↪      (progn
↪      (setq
↪      last-speaker (assoc-default 'speaker (car entry)))
↪      (format "[Speaker %s]: " (assoc-default 'speaker (car entry)))
↪      ""))
↪      current-text
↪      ))))
↪      (sort subtitles
↪      (lambda (a b)
↪      ;; sort by time
↪      (< (assoc-default 'start (car a) nil 0)
↪      (assoc-default 'start (car b)
↪      nil 0))))))
↪
;; ##autoload
(defun sach-deepgram-recognize-audio (audio-file &optional diarize other-options)
  "Send AUDIO-FILE to Gladia, save the JSON, and create a VTT.
If DIARIZE is non-nil, identify speakers."
  (interactive (list (if (getenv "GLADIA_API_KEY")

```



```

↩      (read-file-name "Audio file: ")
↩
↩      (error "Please specify GLADIA_API_KEY.)))
      (with-current-buffer (get-buffer-create "*recognition*")
        (erase-buffer)
        (call-process
         "curl" nil t t "--request" "POST" "--header"
         (concat "x-gladia-key: " (getenv "GLADIA_API_KEY"))
         "--header" (concat "Content-Type: multipart/form-data" )
         "--header" (concat "Accept: application/json")
         "-F" (concat "audio=@" (expand-file-name audio-file) ";type=" ↩
        ↩      (mailcap-file-name-to-mime-type audio-file))
         "-F" (concat "toggle_noise_reduction=true&output_format=json" (or ↩
        ↩      other-options "") (if diarize "&toggle_diarization=true" ""))
         "--url" ↩
        ↩      "https://api.gladia.io/audio/text/audio-transcription?toggle_noise_reduction=true&output_format=json"
         "-o"
         (expand-file-name (concat (file-name-sans-extension audio-file) ".json"))))
        (subed-create-file
         (concat (file-name-sans-extension audio-file) ".vtt")
         (sacha-gladia-parse (concat (file-name-sans-extension audio-file) ".json"))))
        (find-file (concat (file-name-sans-extension audio-file) ".vtt")))

```

7.19.15 **DONE** Getting live speech into Emacs with Deepgram's streaming API emacs:speech

- [2023-12-26 Tue]: Reorganized code to call a list of functions and pass the recognition results. Added Etherpad. Took out the mode; will just use the functions. Related: [getting live speech from Emacs into Etherpad](#)

This is a quick demonstration of using [Deepgram's streaming API](#) to do speech recognition live. It isn't as accurate as OpenAI Whisper but since Whisper doesn't have a streaming API, it'll do for now. I can correct misrecognized words manually. I tend to talk really quickly, so it displays the words per minute in my modeline. I put the words into an Org Mode buffer so I can toggle headings with avy and cycle visibility. When I'm done, it saves the text, JSON, and WAV for further processing. I think it'll be handy to have a quick way to take live notes during interviews or when I'm thinking out loud. Could be fun!

<https://sachachua.com/blog/2023/12/live-speech-with-deepgram/2023-12-19-live-speech.webm>

I'm still getting some weirdness when the mode turns on when I don't expect it, so that's something to look into. Maybe I won't use it as a mode for now. I'll just use `sacha-live-speech-start` and `sacha-live-speech-stop`.

General code

```

(defvar sacha-live-speech-buffer "*Speech*")
(defvar sacha-live-speech-process nil)
(defvar sacha-live-speech-output-buffer "*Speech JSON*")

(defvar sacha-live-speech-functions
  '(sacha-live-speech-display-in-speech-buffer
    sacha-live-speech-display-wpm
    sacha-live-speech-append-to-etherpad)
  "Functions to call with one argument, the recognition results.")

;; ##autoload

```

```

(defun sacha-live-speech-start ()
  "Turn on live captions."
  (interactive)
  (with-current-buffer (get-buffer-create sacha-live-speech-buffer)
    (unless (process-live-p sacha-live-speech-process)
      (let ((default-directory "~/proj/deepgram-live"))
        (message "%s" default-directory)
        (with-current-buffer (get-buffer-create ↵
          ↵ sacha-live-speech-output-buffer)
          (erase-buffer))
          (setq sacha-live-speech-recent-words nil
                sacha-live-speech-wpm-string "READY ")
          (setq sacha-deepgram-process
                (make-process
                 :command '("bash" "run.sh")
                 :name "speech"
                 :filter 'sacha-live-speech-json-filter
                 :sentinel ↵
                 :buffer ↵
          ↵ #'sacha-live-speech-process-sentinel
          ↵ sacha-live-speech-output-buffer)))
      (org-mode))
      (display-buffer (current-buffer))))

;;;###autoload
(defun sacha-live-speech-stop ()
  (interactive)
  (if (process-live-p sacha-live-speech-process)
      (kill-process sacha-live-speech-process))
  (setq sacha-live-speech-wpm-string nil))

;; (define-minor-mode sacha-live-speech-mode
;;   "Show live speech and display WPM.
;;   Need to check how to reliably turn this on and off."
;;   :global t :group 'sachac
;;   (if sacha-live-speech-mode
;;       (sacha-live-speech-start)
;;       (sacha-live-speech-stop)
;;       (setq sacha-live-speech-wpm-string nil)))

;; based on subed-mpv::client-filter
;;;###autoload
(defun sacha-live-speech-handle-json (line-object)
  "Process the JSON object in LINE."
  (run-hook-with-args 'sacha-live-speech-functions (json-parse-string line :object-type ↵
    ↵ 'alist)))

;;;###autoload
(defun sacha-live-speech-process-sentinel (proc event)
  (when (string-match "finished" event)
    (sacha-live-speech-stop)
    ;(sacha-live-speech-mode -1)
  ))

;;;###autoload
(defun sacha-live-speech-json-filter (proc string)
  (when (buffer-live-p (process-buffer proc))
    (with-current-buffer (process-buffer proc)
      (let* ((proc-mark (process-mark proc))

```

```

                                (moving (= (point) proc-mark)))
;; insert the output
(save-excursion
  (goto-char proc-mark)
  (insert string)
  (set-marker proc-mark (point)))
(if moving (goto-char proc-mark))
;; process and remove all complete lines of JSON (lines are ←
↪ complete if ending with \n)
(let* ((pos (point-min))
      (while (progn (goto-char pos)
                    ↪
                    (end-of-line)
                    ↪
                    (equal (following-char) ?\n))
          (let* ((end (point))
                (line ↪
                    (buffer-substring pos end)))
              (delete-region pos (+ end 1))
              (with-current-buffer (get-buffer ↪
                    (sacha-live-speech-buffer)
                    (sacha-live-speech-handle-json ↪
                    (line)))))))))

```

Python code based on [the Deepgram streaming test suite](#):

nil

The Python script sends the microphone stream to Deepgram and prints out the JSON output. The Emacs Lisp code starts an asynchronous process and reads the JSON output, displaying the transcript and calculating the WPM based on the words. `run.sh` just loads the venv for this project (requirements.txt based on the streaming text suite) and then runs `app.py`, since some of the Python library versions conflict with other things I want to experiment with.

I also added `sacha-live-speech-wpm-string` to my `mode-line-format` manually using `Customize`, since I wanted it displayed on the left side instead of getting lost when I turn `keycast-mode` on.

I'm still a little anxious about accidentally leaving a process running, so I check with `ps aux | grep python3`. Eventually I'll figure out how to make sure everything gets properly stopped when I'm done.

Anyway, there it is!

Display in speech buffer

```

;; ##autoload
(defun sacha-live-speech-display-in-speech-buffer (recognition-results)
  (with-current-buffer (get-buffer-create sacha-live-speech-buffer)
    (let-alist recognition-results
      (let* ((pos (point))
            (at-end (eobp)))
        (goto-char (point-max))
        (unless (eolp) (insert "\n"))
        (when .msg
          (insert .msg "\n"))
        (when .transcript
          (insert .transcript "\n"))
        ;; scroll to the bottom if being displayed
        (if at-end
            (when (get-buffer-window (current-buffer))

```

```

                                (set-window-point (get-buffer-window ↵
↵ (current-buffer)) (point)))
                                (goto-char pos))))))

;;;###autoload
(defun sachalive-speech-toggle-heading ()
  "Toggle a line as a heading."
  (interactive)
  (with-current-buffer (get-buffer sachalive-speech-buffer)
    (display-buffer (current-buffer))
    (with-selected-window (get-buffer-window (get-buffer sachalive-speech-buffer))
      (let ((avy-all-windows nil))
        (avy-goto-line 1))
      (org-toggle-heading 1))))

;;;###autoload
(defun sachalive-speech-cycle-visibility ()
  "Get a quick overview."
  (interactive)
  (with-current-buffer (get-buffer sachalive-speech-buffer)
    (display-buffer (current-buffer))
    (if (eq org-cycle-global-status 'contents)
      (progn
        (run-hook-with-args 'org-cycle-pre-hook 'all)
        (org-fold-show-all '(headings blocks))
        (setq org-cycle-global-status 'all)
        (run-hook-with-args 'org-cycle-hook 'all))
      (run-hook-with-args 'org-cycle-pre-hook 'contents)
      (org-cycle-content)
      (setq org-cycle-global-status 'contents)
      (run-hook-with-args 'org-cycle-hook 'contents))))

```

Display words per minute

```

(defun sachalive-speech-wpm-window-seconds 15 "How many seconds to calculate WPM for.")
(defvar sachalive-speech-recent-words nil "Words spoken in ↵
↵ `sachalive-speech-wpm-window-minutes'.")
(defvar sachalive-speech-wpm nil "Current WPM.")
(defvar sachalive-speech-wpm-colors ; haven't figured out how to make these work yet
  '((180 :foreground "red")
    (170 :foreground "yellow")
    (160 :foreground "green")))
(defvar sachalive-speech-wpm-string nil "Add this somewhere in `mode-line-format'.")
;;;###autoload
(defun sachalive-speech-wpm-string ()
  (propertize
    (format "%d WPM " sachalive-speech-wpm)
    'face
    (cdr (seq-find (lambda (row) (> sachalive-speech-wpm (car row))) ↵
    ↵ sachalive-speech-wpm-colors))))

;;;###autoload
(defun sachalive-speech-display-wpm (recognition-results)
  (let-alist recognition-results
    (when .words
      ;; calculate WPM
      (setq sachalive-speech-recent-words
        (append sachalive-speech-recent-words .words ↵
    ↵ nil))

```

```

        (let ((threshold (- (assoc-default 'end (aref .words (1- (length ↵
↵ .words))))))
                                                    ↵
        (setf sachalivespeechwpmwindowseconds))
        (setq sachalivespeechrecentwords
              (seq-filter
               (lambda (o)
                 (≥ (assoc-default 'start o)
                     threshold)))
              sachalivespeechrecentwords))
        (setq sachalivespeechwpm
              (/
               (length sachalivespeechrecentwords)
               (/ (- (assoc-default 'end (aref ↵
↵ .words (1- (length .words))))))
                                                    ↵
        (assoc-default 'start (car sachalivespeechrecentwords)))
        60.0)))
        (setq sachalivespeechwpmstring ↵
        (sachalivespeechwpmstring))))))

```

Append to EmacsConf Etherpad

```

(defvar sachalivespeechetherpadid nil)
;;;###autoload
(defun sachalivespeechappend-to-etherpad (recognition-results)
  (when sachalivespeechetherpadid
    (emacsconf-pad-append-text sachalivespeechetherpadid (concat " " ↵
    (assoc-default 'transcript recognition-results))))

```

7.20 UTF-8

From <http://www.wisdomandwonder.com/wordpress/wp-content/uploads/2014/03/C3F.html>

```

(prefer-coding-system 'utf-8)
(when (display-graphic-p)
  (setq x-select-request-type '(UTF8_STRING COMPOUND_TEXT TEXT STRING)))

```

7.21 Wdiff

[2024-10-20 Sun]: I switched to using proper font-locking, yay.

This uses the [wdiff](#) tool for word-based diffs.

```

(defvar sachawdiff-mode-font-lock-keywords
  `(("{\\+\\(\\.\\*?\\)\\+}" . 'diff-added)
    ("\\[\\-\\(\\.\\*?\\)\\-\\]" . 'diff-removed)))

(defconst sachawdiff-mode-font-lock-defaults
  '(sachawdiff-mode-font-lock-keywords t nil nil nil (font-lock-multiline . t)))

;;;###autoload
(define-derived-mode sachawdiff-mode fundamental-mode "Word diff" "Highlight word diffs."
  (setf-local font-lock-defaults sachawdiff-mode-font-lock-defaults))

```

```

;;;###autoload
(defun sacha-wdiff (old-file new-file)
  (interactive (list (read-file-name "Original: ")
                     ↵
                     ↵ (buffer-file-name)))
  (with-current-buffer (get-buffer-create "*wdiff*")
    (erase-buffer)
    (call-process "wdiff" nil t t (expand-file-name old-file)
                  (expand-file-name ↵
                  ↵ new-file))
    (goto-char (point-min))
    (sacha-wdiff-mode)
    (switch-to-buffer (current-buffer))))

;;;###autoload
(defun sacha-wdiff-strings (original new)
  (let ((original-file (make-temp-file "wdiff"))
        (new-file (make-temp-file "wdiff")))
    (write-region original nil original-file)
    (write-region new nil new-file)
    (sacha-wdiff original-file new-file)
    (delete-file original-file)
    (delete-file new-file)))

;;;###autoload
(defun sacha-wdiff-org-text-with-clipboard ()
  (interactive)
  (sacha-wdiff-strings (sacha-org-subtree-text-without-blocks)
                       (car kill-ring)))

;;;###autoload
(defun sacha-wdiff-buffer-with-file ()
  (interactive)
  (let ((s (buffer-string))
        (temp-file (make-temp-file "temp")))
    (with-temp-file temp-file
      (insert s))
    (sacha-wdiff (buffer-file-name) temp-file)
    (delete-file temp-file)))

;;;###autoload
(defun sacha-wdiff-find-at-point ()
  (interactive)
  (unless (looking-at "\\[-")
    (re-search-backward "\\[-" nil t)
    (when (looking-at "\\[-\\(\\.+?\\)-\\] {\\+\\(\\.+?\\)\\+}")
      (let ((s (match-string 1))
            (rep (match-string 2)))
        (goto-char (match-end 0))
        (other-window 1)
        (if (re-search-forward (regexp-quote s) nil t)
            (progn
              (save-match-data (pulse-momentary-highlight-region (match-beginning 0)
                                                                    (match-end 0)))
              (when (save-match-data (y-or-n-p (format "Change %s to %s: " s rep)))
                (replace-match rep t t)
                t))
            (message "Could not find %s to change to %s" s rep))
        (message "Could not find %s to change to %s" s rep))
    (message "Could not find %s to change to %s" s rep))

```

```

        nil))))))

;;;###autoload
(defun sacha-wdiff-next ()
  (interactive)
  (other-window 1)
  (re-search-forward "{\\+\\(\\.+?\\)\\+}")
  (pulse-momentary-highlight-region (match-beginning 0) (match-end 0))
  (sacha-wdiff-find-at-point))

;;;###autoload
(defun sacha-wdiff-next-loop ()
  (interactive)
  (while (sacha-wdiff-next)))

```

7.22 Denote

```

(use-package denote
  :config
  (setopt denote-directory "~/sync/Notes")
)

```

8 Org Mode

org

I use [Org Mode](#) to take notes, publish my blog, and do all sorts of stuff.

```

(defvar sacha-org-inbox-file "~/sync/orgzly/Inbox.org")
(use-package org
  :load-path ("~/vendor/org-mode/lisp" "~/vendor/org-mode/contrib/lisp")
  :preface (load "~/vendor/org-mode/lisp/org-loaddefs.el" nil t)
  :bind
  (:map org-mode-map
    ("C-M-<return>" . org-insert-subheading)
    ("M-." . sacha-org-defun-open))
  :custom
  (org-export-with-sub-superscripts nil)
  (org-footnote-section nil)
  (org-fold-catch-invisible-edits 'smart))

```

8.1 Time zones

```

(defvar sacha-time-zones nil "List of time-zones (ex: America/Toronto) to translate time to.")
;;;###autoload
(defun sacha-summarize-times (time &optional time-zones)
  "Return TIME in TIME-ZONES."
  (let (prev-day)
    (mapconcat
      (lambda (tz)
        (let ((cur-day (format-time-string "%a %b %-e" time tz))
              (cur-time (format-time-string "%H%M %Z" time tz)))
          (if (equal prev-day cur-day)
              cur-time
              (setq prev-day cur-day))))
      time-zones)))

```

```

        (concat cur-day " " cur-time))))
    (or time-zones sachatime-zones)
    " / "))

;;;###autoload
(defun sachatime-org-timestamp-in-time-zones (&optional s)
  "Copy the current timestamp in `sachatime-zones'."
  (interactive)
  (let* ((s (or s (org-element-property :raw-value (org-element-context))))
         (time (org-timestamp-to-time (org-timestamp-from-string s)))
         (result (format "%s (%s) = %s"
                        ↪
                        s
                        ↪
                        (format-time-string "%z")
                        ↪
                        (sachatime-summarize-times time))))
    (when (called-interactively-p)
      (message "%s" result)
      (kill-new result))
    result))

```

```

(setq sachatime-timezones '("US/Eastern" "US/Central" "US/Mountain" "US/Pacific" "UTC" ↪
  ↪ "Europe/Paris" "Europe/Athens" "Asia/Kolkata" "Asia/Singapore" "Asia/Tokyo"))

```

8.2 Set file property

```

;;;###autoload
(defun sachatime-org-set-file-property (property value)
  "Set PROPERTY to VALUE using #+keywords."
  (save-restriction
    (save-excursion
      (widen)
      (goto-char (point-min))
      (if (re-search-forward (concat (regexp-quote (concat "#+" property)) ↪
  ↪ ": \\(\\.+\\)") nil t)
          (replace-match value t t nil 1)
          (goto-char (point-min))
          (while (looking-at "^#")
            (forward-line 1))
          (insert "#+" property ": " value "\n")))))

(ert-deftest sachatime-org-set-file-property ()
  ;; Add new
  (should
    (equal
      (with-temp-buffer
        (org-mode)
        (insert "* This is a heading")
        (sachatime-org-set-file-property "MODIFIED" "2026-01-01")
        (buffer-substring-no-properties (point-min) (point-max)))
      "#+MODIFIED: 2026-01-01\n* This is a heading"))
  (should
    ;; Replace existing
    (equal
      (with-temp-buffer

```



```

(org-mode)
(insert "* This is a heading\n\n#+MODIFIED: 2025-01-01\n")
(sacha-org-set-file-property "MODIFIED" "2026-01-01")
(buffer-substring-no-properties (point-min) (point-max)))
"* This is a heading\n\n#+MODIFIED: 2026-01-01\n")
;; Add after
(should
 (equal
  (with-temp-buffer
   (org-mode)
   (insert "#+OTHER: 1\n* This is a heading\n")
   (sacha-org-set-file-property "MODIFIED" "2026-01-01")
   (buffer-substring-no-properties (point-min) (point-max)))
  "#+OTHER: 1\n#+MODIFIED: 2026-01-01\n* This is a heading\n")))

```

8.3 Automatically continue lists

[2026-04-05 Sun] Thanks to Sawomir G. for the tip! I want to tweak it slightly so that I can still add newlines to a list item if I use `C-u M-RET`.

```

;;;###autoload
(defun sacha-org-autolist-allow-newlines ()
  "Insert newline conditionally."
  (when (and current-prefix-arg (org-in-item-p) org-autolist-mode)
    (insert "\n")
    t))

```

```

(use-package org-autolist
  :hook
  ((org-mode . org-autolist-mode)
   (org-metareturn . sacha-org-autolist-allow-newlines)))

```

8.4 PDFs

```

(use-package org-pdftools
  :hook (org-mode . org-pdftools-setup-link))

```

8.5 After I jump to a task from org-clock-goto, narrow to it automatically

```

(with-eval-after-load 'org
  (add-hook 'org-clock-goto-hook #'org-narrow-to-subtree)
  (add-hook 'org-agenda-after-show-hook #'org-narrow-to-subtree))

```

Send the recognized text to a list of functions

8.6 Find first common Org Mode heading

```

;;;###autoload
(defun sacha-org-find-first-common-heading (other-buffer)
  "Go to the first top-level heading in common with OTHER-BUFFER.
This is helpful when resolving sync conflicts."

```

```

(interactive (list (read-buffer "Other buffer: ")))
(let ((other-headings (with-current-buffer (get-buffer other-buffer)
  ↪
  (org-map-entries (lambda () (org-entry-get (point) "ITEM")) "LEVEL=1"))))
  (goto-char
    (catch 'done
      (org-map-entries
        (lambda ()
          (when (member (org-entry-get (point) "ITEM") ↪
            ↪ other-headings)
            (throw 'done (point))))
        "LEVEL=1"))))
  ))

```

8.7 Writing about sketches and including their text

I use [Google Cloud Vision](#) to convert my handwriting to text. This is handled by my [sacha-image-recognize](#) (p. 32) function. My [sacha-sketch-process](#) (p. 454) function extracts the text, renames the file, recolors the image, and opens the associated text file for further editing. After I correct the text, I use these functions to start writing about the sketch. They insert the image and the text for me.

```

;;;###autoload
(defun sacha-insert-sketch-and-text (sketch)
  (interactive (list (sacha-complete-sketch-filename)))
  (when (and (listp sketch) (alist-get 'source_path sketch))
    (setq sketch (sacha-get-image-filename (file-name-base (alist-get 'source_path sketch)))))
    (insert
      (if (string= (file-name-extension sketch) "svg")
        (format
          "#+begin_panzoom\n%s\n#+end_panzoom\n\n"
          (org-link-make-string (concat "file:" sketch)))
        (concat (org-link-make-string (concat "sketchFull:" (file-name-base sketch)) ↪
          ↪ "\n\n"))
        (let ((links (sacha-org-links-from-file (concat (file-name-sans-extension sketch) ↪
          ↪ ".txt"))))
          (subheading-level (1+ (org-current-level))))
          (insert (if links
            ↪ "#+begin_my_details Text and ↪
            ↪ links from sketch\n"
            ↪ "#+begin_my_details Text from sketch\n"))
          (sacha-sketch-insert-text sketch)
          (unless (bolp) (insert "\n"))
          (insert "#+end_my_details")
          (dolist (section (seq-filter (lambda (entry) (string-match "^#" (cdr entry)))
            ↪
            ↪ links))
            (org-end-of-subtree)
            (insert "\n\n")
            (org-insert-heading nil nil subheading-level)
            (insert (car section))
            (org-entry-put (point) "CUSTOM_ID" (substring (cdr section) 1))))))

;;;###autoload
(defun sacha-write-about-sketch (sketch)
  (interactive (list (sacha-complete-sketch-filename)))
  ↪ (shell-command "make-sketch-thumbnails")

```

```
(find-file "~/sync/orgzly/posts.org")
(goto-char (point-min))
  (unless (org-at-heading-p) (outline-next-heading))
(org-insert-heading nil nil t)
  (insert (string-trim (replace-regexp-in-string "^[-0-9]+ *" "" (file-name-base ↵
↵ sketch)))) "\n\n")
  (sacha-insert-sketch-and-text sketch)
  (insert "\n/Feel free to use this sketch under the ↵
↵ [[https://creativecommons.org/licenses/by/4.0/][Creative Commons Attribution ↵
↵ license]]./\n")
(delete-other-windows)
(save-excursion
  (with-selected-window (split-window-horizontally)
    (find-file sketch))))
```

8.8 Remove open Org Mode clock entries

Pedro pointed out that I had some incomplete clock entries in my Emacs configuration. [org-resolve-clocks](#) prompts you for what to do with each open clock entry in your Org agenda files and whatever Org Mode files you have open.

If you don't feel like cancelling each clock with `C`, I also wrote this function to delete all open clocks in the current file.

```
;; ##autoload
(defun sacha-org-delete-open-clocks ()
  (interactive)
  (flush-lines
   (rx
    line-start
    (zero-or-more space)
    "CLOCK:"
    (one-or-more space)
    (regexp org-ts-regexp-inactive)
    (zero-or-more space)
    line-end))))
```

8.9 Insert a link to an Org Mode heading from an org-refile prompt `embark:org`

I often want to link to an Org heading from somewhere in my `org-refile-targets`, which includes my agenda files and other things. I don't want to think about where the heading is, I just want to link to it. I could use `C-u C-c C-w` (`org-refile`) to go to the heading, use `C-c l` (my shortcut for `org-store-link`), head back with `org-mark-ring-goto`, and then insert it with `C-c C-l` (`org-insert-link`).

Or I can use this code to make an Embark command that hooks into minibuffer prompts that include "goto" or "refile", so I can link to something with `C-. i` right from a refile or goto query.

```
;; ##autoload
(defun sacha-embark-org-insert-link-from-path (path)
  (interactive (list (car (org-refile-get-location))))
  (let* ((extra (if org-refile-use-outline-path "/" ""))
         (tbl (mapcar
                  (lambda (x)
                    (if (and (not (member ↵
↵ org-refile-use-outline-path
```

```
(with-eval-after-load 'marginalia
  (add-to-list 'marginalia-prompt-categories ("Goto\\|Refile" . sachal-org-path)))
(with-eval-after-load 'embark
  (add-to-list 'embark-keymap-alist '(sachal-org-path . sachal-org-path-map)))
```

```
(with-eval-after-load 'consult-org
  (keymap-set embark-org-heading-map "L" #'embark-org-insert-link-to))
```

8.10 Org - mapping blog posts and image URLs from bulk exports

```
(defun sach-a-org-map-blog-and-image-urls ()
  "Extract and map blog post / image URLs."
  (interactive)
  (goto-char (point-min))
  (keep-lines "h2\\|img")
  (goto-char (point-min))
  (while (re-search-forward
         "^.*?h2.*?a href=\"\\((.*?\\)\\\".*$" nil t)
    (replace-match "\\1"))
  (goto-char (point-min))
  (while (re-search-forward
         "^.*?src=\"\\((.*?\\)\\\".*$" nil t)
    (replace-match "\\1"))
  (let (last-post current-url result)
```

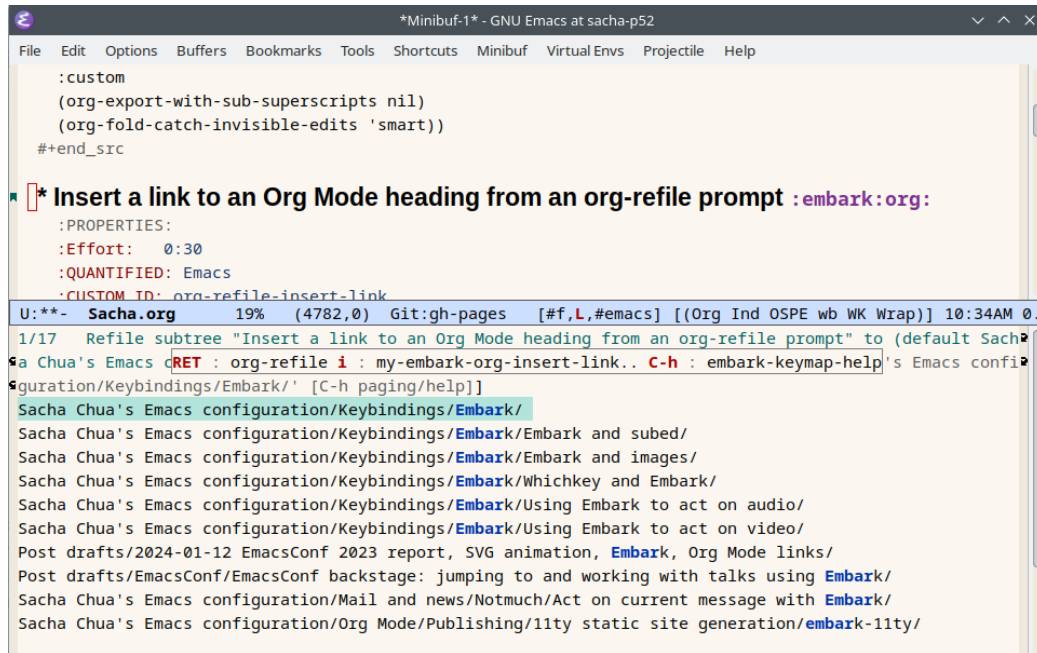


Figure 10: Screenshot of Embark menu invoked with "C-.", showing the new "i" shortcut for inserting a link

```
(goto-char (point-min))
(while (re-search-forward "http://\\(.*\\)" nil t)
  (setq current-url (match-string 0))
  (if (string-match "\\([^\n/]*\\)\\(_thumb\\|640x.*\\)?.png" current-url)
      (setq result (cons (concat (match-string 1 current-url) "\n" last-post) result))
      (setq last-post current-url)))
(kill-new (mapconcat 'identity result "\n")))
```

8.11 Move Org Mode properties from subtree to parent

Sometimes I set Org properties on a subtree when I meant to set them on the parent heading. This function moves all the properties up one level.

```
;; ##autoload
(defun sacha-org-move-properties-to-parent ()
  "Move entry properties to parent."
  (interactive)
  (let ((properties (org-entry-properties (point) 'standard)))
    ;; delete properties from the current entry
    (mapc (lambda (prop)
              (unless (string= (car prop) "CATEGORY")
                (org-entry-delete (point) (car prop))))
          properties)
    ;; add properties
    (outline-up-heading 1)
    (mapc (lambda (prop)
              (org-entry-put (point) (car prop) (cdr prop)))
          properties)))
```

8.12 My files

#

Here are the Org files I use. I should probably organize them better. =)

organizer.org	My main Org file. Inbox for M-x org-capture, tasks, weekly reviews, etc.
news.org	inbox for Emacs News
business.org	Business-related notes and TODOs
people.org	People-related tasks
evil-plans/index.org	High-level goals
sharing/index.org	Things to write about
decisions.org	Pending, current, and reviewed decisions
blog.org	Topic index for my blog
learning.org	Learning plan
outline.org	Huge outline of notes by category
tracking.org	Temporary Org file for tracking various things
delegation.org	Templates for assigning tasks - now using Google Docs instead
books.org	Huge file with book notes
calendar.org	Now using this with org-gcal
ideal.org	Planning ideal days
archive.org	Archived subtrees
latin.org	Latin notes
101things.org	Old goals for 101 things in 1001 days
life.org	Questions, processes, tools
sewing.org	Sewing projects, fabric tracking, etc.

- [emacs auto save for org-mode only - Stack Overflow](#)

8.13 Modules

Org has a whole bunch of optional modules. These are the ones I'm currently experimenting with.

```
(setq org-modules '(org-habit
                    org-mouse
                    org-protocol
                    org-annotate-file
                    ol-info
                    org-expiry
                    org-interactive-query
                    org-collector
                    org-panel
                    org-screen
                    org-toc))
(eval-after-load 'org
  '(org-load-modules-maybe t))
;; Prepare stuff for org-export-backends
(setq org-export-backends '(org latex icalendar html ascii))
```

8.14 Keyboard shortcuts

```
(keymap-global-set "C-c r" 'org-capture)
(keymap-global-set "C-c a" 'org-agenda)
(keymap-global-set "C-c l" 'org-store-link)
(keymap-global-set "C-c L" 'org-insert-link-global)
(keymap-global-set "C-c O" 'org-open-at-point-global)
```

`append-next-kill` is more useful to me than `org-table-copy-region`.

```
(with-eval-after-load 'org
  (keymap-set org-mode-map "C-M-w" 'append-next-kill)
  (keymap-set org-mode-map "C-TAB" 'org-cycle)
  (keymap-set org-mode-map "C-c C-r" 'org-refile)
  (keymap-set org-mode-map "C-c R" 'org-reveal)
  (keymap-set org-mode-map "C-c d" 'sacha-org-move-line-to-destination)
  (keymap-set org-mode-map "C-c t s" 'sacha-split-sentence-and-capitalize)
  (keymap-set org-mode-map "C-c t -" 'sacha-split-sentence-delete-word-and-capitalize)
  (keymap-set org-mode-map "C-c t d" 'sacha-delete-word-and-capitalize)

  (keymap-set org-mode-map "C-c C-p C-p" 'sacha-org-publish-maybe)
  (keymap-set org-mode-map "C-c C-r" 'sacha-org-refile-and-jump))
```

I don't use the diary, but I do use the clock a lot.

```
(with-eval-after-load 'org-agenda
  (keymap-set org-agenda-mode-map "i" 'org-agenda-clock-in))
```

8.14.1 Speed commands

Org Mode: Cutting the current list item (including nested lists) with a speed command

[2025-09-17 Wed]: Limited `sacha-org-cut-subtree-or-list-item` to Inbox.org, posts.org, news.org.

Defining shortcuts in `org-speed-commands` is handy because you can use these single-key shortcuts at the beginning of a subtree. With a little modification, they'll also work at the beginning of list items.

```
;; ##autoload
(defun sacha-org-use-speed-commands-for-headings-and-lists ()
  "Activate speed commands on list items too."
  (or (and (looking-at org-outline-regexp) (looking-back "^\\*" nil))
      (save-excursion (and (looking-at (org-item-re)) (looking-back "^[ \\t]*" nil)))))
```

```
(setq org-use-speed-commands 'sacha-org-use-speed-commands-for-headings-and-lists)
```

I want `k` to be an `org-speed-commands` that cuts the current subtree or list item. This is handy when I'm cleaning up the Mastodon toots in my weekly review or getting rid of outline items that I no longer need. By default, `k` is mapped to `org-cut-subtree`, but it's easy to override.

```
(defvar sacha-org-cut-subtree-file-regexp
  (regexp-opt '("Inbox.org" "posts.org" "news.org" "ipad.org")))
  "Only cut subtrees in files that match this regular expression.
See `sacha-org-cut-subtree-or-list-item'."

;; ##autoload
(defun sacha-org-cut-subtree-or-list-item (&optional n)
  "Cut current subtree or list item."
  (cond
    ; limit this to certain files
```

```

((and sacha-org-cut-subtree-file-regexp
(not
(string-match sacha-org-cut-subtree-file-regexp
(or ↵
↵ (buffer-file-name) ""))))
(message "Ignoring files not matching `sacha-org-cut-subtree-file-regexp'")) ; ↵
↵ do nothing
((and (looking-at org-outline-regexp) (looking-back "^\\*" nil))
(org-cut-subtree n))
((looking-at (org-item-re))
(kill-region (org-beginning-of-item) (org-end-of-item))))))

```

```

(with-eval-after-load 'org
  (setf (alist-get "k" org-speed-commands nil nil #'string=)
        #'sacha-org-cut-subtree-or-list-item))

```

So now, if I put my cursor before "1." below and press `k`:

```

- this
  1. is a
    - nested
  2. list
- with levels

```

it will turn into:

- this
 - 1. list
- with levels

You can find out a little more about [Org Mode speed commands](#) in the Org manual: `(info "(org) Speed Keys")`.

8.14.2 Other speed commands

```

(setq org-use-effective-time t)

```

```

;;;###autoload
(defun sacha-org-goto-text-start ()
  "Go to the start of the text after properties and drawers."
  (if (org-before-first-heading-p)
      (goto-char (point-min))
      (org-back-to-heading)
      (org-end-of-meta-data t)))

;;;###autoload
(defun sacha-org-subtree-text ()
  (if (derived-mode-p 'org-mode)
      (if (org-before-first-heading-p)
          (buffer-substring (point-min)
                           (save-excursion
                             (org-next-visible-heading)
                             (line-beginning-position)))
          (save-excursion
            (buffer-substring (save-excursion (org-end-of-meta-data t) (point))
                             (line-beginning-position)))
      (buffer-substring (point-min) (line-beginning-position)))

```



```

                                (org-end-of-subtree)))
  (buffer-string)))

```

```

;;;###autoload
(defun sach-org-copy-subtree-text ()
  (interactive)
  (kill-new (sach-org-subtree-text)))

;;;###autoload
(defun sach-org-mark-done ()
  (interactive)
  (sach-org-with-current-task (org-todo "DONE")))

```

```

;;;###autoload
(defun sach-org-mark-done-and-add-to-journal (&optional note category)
  (interactive (list (if current-prefix-arg
                        (read-string (format "Note (%s): " (org-get-heading t t t)))
                        (org-get-heading t t t))
                    (or (org-entry-get (point) "JOURNAL_CAT") (sach-journal-read-category ←
→ (sach-journal-guess-category))))))
  (sach-org-with-current-task
   (org-todo "DONE")
   (org-entry-put (point) "JOURNAL_CAT" category)
   (let* ((title (or note (org-get-heading t t t)))
          (zid (org-entry-get (point) "ZIDSTRING"))
          (other (if current-prefix-arg (substring-no-properties (sach-org-subtree-text)))
                (date (unless zid
                          (format-time-string "%Y-%m-%d %H:%M"
                                                (let ((base-date (org-read-date nil t (org-entry-get ←
→ (point) "CREATED"))))
                              (if (string-match "Yesterday " title)
                                  (progn
                                   (setq title (replace-match "" nil nil title))
                                   (org-read-date nil t "--1" nil ←
→ (org-time-string-to-time (org-entry-get (point) "CREATED"))))
                              base-date))))))
          (if zid
              (sach-journal-update (list :ZIDString zid :Note title :Category category :Other other))
              (org-entry-put (point) "ZIDSTRING"
                            (plist-get
                             (sach-journal-post title
                                                  :Category category
                                                  :Other other
                                                  :Date date)
                             :ZIDString)))
          (org-back-to-heading)
          (sach-copy-observation))))

```

```

(with-eval-after-load 'org
  (let ((listvar (if (boundp 'org-speed-commands) 'org-speed-commands
                    'org-speed-commands-user)))
    (add-to-list listvar '("A" org-archive-subtree-default))
    (add-to-list listvar '("x" org-todo "DONE"))
    (add-to-list listvar '("X" call-interactively 'sach-org-mark-done-and-add-to-journal))
    (add-to-list listvar '("y" org-todo-yesterday "DONE"))
    (add-to-list listvar '("! " sach-org-clock-in-and-track))
    (add-to-list listvar '("s" call-interactively 'org-schedule))

```

```
(add-to-list listvar '("d" sach-org-move-line-to-destination))
(add-to-list listvar '("i" call-interactively 'org-clock-in))
(add-to-list listvar '("o" call-interactively 'org-clock-out))
(add-to-list listvar '("$" call-interactively 'org-archive-subtree)))
(keymap-set org-agenda-mode-map "!" 'sacha-org-clock-in-and-track))
```

8.15 Org navigation

From <http://stackoverflow.com/questions/15011703/is-there-an-emacs-org-mode-command-to-jump-to-an-org-heading>

```
(setq org-goto-interface 'outline-path-completion
      org-goto-max-level 10)
(require 'imenu)
(setq org-startup-folded nil)
(setq org-startup-with-inline-images nil)
(setq org-startup-with-link-previews nil)
(keymap-global-set "C-c j" 'org-clock-goto) ;; jump to current task from anywhere
(keymap-global-set "C-c C-w" 'org-refile)
(setq org-cycle-include-plain-lists 'integrate)
(setq org-catch-invisible-edits 'show-and-error)
```

8.15.1 Link Org subtrees and navigate between them

The following code makes it easier for me to link trees with entries, as in <http://sachachua.com/evil-plans>

```
;; ###autoload
(defun sach-org-follow-entry-link ()
  "Follow the defined link for this entry."
  (interactive)
  (if (org-entry-get (point) "LINK")
      (org-open-link-from-string (org-entry-get (point) "LINK"))
      (org-open-at-point)))

;; ###autoload
(defun sach-org-link-projects (location)
  "Add link properties between the current subtree and the one specified by LOCATION."
  (interactive)
  (list (let ((org-refile-use-cache nil))
          (org-refile-get-location "Location"))))
  (let ((link1 (org-store-link nil)) link2)
    (save-window-excursion
      (org-refile 4 nil location)
      (setq link2 (org-store-link nil))
      (org-set-property "LINK" link1)
      (org-set-property "LINK" link2))))
```

8.15.2 Viewing, navigating, and editing the Org tree

I often cut and paste subtrees. This makes it easier to cut something and paste it elsewhere in the hierarchy.

```
(with-eval-after-load 'org
```

```
(keymap-set org-mode-map "C-c k" 'org-cut-subtree)
(setq org-yank-adjusted-subtrees t))
```

8.15.3 Finding my place on a small mobile screen with org-back-to-heading

As it turns out, I can use `outline-previous-heading` to go back.

```
(use-package org
  :bind (:map org-mode-map
    ("C-c b" . outline-previous-heading)
    ("C-c p" . org-display-outline-path)))
```

8.15.4 Dealing with big tables

Sometimes I forget where I am in a big table. This would be nice to turn into a minor mode someday.

```
;; ##autoload
(defun sacha-org-show-row-and-column (point)
  (interactive "d")
  (save-excursion
    (goto-char point)
    (let ((row (s-trim (org-table-get nil 1)))
          (col (s-trim (org-table-get 1 nil)))
          (message-log-max nil))
      (message "%s - %s" row col))))
```

8.16 Taking notes

```
(setq org-directory "~/sync/orgzly/")
(setq org-default-notes-file "~/sync/orgzly/organizer.org")
```

8.16.1 Date trees

This quickly adds a same-level heading for the succeeding day.

```
;; ##autoload
(defun sacha-org-insert-heading-for-next-day ()
  "Insert a same-level heading for the following day."
  (interactive)
  (let ((new-date
        (seconds-to-time
         (+ 86400.0
          (float-time
           (org-read-date nil 'to-time (elt (org-heading-components) 4))))))
        (org-insert-heading-after-current)
        (insert (format-time-string "%Y-%m-%d\n\n" new-date))))
```

8.16.2 Templates

I use `org-capture` templates to quickly jot down tasks, ledger entries, notes, and other semi-structured pieces of information.

```
;; ##autoload
(defun sach-org-contacts-template-email (&optional return-value)
  "Try to return the contact email for a template.
   If not found return RETURN-VALUE or something that would ask the user."
  (eval-when-compile (require 'gnus-art nil t))
  (eval-when-compile (require 'org-contacts nil t))
  (or (cadr (if (gnus-alive-p)
                (gnus-with-article-headers
                 (mail-extract-address-components
                  (or (mail-fetch-field "Reply-To") (mail-fetch-field "From") ""))))))
      return-value
      (concat "%{" org-contacts-email-property "}"p)))
```

```
(defvar sach-org-basic-task-template "* TODO %{Task}
:PROPERTIES:
:Effort: %{effort|1:00|0:05|0:15|0:30|2:00|4:00}
:END:
Captured %<%Y-%m-%d %H:%M>
%?

%i
" "Basic task data")
(defvar sach-ledger-file "~/cloud/ledger/current.ledger")
(with-eval-after-load 'org-capture
  (setq org-capture-templates
        (seq-uniq
         (append

          `(("r" "Note" entry
            (file ,sach-org-inbox-file)
            "* %?\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n%i\n\n- %a"
            :prepend t)

            ("t" "Task with annotation" entry
            (file ,sach-org-inbox-file)
            "* TODO %?\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n%a\n"
            :prepend t)

            ("i" "Interrupting task" entry
            (file ,sach-org-inbox-file)
            "* STARTED %{Task}\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n%a\n"
            :clock-in :clock-resume
            :prepend t)

            ("T" "Task without annotation" entry
            (file ,sach-org-inbox-file)
            "* TODO %{Task}\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n"
            :prepend t)

          ;; From https://takeonrules.com/2022/10/16/adding-another-function-to-sacha-workflow/
          ("c" "Contents to current clocked task"
           plain (clock)
           "%i?\n%a"
           :empty-lines 1)

          ;; ("p" "Podcast log - timestamped" item
          ;; (file+olp+datetree "~/sync/orgzly/timestamped.org")
          ;; "%<%H:%M:%S,%3N> %{Note}"
          ;; :immediate-finish t)

          ;; ("b" "Plover note" table-line
          ;; (file+headline "~/proj/plover-notes/README.org" "Brief notes")
          ;; "| %{Stroke} | %{Translation} | %{Note} |"
```

```

;; :immediate-finish t)
;; ("c" "Plover review from clippy" table-line
;; (file+headline "~/proj/plover-notes/README.org" "For review")
;; "%(let ((last (sacha-clippy-last))) (format \"%| %s | %s |\" (car last) (cdr last)))"
;; :immediate-finish t)

("." "Today" entry
 (file ,sacha-org-inbox-file)
 "* TODO %^{Task}\\nSCHEDULED: %t\\n:PROPERTIES:\\n:CREATED: %U\\n:END:\\n"
 :immediate-finish t)
("v" "Video" entry
 (file ,sacha-org-inbox-file)
 "* TODO %^{Task} :video:\\nSCHEDULED: %t\\n"
 :immediate-finish t)
("e" "Errand" entry
 (file ,sacha-org-inbox-file)
 "* TODO %^{Task} :errands:\\n:PROPERTIES:\\n:CREATED: %U\\n:END:\\n"
 :immediate-finish t)
("n" "Note" entry
 (file ,sacha-org-inbox-file)
 "* %^{Note}\\n:PROPERTIES:\\n:CREATED: %U\\n:END:\\n"
 :immediate-finish t)
("N" "Note" entry
 (file ,sacha-org-inbox-file)
 "* %^{Note}\\n:PROPERTIES:\\n:CREATED: %U\\n:END:\\n"
 :prepend t)
      ("s" "Selection from browser" entry
       (file ,sacha-org-inbox-file)
       "* %a :website:\\n:PROPERTIES:\\n:CREATED: ↩
↪ %U\\n:END:\\n#+begin_quote\\n%i\\n#+end_quote\\n\\n%?\\n"
       :prepend t)
      ("S" "Screenshot" entry
       (file ,sacha-org-inbox-file)
       "* %i\\n:PROPERTIES:\\n:CREATED: %U\\n:END:\\n\\n#+CAPTION: ↩
↪ %(file-name-nondirectory ↩
↪ (sacha-latest-screenshot))\\n[[file:%(sacha-latest-screenshot)]]\\n"
       :prepend t)
("b" "Business task" entry
 (file+headline "~/personal/business.org" "Tasks")
 ,sacha-org-basic-task-template)
("j" "Journal entry" plain
 (file+olp+datetree "~/sync/orgzly/journal.org")
 "%K - %a\\n%i\\n%?\\n"
 :unnarrowed t)
("db" "Done - Business" entry
 (file+headline "~/personal/business.org" "Tasks")
 "* DONE %^{Task}\\nSCHEDULED: %^t\\n%?")
("dp" "Done - People" entry
 (file+headline "~/personal/people.org" "Tasks")
 "* DONE %^{Task}\\nSCHEDULED: %^t\\n%?")
("dt" "Done - Task" entry
 (file+headline "~/sync/orgzly/organizer.org" "Inbox")
 "* DONE %^{Task}\\nSCHEDULED: %^t\\n%?")
("q" "Quick note" item
 (file+headline "~/sync/orgzly/organizer.org" "Quick notes"))
("L" "Ledger")
("Lc" "Cash expense" plain
 (file ,sacha-ledger-file)
 "%(ledger-read-date \"Date: \") * %^{Payee}

```

```

        Expenses:Cash
        Expenses:%^{Account}  %^{Amount}
    ")
    ("Lb" "BDO CAD" plain
    (file ,sacha-ledger-file)
    "%(ledger-read-date \"Date: \") * %^{Payee}
        Expenses:Play    $ %^{Amount}
        Assets:BDO
    ")
    ("Lp" "BDO PHP" plain
    (file ,sacha-ledger-file)
    "%(ledger-read-date \"Date: \") * %^{Payee}
        Expenses:Play    PHP %^{Amount}
        Assets:BDO
    ")
    ("B" "Book" entry
    (file+olp+datetree "~/personal/books.org" "Inbox")
    "* %^{Title}  %^g
        %i
        *Author(s):* %^{Author} \\\
        *ISBN:* %^{ISBN}

        %?

        *Review on:* %^t \
        %a
        %U"
    :clock-in :clock-resume)
    ("C" "Contact" entry (file "~/sync/orgzly/people.org")
    "* %(org-contacts-template-name)

:PROPERTIES:
:EMAIL: %(sacha-org-contacts-template-email)
:END:")

        ("w" "Web" entry (file "~/sync/orgzly/Inbox.org")
        "* %a

:PROPERTIES:
:CREATED: %U
:END:

%i
")

        ("W" "Web bookmark" entry (file "~/sync/orgzly/resources.org")
        "* %a

:PROPERTIES:
:CREATED: %U
:END:

%i
"

        :prepend t)
        ("y" "Yay Emacs" entry (file+headline ↵
        ↵ "~/proj/stream/index.org" "Notes for this session")
        "* %?\n:PROPERTIES:\n:CREATED: %U\n:END:\n

%i

%a
"))

org-capture-templates))))

```

```
(keymap-global-set "C-M-r" 'org-capture)

;;(keymap-global-set (kbd "<f5>") 'org-capture)
```

Allow refiling in the middle(ish) of a capture This lets me use `C-c C-r` to refile a capture and then jump to the new location. I wanted to be able to file tasks under projects so that they could inherit the QUANTIFIED property that I use to track time (and any Beeminder-related properties too), but I also wanted to be able to clock in on them.

```
;;;###autoload
(defun sacha-org-refile-and-jump ()
  (interactive)
  (if (derived-mode-p 'org-capture-mode)
      (org-capture-refile)
      (call-interactively 'org-refile))
  (org-refile-goto-last-stored))
```

```
(with-eval-after-load 'org-capture
  (keymap-set org-capture-mode-map "C-c C-r" 'sacha-org-refile-and-jump))
```

8.16.3 Try out this capture command

From <https://takeonrules.com/2022/10/16/adding-another-function-to-sacha-workflow/>, modified slightly so that it creates a new entry if we are not currently clocked in. `C-c c` is a high-value keybinding, but I haven't been using this enough. I wonder where else it makes sense to have this.

```
(use-package git-link :defer t)
(keymap-global-set "C-c c" 'jf/capture-region-contents-with-metadata)
```

```
;;;###autoload
(defun sacha-org-capture-region-contents-with-metadata (start end parg)
  "Write selected text between START and END to currently clocked `org-mode' entry.

With PARG, kill the content instead.
If there is no clocked task, create it as a new note in my inbox instead.

From https://takeonrules.com/2022/10/16/adding-another-function-to-sacha-workflow/, modified ↵
↳ slightly so that it creates a new entry if we are not currently clocked in."
  (interactive "r\nP")
  (let ((text (sacha-org-region-contents-get-with-metadata start end)))
    (if (car parg)
        (kill-new text)
        (org-capture-string (concat "——\n" text)
                           (if (org-clocking-p) "c"
                               "r")))))

;;;###autoload
(defun sacha-org-region-contents-get-with-metadata (start end)
  "Get the region contents between START and END and return an `org-mode' formatted string.

From https://takeonrules.com/2022/10/16/adding-another-function-to-sacha-workflow/"
```

```

(require 'magit)
(require 'git-link)
(let* ((file-name (buffer-file-name (current-buffer)))
      (org-src-mode (replace-regexp-in-string
                     "-mode"
                     ""
                     (format "%s" major-mode))))
  (func-name (which-function))
  (type (if (derived-mode-p 'prog-mode) "SRC" "EXAMPLE"))
  (code-snippet (buffer-substring-no-properties start end))
  (file-base (file-name-nondirectory file-name))
  (line-number (line-number-at-pos (region-beginning)))
  (remote-link (when (magit-list-remotes)
                   (progn
                     (call-interactively 'git-link)
                     (car kill-ring))))
  (initial-txt (if (null func-name)
                   (format "From [[file:%s::%s][%s]]:"
                           file-name
                           line-number
                           file-base)
                   (format "From ~%s~ (in [[file:%s::%s][%s]]):"
                           func-name
                           file-name
                           line-number
                           file-base))))
  (format (concat "\n- Local :: %s"
                  (when remote-link (format "\n- Remote :: %s" remote-link))
                  "\n\n#+BEGIN_%s %s"
                  "\n%s"
                  "\n\n#+END_%s\n")
          initial-txt
          type
          org-src-mode
          code-snippet
          type)))

```

8.16.4 Estimating WPM

I'm curious about how fast I type some things.

```

(require 'org-clock)
;;;###autoload
(defun sachal-org-entry-wpm ()
  (interactive)
  (save-restriction
    (save-excursion
      (org-narrow-to-subtree)
      (goto-char (point-min))
      (let* ((words (count-words-region (point-min) (point-max)))
            (minutes (org-clock-sum-current-item))
            (wpm (/ words minutes)))
        (message "WPM: %d (words: %d, minutes: %d)" wpm words minutes)
        (kill-new (number-to-string wpm)))))

```

8.17 Logbook


```
;; ##autoload
(defun sacha-org-log-note (note)
  "Add NOTE to the current entry's logbook."
  (interactive "MNote: ")
  (setq org-log-note-window-configuration (current-window-configuration))
  (move-marker org-log-note-return-to (point))
  (move-marker org-log-note-marker (point))
  (setq org-log-note-purpose 'note)
  (with-temp-buffer
    (insert note)
    (org-store-log-note)))
```

8.18 Tasks

8.18.1 Managing tasks

Get things to be set to TODO when they repeat I use `STARTED` to keep track of things I've started working on, and this is updated when I clock into the task. If I have a repeating task, I want the task to be automatically reset to `TODO`.

```
(setq org-todo-repeat-to-state "TODO")
```

Track TODO state The parentheses indicate keyboard shortcuts that I can use to set the task state. `@` and `!` toggle logging. `@` prompts you for a note, and `!` automatically logs the timestamp of the state change.

```
(setq org-todo-keywords
  '((sequence
    "STARTED(s)"
    "TODO(t)" ; next action
    "TOBLOG(b)"
    "WAITING(w@/!)"
    "READY(r)"
    "SOMEDAY(.)" "BLOCKED(k@/!)" "|" "DONE(x!)" "CANCELLED(c)"
    (sequence "PROJECT" "|" "DONE(x)")
    (sequence "LEARN" "TRY" "TEACH" "|" "COMPLETE(x)")
    (sequence "TOSKETCH" "SKETCHED" "|" "POSTED")
    (sequence "TOBUY" "TOSHRINK" "TOCUT" "TOSEW" "|" "DONE(x)")
    (sequence "TODELEGATE(-)" "DELEGATED(d)" "|" "COMPLETE(x))))))
```

```
(setq org-log-done 'time)
```

TODO Change Org Mode TODO keyword color based on the state and the current Modus theme I use `modus-theme-toggle` to switch between `modus-vivendi-tinted` and `modus-operandi-tinted` depending on whether I want a dark background or a light one. I also customize my `org-todo-keyword-faces` to visually distinguish TODO, DONE, WAITING, and SOMEDAY. This is how to colour them based on the current Modus theme.

```
;; ##autoload
(defun sacha-org-todo-set-keyword-faces ()
  (setq org-todo-keyword-faces
    `(("TODO" . (:foreground ,(modus-themes-get-color-value ←
      ↪ 'blue-warmer) :weight bold))
```

```

    ↪ ("DONE" . (:foreground ,(modus-themes-get-color-value ↪
    ↪ 'green-warmer) :weight bold))
    ↪ ("WAITING" . (:foreground ↪
    ↪ ,(modus-themes-get-color-value 'red-warmer) :weight bold))
    ↪ ("SOMEDAY" . (:foreground ↪
    ↪ ,(modus-themes-get-color-value 'fg-dim) :weight bold))))
    (when (derived-mode-p 'org-mode)
      (font-lock-fontify-buffer)))

```

```

(with-eval-after-load 'modus-themes
  (add-hook 'modus-themes-after-load-theme-hook #'sacha-org-todo-set-keyword-faces))

```



Figure 11: Light background



Figure 12: Dark background

Projects Projects are headings with the `:project:` tag, so we generally don't want that tag inherited, except when we display unscheduled tasks that don't belong to any projects.

```

(setq org-tags-exclude-from-inheritance '("project" "inboxtarget"))

```

This code makes it easy for me to focus on one project and its tasks.

```
(with-eval-after-load 'org
  (let ((listvar (if (boundp 'org-speed-commands) 'org-speed-commands
                    'org-speed-commands-user)))
    (add-to-list listvar '("N" org-narrow-to-subtree))
    (add-to-list listvar '("W" widen))
    (add-to-list listvar '("T" sacha-org-agenda-for-subtree))
    (add-to-list listvar '("b" sacha-org-bounce-to-file))))
```

```
;;###autoload
(defun sacha-org-agenda-for-subtree ()
  (interactive)
  (when (derived-mode-p 'org-agenda-mode) (org-agenda-switch-to))
  (sacha-org-with-current-task
   (let ((org-agenda-view-columns-initially t))
     (org-agenda nil "t" 'subtree))))
```

There's probably a proper way to do this, maybe with `<`. Oh, that would work nicely. `< C-c a t` too.

And sorting:

```
(with-eval-after-load 'org
  (let ((listvar (if (boundp 'org-speed-commands) 'org-speed-commands
                    'org-speed-commands-user)))
    (add-to-list listvar '("S" call-interactively 'org-sort))))
```

Tag tasks with GTD-ish contexts This defines keyboard shortcuts for those, too.

```
(setq org-tag-alist '(("work" . ?b)
                     ("home" . ?h)
                     ("writing" . ?w)
                     ("errands" . ?e)
                     ("drawing" . ?d)
                     ("coding" . ?c)
                     ("video" . ?v)
                     ("kaizen" . ?k)
                     ("phone" . ?p)
                     ("learning" . ?a)
                     ("reading" . ?r)
                     ("computer" . ?l)
                     ("quantified" . ?q)
                     ("shopping" . ?s)
                     ("focus" . ?f)))
```

Enable filtering by effort estimates That way, it's easy to see short tasks that I can finish.

```
(with-eval-after-load 'org
  (add-to-list 'org-global-properties
    '("Effort_ALL". "0:05 0:15 0:30 1:00 ↵
    ↵ 2:00 3:00 4:00")))
```

Track time

```
(use-package org
  :init
  (progn
    (setq org-expiry-inactive-timestamps t)
    (setq org-clock-idle-time nil)
    (setq org-log-done 'time)
    (setq org-clock-auto-clock-resolution nil)
    (setq org-clock-continuously nil)
    (setq org-clock-persist t)
    (setq org-clock-in-switch-to-state "STARTED")
    (setq org-clock-in-resume nil)
    (setq org-show-notification-handler 'message)
    (setq org-clock-report-include-clocking-task t))
  :config
  (org-clock-persistence-insinuate))
```

Too many clock entries clutter up a heading.

```
(setq org-log-into-drawer "LOGBOOK")
(setq org-clock-into-drawer 1)
```

Habits I like using org-habits to track consistency. My task names tend to be a bit long, though, so I've configured the graph column to show a little bit more to the right.

```
(setq org-habit-graph-column 80)
(setq org-habit-show-habits-only-for-today nil)
```

If you want to use habits, be sure to schedule your tasks and add a STYLE property with the value of `habit` to the tasks you want displayed.

8.18.2 Estimating tasks

From "Add an effort estimate on the fly when clocking in" on the [Org Hacks](#) page:

```
(add-hook 'org-clock-in-prepare-hook
  'sacha-org-mode-ask-effort)
```

```
;;###autoload
(defun sacha-org-mode-ask-effort ()
  "Ask for an effort estimate when clocking in."
  (unless (org-entry-get (point) "Effort")
    (let ((effort
      (completing-read
        "Effort: "
        (org-entry-get-multivalued-property (point) "Effort"))))
      (unless (equal effort "")
        (org-set-property "Effort" effort)))))
```

8.18.3 Flexible scheduling of tasks

I (theoretically) want to be able to schedule tasks for dates like the first Saturday of every month. Fortunately, [someone else has figured that out!](#)

```
;; Get this from https://raw.githubusercontent.com/chenfengyuan/elisp/master/next-spec-day.el
(load "~/elisp/next-spec-day.el" t)
```

8.18.4 Task dependencies

```
(setq org-enforce-todo-dependencies t)
(setq org-track-ordered-property-with-tag t)
(setq org-agenda-dim-blocked-tasks t)
```

8.18.5 Quick way to archive all DONE from inbox

emacs:computer

```
;; ##autoload
(defun sach-org-clean-up-inbox ()
  "Archive all DONE tasks and sort the remainder by TODO order."
  (interactive)
  (with-current-buffer (find-file sach-org-inbox-file)
    (sach-org-archive-done-tasks 'file)
    (goto-char (point-min))
    (if (org-at-heading-p) (save-excursion (insert "\n"))))
    (org-sort-entries nil ?p)
    (goto-char (point-min))
    (org-sort-entries nil ?o)
    (save-buffer)))

;; ##autoload
(defun sach-org-archive-done-tasks (&optional scope)
  "Archive finished or cancelled tasks.
   SCOPE can be 'file or 'tree."
  (interactive)
  (org-map-entries
   (lambda ()
     (org-archive-subtree)
     (setq org-map-continue-from (outline-previous-heading))))
   "TODO=\"DONE\"|TODO=\"CANCELLED\" (or scope (if (org-before-first-heading-p) 'file 'tree))))
```

8.18.6 Strike through DONE headlines

I wanted a quick way to visually distinguish DONE tasks from tasks I still need to do. This [handy snippet from the Emacs Org-mode mailing list](#) does the trick by striking through the headlines for DONE tasks.

```
(setq org-fontify-done-headline t)
(custom-set-faces
 '(org-done ((t (:foreground "PaleGreen"
                          :weight normal
                          :strike-through t))))
 '(org-headline-done
  (((class color) (min-colors 16) (background dark))
   (:foreground "LightSalmon" :strike-through t))))
```

8.18.7 Checklists

<https://orgmode.org/worg/org-contrib/org-checklist.html>

```
(with-eval-after-load 'org
  (require 'org-checklist))
```

8.19 Templates

8.19.1 Structure templates

Org makes it easy to insert blocks by typing `<s[TAB]`, etc. I hardly ever use $\text{\LaTeX}$, but I insert a lot of Emacs Lisp blocks, so I redefine `<l` to insert a Lisp block instead.

```
(setq org-structure-template-alist
  '(("a" . "export ascii")
    ("C" . "center")
    ("c" . "comment")
    ("d" . "my_details")
    ("e" . "example")
    ("E" . "export")
    ("m" . "export md")
    ("M" . "media-post")
    ("h" . "export html")
    ("j" . "src js :spookfox t")
    ("l" . "src emacs-lisp")
    ("p" . "src python")
    ("n" . "notes")
    ("q" . "quote")
    ("s" . "src")
    ("S" . "src sh")
    ("u" . "update")
    ("v" . "verse")))
```

This lets me nest quotes. <http://emacs.stackexchange.com/questions/2404/exporting-org-mode-nested-blocks-to-html>

```
;;;###autoload
(defun sacha-org-html-quote2 (block backend info)
  (when (org-export-derived-backend-p backend 'html)
    (when (string-match "\\`<div class=\"quote2\\`>" block)
      (setq block (replace-match "<blockquote>" t nil block))
      (string-match "</div>\n\\`" block)
      (setq block (replace-match "</blockquote>\n" t nil block))
      block)))
```

```
(eval-after-load 'ox
  '(add-to-list 'org-export-filter-special-block-functions 'sacha-org-html-quote2))
```

Demarcate, but for all blocks

EMACS:CONFIG

I often want to split an Org Mode block so that I can add stuff in between. This code is based on <https://scripter.co/splitting-an-org-block-into-two/>.

```
;;;###autoload
(defun modi/org-split-block ()
  "Sensibly split the current Org block at point."
  (interactive))
```

```

(if (modi/org-in-any-block-p)
  (save-match-data
    (save-restriction
      (widen)
      (let ((case-fold-search t)
            (at-bol (bolp))
            block-start
            block-end)
        (save-excursion
          (re-search-backward "^\\(?:[:blank:]]*#\\+begin_.+?\\)\\(?:[:blank:]]*\\)$" nil nil 1)
          (setq block-start (match-string-no-properties 0))
          (setq block-end (replace-regexp-in-string
                           "begin_" "end_" ;Replaces "begin_" with "end_", "BEGIN_" with ↵
                           ↵ "END_"
                           (match-string-no-properties 1))))
          ;; Go to the end of current line, if not at the BOL
          (unless at-bol
            (end-of-line 1))
          (insert (concat (if at-bol "" "\n")
                          block-end
                          "\n\n"
                          block-start
                          (if at-bol "\n" "")))
          ;; Go to the line before the inserted "#+begin_ .." line
          (beginning-of-line (if at-bol -1 0))))
    (message "Point is not in an Org block")))
(defalias 'sacha-org-demarcate-block #'modi/org-split-block)
(defalias 'sacha-org-split-block #'modi/org-split-block)

;;###autoload
(defun modi/org-in-any-block-p ()
  "Return non-nil if the point is in any Org block."

  The Org block can be *any*: src, example, verse, etc., even any
  Org Special block.

  This function is heavily adapted from `org-between-regexps-p'."
  (save-match-data
    (let ((pos (point))
          (case-fold-search t)
          (block-begin-re "^[:blank:]]*#\\+begin_\\(?:[:blank:]]*\\.+?\\)\\(?:[:blank:]]*\\)$"
              (limit-up (save-excursion (outline-previous-heading)))
              (limit-down (save-excursion (outline-next-heading)))
              beg end)
          (save-excursion
            ;; Point is on a block when on BLOCK-BEGIN-RE or if
            ;; BLOCK-BEGIN-RE can be found before it...
            (and (or (org-in-regexp block-begin-re)
                     (re-search-backward block-begin-re limit-up :noerror))
                 (setq beg (match-beginning 0))
                 ;; ... and BLOCK-END-RE after it...
                 (let ((block-end-re (concat "^[:blank:]]*#\\+end_"
                                              (match-string-no-properties 1)
                                              "\\(?:[:blank:]]*\\)$")))
                   (goto-char (match-end 0))
                   (re-search-forward block-end-re limit-down :noerror))
                 (> (setq end (match-end 0)) pos)
                 ;; ... without another BLOCK-BEGIN-RE in-between.

```

```

(goto-char (match-beginning 0))
(not (re-search-backward block-begin-re (1+ beg) :noerror))
;; Return value.
(cons beg end))))))

```

8.19.2 Emacs chats, Emacs hangouts

```

;;;###autoload
(defun sacha-org-link-youtube-time (url beg end)
  "Link times of the form h:mm to YouTube video at URL.
   Works on region defined by BEG and END."
  (interactive (list (read-string "URL: " (org-entry-get-with-inheritance "YOUTUBE")) (point) ←
    ↪ (mark)))
  (save-excursion
    (save-restriction
      (narrow-to-region beg end)
      (goto-char (point-min))
      (let ((char (if (string-match "\\?" url) "&" "?")))
        (while (re-search-forward "\\(\\([0-9]+\\):\\([0-9]+\\)\\)\\(:\\([0-9]+\\)\\)\\)?\\)" nil t)
          (replace-match
            (format "[[%s%st=%sh%sm%ss][%s]] "
              url
              char
              (match-string 2)
              (match-string 3)
              (or (match-string 5) "0")
              (match-string 1)) nil t))))))

;;;###autoload
(defun sacha-clean-up-google-hangout-chat ()
  (interactive)
  (save-excursion
    (while (re-search-forward "<hr.*?div class=\"Kc-Ma-m\".*?>" nil t)
      (replace-match "\\n| ")))
  (save-excursion
    (while (re-search-forward "</div><div class=\"Kc-yi-m\">" nil t)
      (replace-match " | ")))
  (save-excursion
    (while (re-search-forward "</div></div><div class=\"Kc-ib\">" nil t)
      (replace-match " | ")))
  (save-excursion
    (while (re-search-forward "<a rel=\"nofollow\" target=\"_blank\" ←
    ↪ href=\"\\(\\.\\.*?\\)\">\\(\\.\\.*?\\)</a>" nil t)
      (replace-match "[[\\1][\\2]]")))
  (save-excursion
    (while (re-search-forward "</div></div></div></div>" nil t)
      (replace-match " | ")))
  (save-excursion
    (while (re-search-forward "&nbsp;" nil t)
      (replace-match " ")))
  (save-excursion
    (while (re-search-forward "</div><div class=\"Kc-ib\">" nil t)
      (replace-match " ")))
  (save-excursion
    (while (re-search-forward "<img.*?>" nil t)
      (replace-match "")))
  (save-excursion

```



```
(while (re-search-forward "<wbr>" nil t)
  (replace-match ""))
)
```

8.20 Org agenda

8.20.1 Basic configuration

I have quite a few Org files, but I keep my agenda items and TODOs in only a few of them them for faster scanning.

```
(defvar sachakid-org-file nil "Defined in secrets")
(setq sachakid-org-agenda-files
  (delq nil
    (mapcar (lambda (x) (and x (file-exists-p x) x))
      `("~/sync/orgzly/organizer.org"

↳ "~/sync/orgzly/ipad.org"
      "~/sync/orgzly/Inbox.org"
      "~/sync/orgzly/garden.org"
      "~/sync/orgzly/decisions.org"
      "~/sync/orgzly/computer-inbox.org"
      "~/sync/orgzly/posts.org"
      "~/sync/orgzly/crafts.org"
      "~/sync/emacs/Sacha.org"
      "~/proj/emacsconf/wiki/2025/organizers-notebook/index.org"
      "~/proj/emacsconf/wiki/organizers-notebook/index.org"
      "~/proj/stream/index.org"
      "~/proj/plover-notes/README.org"
      "~/personal/sewing.org"
      "~/sync/orgzly/people.org"
      "~/sync/orgzly/business.org"
      "~/Dropbox/wsmef/trip.txt"
      ,sachakid-org-file
      "~/personal/orgzly.org"
      "~/personal/calendar.org"
      "~/Dropbox/tasker/summary.txt"
      "~/Dropbox/public/sharing/index.org"
      "~/dropbox/public/sharing/learning.org"
      "~/proj/emacs-notes/tasks.org"
      "~/proj/sachac.github.io/evil-plans/index.org"
      "~/sync/orgzly/cooking.org"
      "~/sync/orgzly/routines.org"))))
(setq org-agenda-files sachakid-org-agenda-files)
(setq org-agenda-dim-blocked-tasks nil)
(add-to-list 'auto-mode-alist '("\\.txt$" . org-mode))
```

I like looking at two days at a time when I plan using the Org agenda. I want to see my log entries, but I don't want to see scheduled items that I've finished. I like seeing a time grid so that I can get a sense of how appointments are spread out.

```
(setq org-agenda-span 2)
(setq org-agenda-tags-column -100) ; take advantage of the screen width
(setq org-agenda-sticky nil)
(setq org-agenda-inhibit-startup t)
(setq org-agenda-use-tag-inheritance t)
(setq org-agenda-show-log t)
```

```
(setq org-agenda-skip-scheduled-if-done t)
(setq org-agenda-skip-deadline-if-done t)
(setq org-agenda-skip-deadline-prewarning-if-scheduled 'pre-scheduled)
(setq org-agenda-time-grid
  '((daily today require-timed)
    (800 1000 1200 1400 1600 1800 2000)
    "....." "_____"))
(setq org-columns-default-format "%14SCHEDULED %Effort{:} %1PRIORITY %TODO %50ITEM %TAGS")
```

Some other keyboard shortcuts:

```
(with-eval-after-load 'org
  (keymap-set org-agenda-mode-map "Y" 'org-agenda-todo-yesterday))
```

8.20.2 Starting my weeks on Saturday

I like looking at weekends as [week beginnings](#) instead, so I want the Org agenda to start on Saturdays.

```
(setq org-agenda-start-on-weekday 6)
```

8.20.3 Display projects with associated subtasks

I wanted a view that showed projects with a few subtasks underneath them. Here's a sample of the output:

```
Headlines with TAGS match: +PROJECT
Press `C-u r' to search again with new search string
organizer: Set up communication processes for Awesome Foundation Toronto
organizer: TODO Announce the next pitch night
organizer: TODO Follow up with the winner of the previous pitch night for any news to include i

organizer: Tidy up the house so that I can find things quickly
organizer: TODO Inventory all the things in closets and boxes           :@home:
organizer: TODO Drop things off for donation                          :@errands:

organizer: Learn how to develop for Android devices
```

```
;; ##autoload
(defun sachar-org-agenda-project-agenda ()
  "Return the project headline and up to `org-agenda-max-entries' tasks."
  (save-excursion
    (let* ((marker (org-agenda-new-marker))
           (heading
            (org-agenda-format-item "" (org-get-heading) (org-get-category) nil))
           (org-agenda-restrict t)
           (org-agenda-restrict-begin (point))
           (org-agenda-restrict-end (org-end-of-subtree 'invisible))
           ;; Find the TODO items in this subtree
           (list (org-agenda-get-day-entries (buffer-file-name) (calendar-current-date) :todo)))
      (org-add-props heading
        (list 'face 'defaults
              'done-face 'org-agenda-done
              'undone-face 'default
              'mouse-face 'highlight
              'org-not-done-regexp org-not-done-regexp
```

```

'org-todo-regexp org-todo-regexp
'org-complex-heading-regexp org-complex-heading-regexp
'help-echo
(format "mouse-2 or RET jump to org file %s"
  (abbreviate-file-name
    (or (buffer-file-name (buffer-base-buffer))
        (buffer-name (buffer-base-buffer))))))
'org-marker marker
'org-hd-marker marker
'org-category (org-get-category)
'type "tagsmatch")
(concat heading "\n"
  (org-agenda-finalize-entries list))))))

;; ###autoload
(defun sacha-org-agenda-projects-and-tasks (match)
  "Show TODOs for all `org-agenda-files' headlines matching MATCH."
  (interactive "MString: ")
  (let ((todo-only nil))
    (if org-agenda-overriding-arguments
        (setq todo-only (car org-agenda-overriding-arguments)
              match (nth 1 org-agenda-overriding-arguments)))
    (let* ((org-tags-match-list-sublevels
            org-tags-match-list-sublevels)
           (completion-ignore-case t)
           rtn rtnall files file pos matcher
           buffer)
      (when (and (stringp match) (not (string-match "\\S-" match)))
        (setq match nil))
      (when match
        (setq matcher (org-make-tags-matcher match)
              match (car matcher) matcher (cdr matcher)))
      (catch 'exit
        (if org-agenda-sticky
            (setq org-agenda-buffer-name
                  (if (stringp match)
                      (format "*Org Agenda(%s:%s)*"
                              (or org-keys (or (and todo-only "M") "m"))) match)
                      (format "*Org Agenda(%s)*" (or (and todo-only "M") "m")))))
            (org-agenda-prepare (concat "TAGS " match)))
        (org-compile-prefix-format 'tags)
        (org-set-sorting-strategy 'tags)
        (setq org-agenda-query-string match)
        (setq org-agenda-redo-command
              (list 'org-tags-view `(quote ,todo-only)
                    (list 'if 'current-prefix-arg nil `(quote ,org-agenda-query-string)))))
        (setq files (org-agenda-files nil 'ifmode)
              rtnall nil)
        (while (setq file (pop files))
          (catch 'nextfile
            (org-check-agenda-file file)
            (setq buffer (if (file-exists-p file)
                            (org-get-agenda-file-buffer file)
                            (error "No such file %s" file)))
            (if (not buffer)
                ;; If file does not exist, error message to agenda
                (setq rtn (list
                          (format "ORG-AGENDA-ERROR: No such org-file %s" file))
                      rtnall (append rtnall rtn))
                  (setq rtn (list
                          (format "ORG-AGENDA-ERROR: No such org-file %s" file))
                      rtnall (append rtnall rtn))

```

```

(with-current-buffer buffer
  (unless (derived-mode-p 'org-mode)
    (error "Agenda file %s is not in `org-mode'" file))
  (save-excursion
    (save-restriction
      (if org-agenda-restrict
        (narrow-to-region org-agenda-restrict-begin
                          org-agenda-restrict-end)
        (widen))
      (setq rtn (org-scan-tags 'sacha-org-agenda-project-agenda matcher todo-only))
      (setq rtnall (append rtnall rtn))))))
(if org-agenda-overriding-header
  (insert (org-add-props (copy-sequence org-agenda-overriding-header)
                        nil 'face 'org-agenda-structure) "\n")
  (insert "Headlines with TAGS match: ")
  (add-text-properties (point-min) (1- (point))
    (list 'face 'org-agenda-structure
          'short-heading
          (concat "Match: " match))))
(setq pos (point))
(insert match "\n")
(add-text-properties pos (1- (point)) (list 'face 'org-warning))
(setq pos (point))
(unless org-agenda-multi
  (insert "Press `C-u r' to search again with new search string\n"))
(add-text-properties pos (1- (point)) (list 'face 'org-agenda-structure)))
(org-agenda-mark-header-line (point-min))
(when rtnall
  (insert (mapconcat 'identity rtnall "\n") ""))
(goto-char (point-min))
(or org-agenda-multi (org-agenda-fit-window-to-buffer))
(add-text-properties (point-min) (point-max)
  `(org-agenda-type tags
    org-last-args (,todo-only ,match)
    org-redo-cmd ,org-agenda-redo-command
    org-series-cmd ,org-cmd))
(org-agenda-finalize)
(setq buffer-read-only t))))

```

8.20.4 Org agenda custom commands

There are quite a few custom commands here, and I often forget to use them. =) But it's good to define them, and over time, I'll get the hang of using these more!

Key	Description
.	What am I waiting for?
T	Not really an agenda command - shows the to-do tree in the current file
b	Shows business-related tasks
o	Shows personal tasks and miscellaneous tasks (o: organizer)
w	Show all tasks for the upcoming week
W	Show all tasks for the upcoming week, aside from the routine ones
g ...	Show tasks by context: b - business; c - coding; w - writing; p - phone; d - drawing, h - home
0	Show common contexts with up to 3 tasks each, so that I can choose what I feel like working on
) (shift-0)	Show common contexts with all the tasks associated with them
9	Show common contexts with up to 3 unscheduled tasks each
((shift-9)	Show common contexts with all the unscheduled tasks associated with them
d	Timeline for today (agenda, clock summary)
u	Unscheduled tasks to do if I have free time
U	Unscheduled tasks that are not part of projects
P	Tasks by priority
p	My projects
2	Projects with tasks

```
(keymap-global-set "<apps> a" 'org-agenda)
(setq sach-a-org-agenda-contexts
  '((tags-todo "phone")
    (tags-todo "work")
    (tags-todo "drawing")
    (tags-todo "coding")
    (tags-todo "writing")
    (tags-todo "computer")
    (tags-todo "home")
    (tags-todo "errands"))))
```

```
(defvar sach-a-org-agenda-contexts
  nil
  "Usual list of contexts.")
;;###autoload
(defun sach-a-org-agenda-skip-scheduled ()
  (org-agenda-skip-entry-if 'scheduled 'deadline 'regexp "\n]+>"))
;;###autoload
(defun sach-a-org-projects ()
  (interactive)
  (org-ql-search (org-agenda-files)
    '(and (todo "TODO" "WAITING") (ancestors (tags "project"))))
    :super-groups '(:auto-parent t)))
```

```
(use-package org-super-agenda
  :init
  (org-super-agenda-mode 1))
(use-package org-ql)

(setq org-agenda-custom-commands
  `(("a" "Agenda"
    ((agenda "" ((org-agenda-span 2)))
     ;; (alltodo
     ;;   ""
     ;;   ((org-agenda-overriding-header ""))
```

```

;; (org-super-agenda-groups
;; '(:name "Inbox, unscheduled"
;;       :and (:scheduled nil
;;             ;;
;→                                     :file-path "Inbox.org"
;→                                     ;;
;→                                     )
;;
;;       :order 1)
;;     (:name "Important, unscheduled"
;;           :and (:priority "A"
;;                 :scheduled nil)
;;           :order 2)
;;
;;     (:name "Project-related, unscheduled"
;;           :and (:tag "project" :date nil :todo ("STARTED" "WAITING" "TODO"))
;;           :order 3)
;;     (:name "Waiting"
;;           :and (:todo "WAITING"
;;                   :scheduled nil)
;;           :order 4)
;;     (:discard (:todo "SOMEDAY"
;;                     :category "cooking"
;;                     :date t))
;;     (:name "Unscheduled"
;;           :scheduled nil
;;           :order 5)
;;     (:discard (:anything t))
;;   )
;; ))
;; (tags-todo "TODO=\"TODO\"-project-cooking-routine-errands-shopping-video-evilplans"
;;   ((org-agenda-skip-function 'sacha-org-agenda-skip-scheduled)
;;    (org-agenda-prefix-format "%-6e ")
;;    (org-agenda-overriding-header "Unscheduled TODO entries: ")
;;    (org-agenda-sorting-strategy '(priority-down effort-up tag-up ←
;→ category-keep))))
;; )
("e" "Emacs" tags "emacs")
      ("n" "Emacs News" tags "news" ((org-agenda-files ←
;→
;→                                     ←
;→                                     " ~/sync/orgzly/news.org" )))
;→ " ~/sync/orgzly/news.org" )))
("E" "Emacsconf" tags-todo "emacsconf"
      ((org-agenda-sorting-strategy '(priority-down effort-up ←
;→ category-keep)))
      )
("i" "Inbox" alltodo ""
  ((org-agenda-files '("~/sync/orgzly/Inbox.org" "~/sync/orgzly/computer-inbox.org" )))
  ("s" tags-todo "stream"
    ((org-agenda-sorting-strategy '(todo-state-up priority-down effort-up))))
  ("t" tags-todo "-cooking"
    ((org-agenda-sorting-strategy '(todo-state-up priority-down effort-up))))
  ("T" tags-todo "TODO=\"TODO\"-goal-routine-cooking-SCHEDULED={.+}" nil ←
;→ " ~/cloud/agenda/nonroutine.html")
  ("f" tags-todo "focus-TODO=\"DONE\"-TODO=\"CANCELLED\"" )
  ("b" todo ""
    ((org-agenda-files '("~/sync/orgzly/business.org" )))
  ("B" todo ""

```

```

((org-agenda-files '("~/Dropbox/books"))))
("x" "Column view" todo "" ; Column view
  ((org-agenda-prefix-format "")
   (org-agenda-cmp-user-defined 'sacha-org-sort-agenda-items-todo)
   (org-agenda-view-columns-initially t)
  ))
;; Weekly review
("w" "Weekly review" agenda ""
  ((org-agenda-span 7)
   (org-agenda-log-mode 1)) "~/cloud/agenda/this-week.html")
("W" "Weekly review sans routines" agenda ""
  ((org-agenda-span 7)
   (org-agenda-log-mode 1)
   (org-agenda-tag-filter-preset '("-routine")))) ←
↪ "~/cloud/agenda/this-week-nonroutine.html")
("2" "Bi-weekly review" agenda "" ((org-agenda-span 14) (org-agenda-log-mode 1)))
("5" "Quick tasks" tags-todo "EFFORT ≥ \"0:05\" & EFFORT ≤ \"0:15\"")
("0" "Unestimated tasks" tags-todo "EFFORT=\\\"\\\"")
("gb" "Business" todo ""
  ((org-agenda-files '("~/sync/orgzly/business.org"))
   (org-agenda-view-columns-initially t)))
("gc" "Coding" tags-todo "@coding"
  ((org-agenda-view-columns-initially t)))
("gw" "Writing" tags-todo "@writing"
  ((org-agenda-view-columns-initially t)))
("gp" "Phone" tags-todo "@phone"
  ((org-agenda-view-columns-initially t)))
("gd" "Drawing" tags-todo "@drawing"
  ((org-agenda-view-columns-initially t)))
("gh" "Home" tags-todo "@home"
  ((org-agenda-view-columns-initially t)))
("gk" "Kaizen" tags-todo "kaizen"
  ((org-agenda-view-columns-initially t)
   ("~/cloud/agenda/kaizen.html")))
("ge" "Errands" tags-todo "errands"
  ((org-agenda-view-columns-initially t)
   ("~/cloud/agenda/errands.html")))
("c" "Top 3 by context"
  ,sacha-org-agenda-contexts
  ((org-agenda-sorting-strategy '(priority-up effort-down))
   (org-agenda-max-entries 3)))
("C" "All by context"
  ,sacha-org-agenda-contexts
  ((org-agenda-sorting-strategy '(priority-down effort-down))
   (org-agenda-max-entries nil)))
("9" "Unscheduled top 3 by context"
  ,sacha-org-agenda-contexts
  ((org-agenda-skip-function 'sacha-org-agenda-skip-scheduled)
   (org-agenda-sorting-strategy '(priority-down effort-down))
   (org-agenda-max-entries 3)))
("(" "All unscheduled by context"
  ,sacha-org-agenda-contexts
  ((org-agenda-skip-function 'sacha-org-agenda-skip-scheduled)
   (org-agenda-sorting-strategy '(priority-down effort-down))
  ))
("d" "Timeline for today" ((agenda "" ))
  ((org-agenda-ndays 1)
   (org-agenda-show-log t)
   (org-agenda-log-mode-items '(clock closed))

```

```

(org-agenda-clockreport-mode t)
(org-agenda-entry-types '()))
("." "Waiting for" todo "WAITING")
("u" "Unscheduled tasks" tags-todo ←
↪ "-someday-TODO=\"SOMEDAY\"-TODO=\"DELEGATED\"-TODO=\"WAITING\"-project-cooking-routine"
  ((org-agenda-skip-function 'sacha-org-agenda-skip-scheduled)
   (org-agenda-view-columns-initially nil)
   (org-tags-exclude-from-inheritance '("project"))
   (org-agenda-overriding-header "Unscheduled TODO entries: ")
   (org-columns-default-format "%50ITEM %TODO %3PRIORITY %Effort{:} %TAGS")
   (org-agenda-sorting-strategy '(todo-state-up priority-down effort-up tag-up ←
↪ category-keep))))
  ("!" "Someday" tags-todo "TODO=\"SOMEDAY\""
   ((org-agenda-skip-function 'sacha-org-agenda-skip-scheduled)
    (org-agenda-view-columns-initially nil)
    (org-tags-exclude-from-inheritance '("project"))
    (org-agenda-overriding-header "Someday: ")
    (org-columns-default-format "%50ITEM %TODO %3PRIORITY %Effort{:} %TAGS")
    (org-agenda-sorting-strategy '(todo-state-up priority-down effort-up tag-up ←
↪ category-keep))))
  ("U" "Unscheduled tasks outside projects" tags-todo "-project-cooking-routine"
   ((org-agenda-skip-function 'sacha-org-agenda-skip-scheduled)
    (org-tags-exclude-from-inheritance nil)
    (org-agenda-view-columns-initially nil)
    (org-agenda-overriding-header "Unscheduled TODO entries outside projects: ")
    (org-agenda-sorting-strategy '(todo-state-up priority-down tag-up category-keep ←
↪ effort-down))))
  ("P" "By priority"
   ((tags-todo "+PRIORITY=\"A\""
    (tags-todo "+PRIORITY=\"B\""
    (tags-todo "+PRIORITY=\"\""
    (tags-todo "+PRIORITY=\"C\""))
   (org-agenda-prefix-format "%-10c %-10T %e ")
   (org-agenda-sorting-strategy '(priority-down tag-up category-keep effort-down))))
  ("pp" tags "+project-someday-TODO=\"DONE\"-TODO=\"SOMEDAY\"-inactive"
   ((org-tags-exclude-from-inheritance '("project"))
    (org-agenda-sorting-strategy '(priority-down tag-up category-keep effort-down))))
  ("p." tags "+project-TODO=\"DONE\""
   ((org-tags-exclude-from-inheritance '("project"))
    (org-agenda-sorting-strategy '(priority-down tag-up category-keep effort-down))))
  ("S" tags-todo "TODO=\"STARTED\""
  ("C" "Cooking"
   ((tags "vegetables")
    (tags "chicken")
    (tags "beef")
    (tags "pork")
    (tags "other")))
   (org-agenda-files '("~/sync/orgzly/cooking.org"))
   (org-agenda-view-columns-initially t)
   (org-agenda-sorting-strategy '(scheduled-up time-down todo-state-up)))
)
("8" "List projects with tasks" sacha-org-agenda-projects-and-tasks
 "+PROJECT"
 ((org-agenda-max-entries 3))))

```


8.20.5 Shuffling my Org Mode unscheduled tasks

emacs.org

I enjoyed listening to [Podcast #1,029: Treat Your To-Do List Like a River, and Other Mindset Shifts for Making Better Use of Your Time | The Art of Manliness](#) (thanks @ctietze@mastodon.social for the recommendation) and checking out the [Autofocus method](#) ([main website with translations](#)) for reviewing your TODO list without worrying too much about prioritization. I also had an opportunity to reflect on similar topics in a conversation with John Wiegley and Adam Porter about personal information management (which I'll be blogging about once John has a chance to review the draft).

This nudged me to experiment with randomizing my unscheduled task list. I'm not trying to finish everything on my list, I'm just mixing it up so that I enjoy keeping things on my radar and picking something to do. [org-ql](#) lets me create randomly-sorted views, so I wrote some code to show me a shuffled list of my unscheduled TODO tasks and SOMEDAY tasks.

```
;; ###autoload
(defun sacha-org-ql-shuffle-todo ()
  (interactive)
  (org-ql-search (org-agenda-files)
    '(and
      (todo "TODO" "STARTED")
      (not (done))
      (not (scheduled))
      (not (deadline))
      (not (ts-active))
      (not (tags "cooking")))
    :sort 'random))

;; ###autoload
(defun sacha-org-ql-shuffle-someday ()
  (interactive)
  (org-ql-search (org-agenda-files)
    '(and
      (todo "SOMEDAY")
      (not (done))
      (not (scheduled))
      (not (deadline))
      (not (ts-active))
      (not (tags "cooking")))
    :sort 'random))
```

I can't make it part of my [org-agenda-custom-commands](#) yet because of an [open issue](#), but having separate commands is a starting point. It's surprisingly fun. I used [org-agenda-follow-mode](#) to quickly flip through the tasks while looking at the agenda. I've already noticed some tasks to cancel and picked some tasks to do. Could be neat!

8.20.6 Making it easier to tag inbox items

```
(setq org-complete-tags-always-offer-all-agenda-tags t)
(setq org-use-fast-tag-selection nil)
```

8.20.7 Make it easy to mark a task as done

Great for quickly going through the to-do list. Gets rid of one extra keystroke. ;)

```
;; ###autoload
(defun sacha-org-agenda-done (&optional arg)
```

```
"Mark current TODO as done.
  This changes the line at point, all other lines in the agenda referring to
  the same tree node, and the headline of the tree node in the Org-mode file."
(interactive "P")
(org-agenda-todo "DONE"))
```

```
;; Override the key definition for org-exit
(define-key org-agenda-mode-map "x" 'sacha-org-agenda-done)
```

8.20.8 Make it easy to mark a task as done and create a follow-up task

```
;;;###autoload
(defun sacha-org-agenda-mark-done-and-add-followup ()
  "Mark the current TODO as done and add another task after it.
  Creates it at the same level as the previous task, so it's better to use
  this with to-do items than with projects or headings."
  (interactive)
  (org-agenda-todo "DONE")
  (org-agenda-switch-to)
  (org-capture 0 "t"))
```

```
;; Override the key definition
(define-key org-agenda-mode-map "F" 'sacha-org-agenda-mark-done-and-add-followup)
```

8.20.9 Capture something based on the agenda

```
;;;###autoload
(defun sacha-org-agenda-new ()
  "Create a new note or task at the current agenda item.
  Creates it at the same level as the previous task, so it's better to use
  this with to-do items than with projects or headings."
  (interactive)
  (org-agenda-switch-to)
  (org-capture 0))
```

```
;; New key assignment
(define-key org-agenda-mode-map "N" 'sacha-org-agenda-new)
```

8.20.10 Sorting by date and priority

```
(setq org-agenda-sorting-strategy
  '((agenda time-up priority-down tag-up category-keep)
    ;; (todo user-defined-up todo-state-up priority-down effort-up)
    (todo todo-state-up priority-down effort-up)
    ; (tags user-defined-up)
    (search category-keep)))
(setq org-agenda-cmp-user-defined 'sacha-org-sort-agenda-items-user-defined)
```

```
(require 'cl)
```

```

;;;###autoload
(defun sacha-org-get-context (txt)
  "Find the context."
  (car (member-if
        (lambda (item) (string-match "@" item))
        (get-text-property 1 'tags txt))))

;;;###autoload
(defun sacha-org-compare-dates (a b)
  "Return 1 if A should go after B, -1 if B should go after A, or 0 if a = b."
  (cond
   ((and (= a 0) (= b 0)) nil)
   ((= a 0) 1)
   ((= b 0) -1)
   ((> a b) 1)
   ((< a b) -1)
   (t nil)))

;;;###autoload
(defun sacha-org-complete-cmp (a b)
  (let* ((state-a (or (get-text-property 1 'todo-state a) ""))
         (state-b (or (get-text-property 1 'todo-state b) "")))
    (or
     (if (member state-a org-done-keywords-for-agenda) 1)
     (if (member state-b org-done-keywords-for-agenda) -1))))

;;;###autoload
(defun sacha-org-date-cmp (a b)
  (let* ((sched-a (or (get-text-property 1 'org-scheduled a) 0))
         (sched-b (or (get-text-property 1 'org-scheduled b) 0))
         (deadline-a (or (get-text-property 1 'org-deadline a) 0))
         (deadline-b (or (get-text-property 1 'org-deadline b) 0)))
    (or
     (sacha-org-compare-dates
      (sacha-org-min-date sched-a deadline-a)
      (sacha-org-min-date sched-b deadline-b)))))

;;;###autoload
(defun sacha-org-min-date (a b)
  "Return the smaller of A or B, except for 0."
  (funcall (if (and (> a 0) (> b 0)) 'min 'max) a b))

;;;###autoload
(defun sacha-org-sort-agenda-items-user-defined (a b)
  ; compare by deadline, then scheduled date; done tasks are listed at the very bottom
  (or
   (sacha-org-complete-cmp a b)
   (sacha-org-date-cmp a b)))

;;;###autoload
(defun sacha-org-context-cmp (a b)
  "Compare CONTEXT-A and CONTEXT-B."
  (let ((context-a (sacha-org-get-context a))
        (context-b (sacha-org-get-context b)))
    (cond
     ((null context-a) +1)
     ((null context-b) -1)
     ((string< context-a context-b) -1)
     ((string< context-b context-a) +1)

```

```

      (t nil))))
;; ##autoload
(defun sach-org-sort-agenda-items-todo (a b)
  (or
   (org-cmp-time a b)
   (sach-org-complete-cmp a b)
   (sach-org-context-cmp a b)
   (sach-org-date-cmp a b)
   (org-cmp-todo-state a b)
   (org-cmp-priority a b)
   (org-cmp-effort a b)))

```

8.20.11 Preventing things from falling through the cracks

This helps me keep track of unscheduled tasks, because I sometimes forget to assign tasks a date. I also want to keep track of stuck projects.

```

;; ##autoload
(defun sach-org-agenda-list-unscheduled (&rest ignore)
  "Create agenda view for tasks that are unscheduled and not done."
  (let* ((org-agenda-todo-ignore-with-date t)
         (org-agenda-overriding-header "List of unscheduled tasks: "))
    (org-agenda-get-todos)))

```

```

(setq org-stuck-projects
  '("+PROJECT-MAYBE-DONE"
    "TODO")
  nil
  "\\<IGNORE\\>"))

```

8.20.12 Synchronizing with Google Calendar

```

;; ##autoload
(defun sach-org-gcal-notify (title mes)
  (message "%s - %s" title mes))

```

```

(use-package org-gcal
  :if sach-laptop-p
  :load-path "~/elisp/org-gcal.el"
  :init (fset 'org-gcal-notify 'sach-org-gcal-notify))

```

8.20.13 Projects

```

;; ##autoload
(defun sach-org-show-active-projects ()
  "Show my current projects."
  (interactive)
  (org-tags-view nil "project-inactive-someday"))

```

8.21 Reviews

8.21.1 Weekly review

I regularly post [weekly reviews](#) to keep track of what I'm done, remind me to plan for the upcoming week, and list blog posts, sketches, and links. I want to try out grouping tasks by topic first, then breaking it down into previous/next week.

```
(use-package quantified :ensure nil :load-path "~/proj/quantified/lisp"
  :preface (load-file "~/proj/quantified/quantified-autoloads.el")
  :unless sacha-phone-p)
```

```
(defvar sach-weekly-review-line-regexp
  "^ \\([[:^:]]+\\): +\\(Sched[[:^:]]+: +\\)\\)?TODO \\(\\.\\.*?\\)\\(\\(?:[
    ↵ ]+\\(\\(:[[:alnum:]]_@#%:[[:^:]]+\\:\\)\\)\\)?[
      ↵ ]*$"
  "Regular expression matching lines to include.")

(defun sach-weekly-done-line-regexp
  "^ \\([[:^:]]+\\): +\\.\\.*?\\(\\(?:Closed\\|Closed\\):\\.\\.*?\\(TODO\\|DONE\\) \\(\\.\\.*?\\)\\(\\(?:[
    ↵ ]+\\(\\(:[[:alnum:]]_@#%:[[:^:]]+\\:\\)\\)\\)?[
      ↵ ]*$"
  "Regular expression matching lines to include as completed tasks.")

;; ###autoload

(defun sach-quantified-sum (start end cat)
  "Return the number of hours from START to END in CAT."
  (quantified-parse-json
   (quantified-request
    (concat "records.json?start=" (or start "") "&end=" (or end "")
    ↵
    ⇨ "ℴ=newest&display_type=time&split=keep&category=" (url-hexify-string cat))
    (list (cons 'auth_token (quantified-token))) "GET"))))

;; ###autoload

(defun sach-quantified-average-weekly (start end category &optional insert)
  "Calculate average hours per week from START to END for CATEGORY."
  (interactive (list (org-read-date nil nil nil "Start: ")
                     (org-read-date nil nil nil "End: ")
                     (sach-quantified-read-category)
                     current-prefix-arg))

  (let ((hours
        (/ (* 7.0 (sach-quantified-sum start end category))
           (days-between end start))))
    (when (called-interactively-p 'any)
      (if insert
         (insert "%.1f hours" hours)
         (message "%.1f hours" hours)))
    hours))

(defun sach-quantified-categories nil)

;; ###autoload

(defun sach-quantified-read-category ()
  (setq sach-quantified-categories
        (or sach-quantified-categories
            (quantified-parse-json
             (quantified-request ↵
              ↵
              ⇨ "/record_categories.json?all=1"
                ↵
                ⇨ (list (cons 'auth_token (quantified-token)))
                  ↵
```

```

    ↪                                     "GET"))))
    (completing-read
      "Category: "
      (mapcar (lambda (o)
                (cons
                  (alist-get 'full_name o)
                  o))
              sachas-quantified-categories)))

;;;###autoload
(defun sachas-quantified-sum (start end cat)
  "Return the number of hours from START to END in CAT."
  (interactive (list (org-read-date nil nil nil "Start: ")
                    ↪
                    (org-read-date nil nil nil "End: ")
                    ↪
                    (sachas-quantified-read-category)))
  (let* ((records
          (quantified-parse-json
            (quantified-request
              (concat "records.json?start=" (or start "") ↪
              "end=" (or end ""))
            ↪
            "order=newest&display_type=time&filter_string=" (url-hexify-string cat))
          (list (cons 'auth_token (quantified-token))) ↪
          "GET"))
         (duration (apply '+ (delq nil (mapcar (lambda (o) (alist-get ↪
          'duration o 0)) records))))
         (hours (/ duration 3600.0)))
    (when (called-interactively-p 'any)
      (message "%s: %.1f hour(s) in %d entries" cat hours (length records)))
    hours))

;;;###autoload
(defun sachas-quantified-get-hours (category time-summary)
  "Return the number of hours based on the time summary."
  (if (stringp category)
      (if (assoc category time-summary) (/ (cdr (assoc category time-summary)) 3600.0) 0)
      (apply '+ (mapcar (lambda (x) (sachas-quantified-get-hours x time-summary)) category))))

```

```

;;;###autoload
(defun sachas-extract-tasks-from-agenda (string matchers prefix line-re)
  (with-temp-buffer
    (insert string)
    (goto-char (point-min))
    (while (re-search-forward line-re nil t)
      (let ((temp-list matchers))
        (while temp-list
          (if (save-match-data
                (string-match (car (car temp-list)) (match-string 1)))
              (progn
                (add-to-list (cdr (car temp-list)) (concat prefix (match-string 3)) t)
                (setq temp-list nil)))
              (setq temp-list (cdr temp-list))))))
  (with-eval-after-load 'ert
    (ert-deftest sachas-extract-tasks-from-agenda ()
      (let (list-a list-b (line-re "\\([^\n:]+\\):\\( \\)\\(\\.\\.*\\)"))

```

```

(sacha-extract-tasks-from-agenda
 "listA: Task 1\nother: Task 2\nlistA: Task 3"
 '("listA" . list-a)
  ("." . list-b))
"- [ ] "
line-re)
(should (equal list-a '("- [ ] Task 1" "- [ ] Task 3")))
(should (equal list-b '("- [ ] Task 2")))))

;;;###autoload
(defun sacha-get-upcoming-tasks ()
  (save-window-excursion
    (org-agenda nil "W")
    (sacha-extract-tasks-from-agenda (buffer-string)
      '(("routines" . ignore)
        ("business" . business-next)
        ("people" . relationships-next)
        ("tasks" . emacs-next)
        ("." . life-next))
      "- [ ] "
      sacha-weekly-review-line-regexp)))

;;;###autoload
(defun sacha-get-previous-tasks ()
  (let (string)
    (save-window-excursion
      (org-agenda nil "W")
      (org-agenda-later -1)
      (org-agenda-log-mode 16)
      (setq string (buffer-string))
      ;; Get any completed tasks from the current week as well
      (org-agenda-later 1)
      (org-agenda-log-mode 16)
      (setq string (concat string "\n" (buffer-string)))
      (sacha-extract-tasks-from-agenda string
        '(("routines" . ignore)
          ("business" . business)
          ("people" . relationships)
          ("tasks" . emacs)
          ("." . life))
        "- [X] "
        sacha-weekly-done-line-regexp))))

;;;###autoload
(defun sacha-org-summarize-focus-areas (date)
  "Summarize previous and upcoming tasks as a list."
  (interactive (list (org-read-date-analyze (if current-prefix-arg (org-read-date) "-fri") nil
    ↪ '(0 0 0))))
  (let (business relationships life business-next relationships-next life-next string emacs ↪
    ↪ emacs-next
        start end time-summary biz-time ignore base-date)
    (setq base-date (apply 'encode-time date))
    (setq start (format-time-string "%Y-%m-%d" (days-to-time (- (time-to-number-of-days ↪
    ↪ base-date) 6))))
    (setq end (format-time-string "%Y-%m-%d" (days-to-time (1+ (time-to-number-of-days ↪
    ↪ base-date)))))
    (setq time-summary (quantified-summarize-time start end))
    (setq biz-time (sacha-quantified-get-hours "Business" time-summary))
    (sacha-get-upcoming-tasks)
    (sacha-get-previous-tasks)

```

```

(setq string
  (concat
    (format "- *A- (Childcare)* (%.1fh - %d%% of total)\n"
      (sacha-quantified-get-hours '("A-") time-summary)
      (/ (sacha-quantified-get-hours '("A-") time-summary) 1.68))
    (format "- *Business* (%.1fh - %d%%)\n" biz-time (/ biz-time 1.68))
    (mapconcat 'identity business "\n") "\n"
    (mapconcat 'identity business-next "\n")
    "\n"
    (format " - *Earn* (%.1fh - %d%% of Business)\n"
      (sacha-quantified-get-hours "Business - Earn" time-summary)
      (/ (sacha-quantified-get-hours "Business - Earn" time-summary) (* 0.01 ←
↪ biz-time)))
    (format " - *Build* (%.1fh - %d%% of Business)\n"
      (sacha-quantified-get-hours "Business - Build" time-summary)
      (/ (sacha-quantified-get-hours "Business - Build" time-summary) (* 0.01 ←
↪ biz-time)))
    (format " - *Connect* (%.1fh - %d%% of Business)\n"
      (sacha-quantified-get-hours "Business - Connect" time-summary)
      (/ (sacha-quantified-get-hours "Business - Connect" time-summary) (* 0.01 ←
↪ biz-time)))
    (format "- *Relationships* (%.1fh - %d%%)\n"
      (sacha-quantified-get-hours '("Discretionary - Social"
                                   "Discretionary - Family") time-summary)
      (/ (sacha-quantified-get-hours '("Discretionary - Social"
                                   "Discretionary - Family") time-summary) 1.68))
    (mapconcat 'identity relationships "\n") "\n"
    (mapconcat 'identity relationships-next "\n") "\n"
    "\n"
    (format "- *Discretionary - Productive* (%.1fh - %d%%)\n"
      (sacha-quantified-get-hours "Discretionary - Productive" time-summary)
      (/ (sacha-quantified-get-hours "Discretionary - Productive" time-summary) ←
↪ 1.68))
    (format " - *Drawing* (%.1fh)\n"
      (sacha-quantified-get-hours '("Discretionary - Productive - Drawing") ←
↪ time-summary))
    (format " - *Emacs* (%.1fh)\n"
      (sacha-quantified-get-hours "Discretionary - Productive - Emacs" ←
↪ time-summary))
    (mapconcat 'identity emacs "\n") "\n"
    (mapconcat 'identity emacs-next "\n") "\n"
    (format " - *Coding* (%.1fh)\n"
      (sacha-quantified-get-hours "Discretionary - Productive - Coding" ←
↪ time-summary))
    (mapconcat 'identity life "\n") "\n"
    (mapconcat 'identity life-next "\n") "\n"
    (format " - *Sewing* (%.1fh)\n"
      (sacha-quantified-get-hours "Discretionary - Productive - Sewing" ←
↪ time-summary))
    (format " - *Writing* (%.1fh)\n"
      (sacha-quantified-get-hours "Discretionary - Productive - Writing" ←
↪ time-summary))
    (format "- *Discretionary - Play* (%.1fh - %d%%)\n"
      (sacha-quantified-get-hours "Discretionary - Play" time-summary)
      (/ (sacha-quantified-get-hours "Discretionary - Play" time-summary) 1.68))
    (format "- *Personal routines* (%.1fh - %d%%)\n"
      (sacha-quantified-get-hours "Personal" time-summary)
      (/ (sacha-quantified-get-hours "Personal" time-summary) 1.68))
    (format "- *Unpaid work* (%.1fh - %d%%)\n"

```



```
(sacha-quantified-get-hours "Unpaid work" time-summary)
(/ (sacha-quantified-get-hours "Unpaid work" time-summary) 1.68))
(format "- *Sleep* (%.1fh - %d% - average of %.1f per day)\n"
(sacha-quantified-get-hours "Sleep" time-summary)
(/ (sacha-quantified-get-hours "Sleep" time-summary) 1.68)
(/ (sacha-quantified-get-hours "Sleep" time-summary) 7)
)))
(if (called-interactively-p 'any)
  (insert string)
  string)))
```

I use this to put together a quick summary of how I spent my time.

The following code makes it easy to add a line:

The following code makes it easy to add a line:

```
;;###autoload
(defun sachar-org-add-line-item-task (task)
  (interactive "MTask: ")
  (org-insert-heading)
  (insert "[ ] " task)
  (let ((org-capture-entry '("t" "Tasks" entry
                             (file+headline "~/sync/orgzly/organizer.org" "Tasks")
                             "")))
    (org-capture nil "t")
    (insert "TODO " task "\nSCHEDULED: <" (org-read-date) ">")))
  ;(define-key org-mode-map (kbd "C-c t") ←
  ↪ 'sachar-org-add-line-item-task)

;;###autoload
(defun sachar-org-list-from-rss (url from-date &optional to-date)
  "Convert URL to an Org list"
  (with-current-buffer (url-retrieve-synchronously url)
    (goto-char (point-min))
    (re-search-forward "<\\/?xml")
    (goto-char (match-beginning 0))
    (let* ((feed (xml-parse-region (point) (point-max)))
           (is-rss (> (length (xml-get-children (car feed) 'entry)) 0)))
      (mapconcat (lambda (link)
                    (format "- %s\n"
                            (org-link-make-string (car link) (cdr link))))
                  (if is-rss
                      (mapcar
                       (lambda (entry)
                         (cons
                          (xml-get-attribute (car
                                                (seq-filter (lambda (x) (string= ↪
                                                                (xml-get-attribute x 'rel) "alternate"))
                                                            (xml-get-children entry 'link))
                          (xml-get-children entry 'link))) 'href)
                        (elt (car (xml-get-children entry 'title)) 2)))
                      (-filter (lambda (entry)
                                (let ((entry-date (elt (car (xml-get-children entry ↪
                                                            (and
                                                             (org-string≤ from-date entry-date)
                                                             (or (null to-date) (string< entry-date to-date))))
                                  (xml-get-children (car feed) 'entry)))
                              (mapcar (lambda (entry)
```

```

      (cons
        (caddr (car (xml-get-children entry 'link)))
        (caddr (car (xml-get-children entry 'title'))))
      (-filter (lambda (entry)
        (let ((entry-time (format-time-string "%Y-%m-%d"
          (date-to-time (elt (car (xml-get-children entry 'pubDate)) 2))
          t)))
          (and
            (not (string< entry-time from-date))
            (or (null to-date) (string< entry-time to-date)))))
        (xml-get-children (car (xml-get-children (car feed) (elt (car (xml-get-children entry 'channel)) 'item))))
        ""))))

```

Now we put it all together...

```

;; ##autoload
(defun sachal-org-prepare-weekly-review (&optional date skip-urls)
  "Prepare weekly review template."
  (interactive (list (org-read-date nil nil nil "Ending on Sun: " nil "-sun")))
  (let* ((post-date (current-time))
        (base-date (apply 'encode-time (org-read-date-analyze date nil '(0 0 0))))
        start end links prev
        (title (format-time-string "Weekly review: Week ending %B %e, %Y" base-date))
        (post-location (concat (format-time-string "%Y/%m/" post-date) (sachal-make-slug (elt (car (xml-get-children entry 'pubDate)) 2))
        (title))))
    (setq start (format-time-string "%Y-%m-%d 0:00" (days-to-time (- (time-to-number-of-days (date-to-time (current-time-zone))
    (base-date) 6)) (current-time-zone)))
    (setq end (format-time-string "%Y-%m-%d 0:00" (days-to-time (1+ (time-to-number-of-days (date-to-time (current-time-zone))
    (base-date) 6)) (current-time-zone)))
    (setq prev (format-time-string "%Y-%m-%d 0:00" (days-to-time (- (time-to-number-of-days (date-to-time (current-time-zone))
    (base-date) 7 6)) (current-time-zone)))
    (outline-next-heading)
    (insert
      "** " title " :weekly:\n"
      (format
        ":PROPERTIES:
:EXPORT_DATE: %s
:EXPORT_ELEVENTY_PERMALINK: %s
:EXPORT_ELEVENTY_FILE_NAME: %s
:END:\n"
        (format-time-string "%Y-%m-%dT%Tz")
        (concat "/blog/" post-location "/")
        (concat "blog/" post-location))
      (sachal-org-summarize-journal-csv start end nil sachal-journal-category-map (elt (car (xml-get-children entry 'pubDate)) 2))
      "\n\nBlog posts*\n\n"
      (sachal-org-list-from-rss "https://sachachua.com/blog/feed" start end)
      "\n\nSketches*\n\n"
      (sachal-sketches-export-and-extract start end) "\n"
      "\n\nTweets*\n\n"
      (sachal-mastodon-format-sachal-tweets-since start)
      "\n\n#+begin_my_details Time\n"
      (format "#+begin_src emacs-lisp :results table :exports results
(sachal-quantified-compare \"%s\" \"%s\" \"%s\" \"%s\" sachal-quantified-summary-categories (elt (car (xml-get-children entry 'pubDate)) 2))
  => \"The other week %\" \"Last week %\")
#+end_src

```

```

:results:\n" prev start start end)
  (orgtbl-to-orgtbl
    (sacha-quantified-compare prev start start end sacha-quantified-summary-categories ↵
    ↵ "The other week %" "Last week %")
    nil)
    "end:\n\n"
    (format "\n#+begin_src emacs-lisp :exports results :results file ↵
    ↵ :file time-graph.svg :output-dir /tmp\n(quantified-svg-to-text (quantified-svg-days ↵
    ↵ \"%s\" \"%s\")\n#+end_src\n\n" start end)
    "\n#+end_my_details\n\n"))

;;###autoload
(defun sacha-prepare-missing-weekly-reviews ()
  "Prepare missing weekly reviews based on LAST_REVIEW property."
  (interactive)
  (let ((today (substring (org-read-date nil nil ".") 0 10))
        (date (org-entry-get (point) "LAST_REVIEW"))))
    (while (string< date today)
      (setq date (substring (org-read-date nil nil "+1w" nil (org-time-string-to-time ↵
      ↵ date)) 0 10))
      (unless (string< today date)
        (save-excursion
          (sacha-org-prepare-weekly-review date))
        (org-entry-put (point) "LAST_REVIEW" date))))))

```

Sketch list

```

;;###autoload
(defun sacha-sketches-export-and-extract (start end &optional do-insert update-db filter)
  "Create a list of links to sketches."
  (interactive (list (org-read-date) (org-read-date) t current-prefix-arg (read-string ↵
  ↵ "Filter: ")))
  (let ((value
        (mapconcat
          (lambda (filename)
            (let ((base (file-name-nondirectory filename)))
              (format "- %s\n"
                (org-link-make-string
                  (replace-regexp-in-string "#" "%23"
                    (concat "sketch:" base))
                  base))))
          (let ((sacha-sketch-directories '("~/sync/sketches")))) ↵
          ↵ (sacha-get-sketch-filenames-between-dates start end filter))
          "")))
    (if do-insert
      (insert value)
      value)))

```

Link-related convenience functions

```

;;###autoload
(defun kensanata/resolve-redirect (url)
  "Resolve shortened URL by launching `curl --head' and parsing the result."
  (let* ((curl (shell-command-to-string
    (format "curl --silent --head %s" url)))
        (location (when (and (string-match "^HTTP/1\\.1 301" curl)
          (string-match "^Location: \\(.*\\)" curl))
          (match-string 1 curl))))))

```

```

(or location url)))

;;;###autoload
(defun sacha-resolve-urls-in-region (beg end)
  "Expand URLs between BEG and END."
  (interactive "r")
  (save-excursion
    (save-restriction
      (narrow-to-region beg end)
      (goto-char (point-min))
      (while (re-search-forward org-link-bracket-re nil t)
        (replace-match (save-match-data (kensanata/resolve-redirect
                                          (match-string 1))) t t nil 1))
      (goto-char (point-min))
      (while (re-search-forward org-link-re-with-space nil t)
        (replace-match (save-match-data (kensanata/resolve-redirect
                                          (match-string 0))) t t nil))))))

;;;###autoload
(defun sacha-open-urls-in-region (beg end)
  "Open URLs between BEG and END.
   TODO: Get better at detecting and opening all URLs"
  (interactive "r")
  (save-excursion
    (save-restriction
      (narrow-to-region beg end)
      (goto-char (point-min))
      (while (re-search-forward org-link-any-re nil t)
        (save-excursion
          (backward-char)
          (let ((url (match-string 0)))
            (unless (string-match "permalink.gmane.org" url)
              (browse-url url))))))))

```

```

(add-to-list 'browse-url-handlers '("https?://yhetil.org/.*/raw" . sacha-browse-yhetil))
;;;###autoload
(defun sacha-browse-yhetil (url &rest _)
  (when (string-match "\\(https?://yhetil.org/.*/\\)/raw$" url)
    (funcall browse-url-browser-function (match-string 1 url))))

```

8.21.2 Monthly reviews

I want to be able to see what I worked on in a month so that I can write my [monthly reviews](#). This code makes it easy to display a month's clocked tasks and time. I haven't been particularly thorough in tracking time before, but now that I have a shortcut that logs in Quantified Awesome as well as in Org, I should end up clocking more.

```

;;;###autoload
(defun sacha-org-review-month (start-date)
  "Review the month's clocked tasks and time."
  (interactive (list (org-read-date)))
  ;; Set to the beginning of the month
  (setq start-date (concat (substring start-date 0 8) "01"))
  (let ((org-agenda-show-log t)
        (org-agenda-start-with-log-mode t)
        (org-agenda-start-with-clockreport-mode t)
        (org-agenda-clockreport-parameter-plist '(:link t :maxlevel 3)))

```

```
(org-agenda-list nil start-date 'month)))
```

Here's a function like `sacha-org-prepare-weekly-review`:

```
;; ###autoload
(defun sacha-list-blog-posts (start-date end-date)
  (seq-filter (lambda (o)
    (and (or (null start-date) (string< start-date (plist-get o :date)))
      (or (null end-date) (string< (plist-get o :date) end-date))))
    (let ((json-object-type 'plist))
      (json-read-file "~/proj/static-blog/_site/blog/all/index.json")))))

;; ###autoload
(defun sacha-org-get-last-week ()
  "Return dates for filtering last week."
  (if (string= (format-time-string "%u") "6") ;; my week starts on Saturday
    (cons (org-read-date nil nil "-1w") (org-read-date nil nil "."))
    (cons (org-read-date nil nil "-2sat") (org-read-date nil nil "-sat"))))
  ;; ###autoload
(defun sacha-org-get-month (&optional date-string)
  "Return start of month containing DATE and start of following month.
  Result is (START . NEXT)."
  (let* ((date (decode-time (if (stringp date-string) (org-read-date nil t date-string)
    ↪ date-string)))
    (month (elt date 4))
    (year (elt date 5))
    start-date
    end-date)
    (calendar-increment-month month year 1)
    (cons
      (format "%4d-%02d-01" (elt date 5) (elt date 4))
      (format "%4d-%02d-01" year month))))

(defvar sacha-quantified-summary-categories '("Business" "Discretionary - Play" "Unpaid work" ↪
  ↪ "A+" "Discretionary - Family" "Sleep" "Discretionary - Productive" "Personal"))
;; ###autoload
(defun sacha-quantified-summarize-time-table-month (month)
  "Insert or return the table summarizing the month's time, compared with the previous ↪
  ↪ month."
  (interactive (list (org-read-date nil t)))
  (let* ((date (decode-time (if (stringp month) (date-to-time month) month)))
    (month (elt date 4))
    (year (elt date 5))
    start-date
    end-date
    previous-date
    results)
    (calendar-increment-month month year -1)
    (setq start-date (format "%4d-%02d-01 0:00" year month)
      end-date (format "%4d-%02d-01 0:00" (elt date 5) (elt date 4)))
    (calendar-increment-month month year -1)
    (setq previous-date (format "%4d-%02d-01 0:00" year month))
    (setq results (orgtbl-to-orgtbl (sacha-quantified-compare previous-date ↪
  ↪ start-date start-date end-date sacha-quantified-summary-categories "Previous month %" ↪
  ↪ "This month %")
    ↪
    (when (called-interactively-p 'any)
```

```

      (insert results))
    results))

```

```

;;;###autoload
(defun sach-org-prepare-monthly-review (time)
  (interactive (list (org-read-date nil t)))
  (let* ((date (decode-time time))
         (month (elt date 4))
         (year (elt date 5))
         (post-date (current-time))
         post-location
         title
         start-date
         end-date
         previous-date
         posts
         sketches
         time-comparison
         org-date)
    (calendar-increment-month month year -1)
    (setq start-date (format "%4d-%02d-01 0:00" year month)
          end-date (format "%4d-%02d-01 0:00" (elt date 5) (elt date 4))
          title (format-time-string "Monthly review: %B %Y" (encode-time 0 0 0 1 month year))
          post-location (concat (format-time-string "%Y/%m/" post-date) (sacha-make-slug title))
          posts (mapconcat (lambda (o) (concat "- " (org-link-make-string (concat "↵"
↵ "https://sachachua.com" (plist-get o :permalink))
                                                                    (plist-get o :title))))
                           (sacha-list-blog-posts
                            (substring start-date 0 10)
                            (substring end-date 0 10))
                            "\n")
          sketches (sacha-sketches-export-and-extract (substring start-date 0 10) (substring ↵
↵ end-date 0 10) nil t))
    (calendar-increment-month month year -1)
    (setq previous-date (format "%4d-%02d-01 0:00" year month))
    (setq time-comparison (sacha-quantified-compare previous-date start-date start-date ↵
↵ end-date sach-quantified-summary-categories "Previous month %" "This month %"))
    (goto-char (line-end-position))
    (insert
     "\n\n** " title " :monthly:review:\n"
     "*Blog posts*\n"
     posts "\n\n"
     "*Sketches*\n\n"
     sketches
     (format "*Time*\n\n#+begin_src emacs-lisp :results table :exports ↵
↵ results\n(sacha-quantified-compare \"%s\" \"%s\" \"%s\" \"%s\" ↵
↵ sach-quantified-summary-categories \"Previous month %\" \"This month ↵
↵ %\" )\n#+end_src\n\n"
             previous-date start-date start-date end-date)
     (orgtbl-to-orgtbl time-comparison nil)
     (format "\n\n#+begin_src emacs-lisp :exports results :results file :file ↵
↵ monthly-%s.svg :output-dir /tmp\n(quantified-svg-to-text (quantified-svg-days \"%s\" ↵
↵ \"%s\" 'horizontal))\n#+end_src\n\n"
             start-date
             start-date end-date))
    (sacha-org-11ty-prepare-subtree)))
;;;###autoload

```

```

(defun sacha-org-prepare-yearly-review (year-end)
  (interactive (list (org-read-date nil t nil "Year end (exclusive): ")
    (let* ((date (decode-time year-end))
      (month (elt date 4))
      (year (elt date 5))
        (end-date (format-time-string "%Y-%m-%d" year-end))
        (start-date (progn
          (setf (elt date 5) (1- (elt date 5)))
          (format-time-string "%Y-%m-%d" (encode-time date)))
          (previous-date (progn
            (setf (elt date 5) (1- (elt date 5)))
            (format-time-string "%Y-%m-%d" (encode-time date)))
            (posts (mapconcat (lambda (o)
              (concat "- " (org-link-make-string
                (concat sacha-blog-base-url (plist-get o :permalink))
                (plist-get o :title))))
                (sacha-list-blog-posts
                  (substring start-date 0 10)
                  (substring end-date 0 10)
                  "\n"))
                (sketches (sacha-sketches-export-and-extract
                  start-date 0 10) (substring end-date 0 10) nil t))
                (time (sacha-quantified-compare
                  start-date end-date sacha-quantified-summary-categories
                    "The other year %"
                    "Last year %"))
                  (insert
                    "*Blog posts*\n\n" posts "\n\n"
                    "*Sketches*\n\n" sketches
                    "*Time*\n\n"
                    (format "#+begin_src emacs-lisp :results table :exports results
(sacha-quantified-compare \"%s\" \"%s\" \"%s\" \"%s\" sacha-quantified-summary-categories
  ↳ \"The other year %\" \"Last year %\")
#+end_src

:results:\n" previous-date start-date start-date end-date)
                    (orgtbl-to-orgtbl time nil)
                    (format "\n#+begin_src emacs-lisp :exports results :results file :file
↳ time-graph.svg :output-dir /tmp\n(quantified-svg-to-text (quantified-svg-days \"%s\"
↳ \"%s\" 'horizontal))\n#+end_src\n\n" start-date end-date))))

```

8.21.3 Emoji summaries

```

;; ##autoload
(defun sacha-org-emoji-summary (&optional label)
  (let (results)
    (save-excursion
      (goto-char (org-find-property "EXPORT_ELEVENTY_PERMALINK" ↵
      ↵ (org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK")))
      (let ((end (save-excursion (org-end-of-subtree))))
        (while (re-search-forward "^\\([0-9]+\\)\\. ↵
        ↵ \\([A-Za-z0-9]+\\) \\([.+?\\)\\)\\(- weekly highlight\\)?\\n" end t)
          (let ((day (match-string 1))
                (icon (match-string 2))
                (text (match-string 3)))

            (push
              (if (string-match org-link-bracket-re text)
                  (format "<a href=\"%s\" ↵
                  ↵ title=\"%s - %s\">%s</a>"
                  ↵
                  (match-string 1 text)
                  ↵
                  (match-string 2 text)
                  ↵
                  day
                  ↵
                  icon)
                  (format "<span title=\"%s - ↵
                  ↵ %s\">%s</span>"
                  ↵
                  text
                  day
                  icon)))
              results))))))
    (format "<div class=\"%emoji-summary\">%s</div>"
      (if label (concat label ": ") "")
      (string-join (nreverse results) " ")))

```



```
(org-paste-subtree nil nil nil t))))
```

8.22.2 TODO Bounce to another file

computer:phone

On my phone, Emacs in Termux is nice for scripting, and Orgzly is nice for editing long text. Let's see if this function lets me quickly bounce things around from one place to another.

```
;;;###autoload
(defun sacha-org-bounce-to-file (file)
  "Toggle subtree between its home file and another file.
Limitations: Reinserts entry at bottom of subtree, uses kill ring."
  (interactive (list (read-file-name "File: ")))
  (if (string= (buffer-file-name) (expand-file-name file))
      ;; Return it
      (let ((location (org-entry-get (point) "BOUNCE")))
        (when location
          (setq location (read location))
          (org-cut-subtree)
          (save-buffer)
          (with-current-buffer (find-file (car location))
            (save-restriction
              (widen)
              (goto-char (org-find-olp location))
              (org-end-of-subtree)
              (unless (bolp) (insert "\n"))
              (org-paste-subtree (length location) nil nil t)
              (save-buffer))))))
      (org-entry-put (point) "BOUNCE" (prin1-to-string (cons (buffer-file-name)
                                                              ↔
                                                              ↵ (org-get-outline-path))))
      (org-cut-subtree)
      (save-buffer)
      (with-current-buffer (find-file file)
        (save-restriction
          (widen)
          (goto-char (point-max))
          (unless (bolp) (insert "\n"))
          (org-yank)
          (save-buffer)))))
```

8.22.3 Basic refiling configuration

`org-refile` lets you organize notes by typing in the headline to file them under.

```
(setq org-reverse-note-order t) ; I want new notes prepended
(setq org-refile-use-outline-path 'title) ; distinguish between files named the same
(setq org-outline-path-complete-in-steps nil)
(setq org-refile-allow-creating-parent-nodes 'confirm)
(setq org-refile-use-cache t)
(setq org-blank-before-new-entry nil)

(setq org-refile-targets
  '(((("~/sync/orgzly/organizer.org"
        "~/sync/orgzly/routines.org"
        "~/sync/orgzly/business.org"
        "~/sync/orgzly/reference.org"
        "~/sync/orgzly/garden.org"
```

```

~/sync/orgzly/decisions.org"
~/sync/emacs/Sacha.org"
~/sync/orgzly/posts.org"
~/sync/orgzly/people.org"
~/sync/orgzly/resources.org"
~/sync/orgzly/Inbox.org"
↵
↪ "~/proj/emacsconf/wiki/2023/organizers-notebook/index.org")
  . (:maxlevel . 7))
  (( "~/proj/quantified/notes.org"
    "~/proj/sketches/notes.org"
    "~/sync/static-blog/notes.org"
    "~/proj/journal/notes.org"
    "~/sync/orgzly/crafts.org"
    "~/sync/orgzly/misc.org")
    . (:maxlevel . 2))
  (( "~/sync/orgzly/news.org")
    . (:maxlevel . 1))
  ))

```

TEACH Jump to Org location by substring

```

;; Example: (org-refile 4 nil (sacha-org-refile-get-location-by-substring "Other Emacs"))
;; ###autoload
(defun sacha-org-refile-get-location-by-substring (regexp &optional file)
  "Return the refile location identified by REGEXP."
  (let ((org-refile-targets org-refile-targets) tbl)
    (setq org-refile-target-table (org-refile-get-targets)))
  (unless org-refile-target-table
    (user-error "No refile targets"))
  (cl-find regexp org-refile-target-table
    :test
    (lambda (a b)
      (and
        (string-match a (car b))
        (or (null file)
          (string-match file (elt b 1)))))))
;; ###autoload
(defun sacha-org-refile-subtree-to (name)
  (org-refile nil nil (sacha-org-refile-get-location-exact name)))
;; ###autoload
(defun sacha-org-refile-get-location-exact (name &optional file)
  "Return the refile location identified by NAME."
  (let ((org-refile-targets org-refile-targets) tbl)
    (setq org-refile-target-table (org-refile-get-targets)))
  (unless org-refile-target-table
    (user-error "No refile targets"))
  (cl-find name org-refile-target-table
    :test (lambda (a b)
      (and (string-equal a (car b))
        (or (null file)
          (string-match file (elt b 1)))))))
;; Example: (sacha-org-clock-in-refile "Off my computer")
;; ###autoload
(defun sacha-org-clock-in-refile (location &optional file)
  "Clocks into LOCATION.
  LOCATION and FILE can also be regular expressions for ↵
  ↪ `sacha-org-refile-get-location-by-substring'."

```

```

(interactive (list (sacha-org-refile-get-location)))
(save-window-excursion
  (save-excursion
    (if (stringp location) (setq location (sacha-org-refile-get-location-by-substring ←
    ↪ location file))))
    (org-refile 4 nil location)
    (org-clock-in))))

;;###autoload
(defun sacha-org-finish-previous-task-and-clock-in-new-one (location &optional file)
  (interactive (list (sacha-org-refile-get-location)))
  (save-window-excursion
    (org-clock-goto)
    (org-todo 'done))
  (sacha-org-clock-in-and-track-by-name location file))

;;###autoload
(defun sacha-org-clock-in-and-track-by-name (location &optional file)
  (interactive (list (sacha-org-refile-get-location)))
  (save-window-excursion
    (save-excursion
      (if (stringp location) (setq location (sacha-org-refile-get-location-exact location file))
      (org-refile 4 nil location)
      (sacha-org-clock-in-and-track))))
  )

;;###autoload
(defun sacha-org-off-sacha-computer (category)
  (interactive "MCategory: ")
  (eval-when-compile (require 'quantified nil t))
  (sacha-org-clock-in-refile "Off my computer")
  (quantified-track category))

```

Quick way to jump

```

;;###autoload
(defun sacha-org-jump ()
  (interactive)
  (let ((current-prefix-arg '(4)))
    (call-interactively 'org-refile)))

```

8.22.4 TODO Refile inbox entries to a smaller set of org-refile-targets

dotemacs

When I'm filing things from my inbox, I want a faster refile that considers a smaller set of entries.

```

;;###autoload
(defun sacha-org-refile-to-subset (arg)
  "Refile to a smaller set of targets."
  (interactive "P")
  (let ((org-refile-targets '("~/sync/orgzly/organizer.org" . (:tag . "inboxtarget"))
    ↪
    ↪ (~~/sync/orgzly/organizer.org" . (:maxlevel . 3))
    ↪
    ↪ (~~/sync/orgzly/resources.org" . (:maxlevel . 1))
    ↪
    ↪ (nil . (:level . 1))
    ↪
    ↪ (~~/proj/stream/index.org" . (:maxlevel . 3))
    ↪
    ↪ (~~/sync/emacs/Inbox.org" . (:maxlevel . 1))

```

```

↩                                     (~/sync/emacs/Sacha.org" . (:maxlevel . 4))
↩                                     (~/sync/orgzly/people.org" . (:maxlevel . 2))))
      (org-refile arg)))

;;;###autoload
(defun sach-org-refile-to-target-or-subset (&optional arg)
  (interactive "P")
  (or (sach-org-refile-current-entry-to-tag-target)
      (sach-org-refile-to-subset arg)))

```

```
(keymap-global-set "C-c w" 'sach-org-refile-to-target-or-subset)
```

Next steps might include filtering out private stuff, but I don't think I'll use this while streaming, so it might be okay for now.

8.22.5 **DONE** Automatically refiling Org Mode headings based on tags org:emacs

I have lots of different things in my Org Mode inbox. Following the [PARA](#) method, I want to file them under projects, areas, resources, or archive so that I can find related things later. Actually, no, I don't want to refile them. I do want to be able to:

- find all the pieces related to something when I'm ready to start working on a task
- find useful links again, especially if I can use my own words

Refiling is annoying on my phone, so I tend to wait until I'm back at my computer. But even with `org-refile-use-outline-path` set to `file` and the ability to specify substrings, there's still a bit of friction.

Tagging is a little easier to do on my phone. I can add a few tags when I share a webpage or create a task.

I thought it would be nice to have something that automatically refiles my inbox headings tagged with various tags to other subtrees where I've set a `:TAG_TARGET:` property or something like that. For example, I can set the `TAG_TARGET` property to `emacsconf` to mean that anything tagged with `:emacsconf:` should get filed under there.

<https://emacs.stackexchange.com/questions/36360/recursively-refiling-all-subtrees-with-tag-to-a-destination-org-mode...>

```
(defcustom sach-org-refile-tag-targets nil
  "Searches and IDs."
  :group 'sach
  :type '(repeat (cons string string string)))

```

```
(setq sach-org-tag-target-files
      (append '("~/sync/orgzly/news.org"
                  "~/sync/orgzly/resources.org"
                  "~/proj/stream/index.org")
              org-agenda-files))

```

```
(defvar sach-org-tag-target-files
  nil
  "Files to check for tag targets.")
;;;###autoload

```

```

(defun sach-org-update-tag-targets ()
  (interactive)
  (let ((org-agenda-files sach-org-tag-target-files))
    (setq sach-org-refile-tag-targets
          (let (list)
            (org-map-entries
             (lambda ()
               (list (concat "+" (org-entry-get (point) "TAG_TARGET"))
                     (org-id-get-create)
                     (org-entry-get (point) "ITEM"))
              "TAG_TARGET={.}" 'agenda))))
      (customize-save-variable 'sach-org-refile-tag-targets sach-org-refile-tag-targets)))

;;###autoload
(defun sach-org-add-tag-target (tag)
  (interactive "MTag: ")
  (org-entry-put (point) "TAG_TARGET" tag)
  (push (list (concat "+" tag)
              (org-id-get-create)
              (org-entry-get (point) "ITEM"))
        sach-org-refile-tag-targets)
  (customize-save-variable 'sach-org-refile-tag-targets sach-org-refile-tag-targets))

;;###autoload
(defun sach-org-refile-current-entry-to-tag-target (&optional arg target-marker)
  (interactive (list current-prefix-arg (cadr (sach-org-tag-target-for-entry-at-point))))
  (unless target-marker
    (setq target-marker (cadr (sach-org-tag-target-for-entry-at-point))))
  (when (stringp target-marker)
    (setq target-marker (org-id-find target-marker t)))
  (when target-marker
    (org-refile
     arg nil
     (with-current-buffer (marker-buffer target-marker)
       (goto-char target-marker)
       (list (org-get-heading)
             (buffer-file-name (marker-buffer target-marker))
             nil
             target-marker)))))

;; Based on ↵
↵ https://emacs.stackexchange.com/questions/36360/recursively-refiling-all-subtrees-with-tag-to-a-destination
;;###autoload
(defun sach-org-refile-matches-to-heading (match target-heading-id &optional scope copy)
  "Refile all headings within SCOPE (per `org-map-entries') to TARGET-HEADING-ID."
  (if-let (target-marker (org-id-find target-heading-id t))
    (let* ((target-rfloc (with-current-buffer (marker-buffer target-marker)
                        (goto-char target-marker)
                        (list (org-get-heading)
                            (buffer-file-name (marker-buffer target-marker))
                            nil
                            target-marker)))
          (headings-to-copy (org-map-entries (lambda () (point-marker)) match scope)))
      (mapc
       (lambda (heading-marker)
         (with-current-buffer (marker-buffer heading-marker)
           (org-refile
            arg nil
            (with-current-buffer (marker-buffer heading-marker)
              (goto-char (org-id-find target-heading-id t))
              (list (org-get-heading)
                    (buffer-file-name (marker-buffer heading-marker))
                    nil
                    target-marker)))))))
    (org-refile
     arg nil
     (with-current-buffer (marker-buffer target-marker)
       (goto-char target-marker)
       (list (org-get-heading)
             (buffer-file-name (marker-buffer target-marker))
             nil
             target-marker)))))

```

```

        (goto-char heading-marker)
        (org-refile nil nil target-rfloc (when copy "Copy"))))
    (nreverse headings-to-copy))
  (message "%s %d headings!"
    (if copy "Copied" "Refiled")
    (length headings-to-copy)))
  (warn "Could not find target heading %S" target-heading-id)))

;;;###autoload
(defun sacha-org-tag-target-for-entry-at-point ()
  "Return the `sacha-org-refile-tag-targets' entry that matches point."
  (let ((tags (org-get-tags (point)))
        (level (org-current-level))
        (todo (org-get-todo-state))
        matcher)
    (catch 'found
      (dolist (target sacha-org-refile-tag-targets)
        (setq matcher (cdr (org-make-tags-matcher (car target)))))
        (when (funcall matcher todo tags level)
          (throw 'found target))))))

;;;###autoload
(defun sacha-org-refile-to-tag-targets ()
  (interactive)
  (dolist (rule sacha-org-refile-tag-targets)
    (sacha-org-refile-matches-to-heading (car rule) (cadr rule))))

;;;###autoload
(defun sacha-org-refile-inbox-to-tag-targets ()
  (interactive)
  (with-current-buffer (find-file-noselect sacha-org-inbox-file)
    (dolist (rule sacha-org-refile-tag-targets)
      (sacha-org-refile-matches-to-heading (car rule) (cadr rule) 'file))))

```

So when I'm ready, I can call `sacha-org-refile-to-tag-targets` and have lots of things disappear from my inbox.

Next step might be to write a function that will refile just the current subtree (either going straight to the tag target or prompting me for a destination if there isn't a matching one), so I can look at stuff, decide if it needs to be scheduled first or something like that, and then send it somewhere. There must be something I can pass a property match to and it'll tell me if it matches the current subtree - probably something along the lines of `org-make-tags-matcher`...

Anyway, just wanted to share this!

8.22.6 Moving lines around

This makes it easier to reorganize lines in my weekly review.

```

;;;###autoload
(defun sacha-org-move-line-to-destination ()
  "Moves the current list item to DESTINATION in the current buffer.
If no DESTINATION is found, move it to the end of the list
and indent it one level."
  (interactive)
  (save-window-excursion
    (save-excursion
      (let ((string

```

```

        (buffer-substring-no-properties
         (line-beginning-position) (line-end-position)))
        (case-fold-search nil)
        found)
      (delete-region (line-beginning-position) (1+ (line-end-position)))
      (save-excursion
        (goto-char (point-min))
        (when (re-search-forward "DESTINATION" nil t)
          (insert "\n" (make-string (- (match-beginning 0) (line-beginning-position)) ?\ ) ←
→ (s-trim string))
          (setq found t)))
        (unless found
          (org-end-of-item-list)
          (insert string "\n")))))

```

```

;;;###autoload
(defun sacha-org-move-line-to-end-of-list ()
  "Move the current list item to the end of the list."
  (interactive)
  (save-excursion
    (let ((string (buffer-substring-no-properties (line-beginning-position)
                                                    (line-end-position))))
      (delete-region (line-beginning-position) (1+ (line-end-position)))
      (org-end-of-item-list)
      (insert string))))

```

8.22.7 Organizing my blog index

```

;;;###autoload
(defun sacha-org-file-blog-index-entries ()
  "Keep filing until I press `C-g'."
  (interactive)
  (while t
    (sacha-org-file-blog-index-entry
     (line-beginning-position) (1+ (line-end-position)))
    (let ((org-refile-targets
           '("~/proj/sharing/blog.org" . (:maxlevel . 3))))
      (save-excursion (org-refile-get-location "Location")))))

;;;###autoload
(defun sacha-org-file-blog-index-entry (beg end location)
  "Copy entries into blog.org."
  (interactive)
  (list
   (if (region-active-p) (point) (line-beginning-position))
   (if (region-active-p) (mark) (1+ (line-end-position)))
   (let ((org-refile-targets
          '("~/proj/sharing/blog.org" . (:maxlevel . 3))))
     (save-excursion (org-refile-get-location "Location")))))
  (let ((s
        (replace-regexp-in-string
         "[ \t]*- \\(\\[X\\] \\)?"
         "- [X] "
         (buffer-substring-no-properties beg end))))
    ;; if we're already in blog.org, delete the previous entry
    (if (string= buffer-file-name (expand-file-name "~/proj/sharing/blog.org"))

```

```

      (delete-region beg end))
    (save-window-excursion
      (save-excursion
        (find-file (nth 1 location))
        (save-excursion
          (save-restriction
            (widen)
            (goto-char (nth 3 location))
            (re-search-forward org-list-full-item-re nil t)
            (goto-char (line-beginning-position))
            (insert s)
            (org-update-statistics-cookies nil)))))))

```

8.22.8 Refiling Org Mode notes to headings in the same file

org:emacs

I spent some time tidying up my [Emacs configuration](#) . I used `org-babel-demarcate-block` to split up some long `#+begin_src ... #+end_src` blocks and refile sections to group them together. I also promoted more sections to top-level headings in order to make the most of the side navigation provided by the [Read the Org setup file](#) based on [Read the Docs](#). These functions were helpful:

```

;; ##autoload
(defun sach-org-refile-in-file (&optional prefix)
  "Refile to a target within the current file."
  (interactive)
  (let ((org-refile-targets (list (cons nil '(:maxlevel . 5)))))
    (call-interactively 'org-refile)))

;; ##autoload
(defun sach-org-refile-to-previous ()
  "Refile subtree to last position from `sach-org-refile-in-file'."
  (interactive)
  (save-selected-window
    (when (eq major-mode 'org-agenda-mode)
      (org-agenda-switch-to))
    (org-cut-subtree)
    (save-window-excursion
      (save-excursion
        (bookmark-jump (plist-get org-bookmark-names-plist :last-refile))
        (let ((level (org-current-level)))
          (org-end-of-subtree t t)
          (org-paste-subtree)))))))

```

```

(with-eval-after-load 'org
  (push '("w" call-interactively 'org-refile) org-speed-commands)
  (push '("W" call-interactively 'sach-org-refile-in-file) org-speed-commands)
  (push '(".") call-interactively 'sach-org-refile-to-previous) org-speed-commands))

```

I usually have `org-refile-targets` set to a long list of files so that I can use `C-u C-c C-w` to jump to headings from anywhere, but I just wanted to refile other things in my configuration, so it was nice to limit the scope to just that file.

I like using `org-speed-commands` to give me quick shortcuts when I'm at headings.

8.23 Contacts


```

        (setq definition (buffer-substring-no-properties (point) (save-excursion ↵
↵ (end-of-defun) (point))))))
    ;; Fallback: Print function definition
    (setq definition (concat (prin1-to-string (symbol-function function)) "\n"))
    (if (org-in-src-block-p)
        (insert definition)
        (insert "#+begin_src emacs-lisp\n" definition "#+end_src\n"))))

;;;###autoload
(defun sach-org-insert-function-and-key (keys)
  (interactive (caar (help--read-key-sequence)))
  (insert (format "%s= (%s=)" (symbol-name (key-binding keys t))
                  (key-description keys))))

```

```

(use-package org
  :hook (org-mode . org-indent-mode)
  :config
  (setq org-indent-indentation-per-level 2)
  (setq org-edit-src-content-indentation 0)
  (setq org-src-preserve-indentation t))

```

8.25 Bookmarks

This lets me easily insert links with completion.

```

(defvar sach-org-bookmark-file "~/sync/orgzly/resources.org")

;;;###autoload
(defun sach-org-bookmarks ()
  "Returns a list of plists like this: (:title ... :url ... ) ... ).
Uses the info from `sach-org-bookmark-file'."
  (delq nil
    (with-current-buffer (find-file-noselect sach-org-bookmark-file)
      (org-map-entries
        (lambda ()
          (let ((title (org-entry-get (point) "ITEM"))
                (url (or (org-entry-get (point) "URL")
                        (progn
                          (let ((end (save-excursion (org-end-of-subtree))))
                            (when (and (< (point) end)
                                      (re-search-forward org-link-any-re end t))
                              (org-element-property
                                :raw-link
                                (org-element-context)))))))
            (when (string-match org-link-bracket-re title)
              (setq title (match-string 2 title)))
            (when url
              (list :title title :url url))))))))))

;;;###autoload
(defun sach-org-bookmark-match (s)
  "Return the first bookmark that matches S."
  (setq s (downcase s))
  (plist-get (seq-find (lambda (bookmark)
                        (string= (downcase (plist-get bookmark :title))
                                s))

```

```

                (sacha-org-bookmarks))
                :url))

(defun sacha-org-bookmarks-for-completion ()
  (mapcar
    (lambda (o)
      (cons (propertize (format "%s - %s"
                                ↪
                                (plist-get o :title)
                                ↪
                                (plist-get o :url))
              ↪
              :title (plist-get o :title)
              :url (plist-get o :url))
            (propertize
              (plist-get o :url)
              :title (plist-get o :title)
              :url (plist-get o :url)
              'title (plist-get o :title)
              'url (plist-get o :url))))
      (sacha-org-bookmarks)))

;;;###autoload
(defun sacha-org-bookmark-complete (&optional initial-text)
  "Complete a bookmark."
  (when (region-active-p)
    (setq initial-text (or initial-text (buffer-substring (region-beginning)
                                                           (region-end)))))
    (let ((bookmarks (sacha-org-bookmarks-for-completion)))
      (assoc-default
        (completing-read
          "Bookmark: " bookmarks nil nil initial-text)
        bookmarks
        #'string=)))

;;;###autoload
(defun sacha-org-bookmark-insert-description (link &optional description)
  "Provide a default description."
  (or description
    (get-text-property 0 'title link)
    (sacha-page-title link)))

;;;###autoload
(defun sacha-org-bookmark-save-link (link title)
  "Save the current link to my resources file."
  (interactive
    (or
      (and (derived-mode-p 'org-mode)
        (let ((elem (org-element-context)))
          (when (eq (org-element-type elem) 'link)
            (list (org-element-property :raw-link elem)
                  (buffer-substring (org-element-contents-begin elem)
                                    (org-element-contents-end elem))))))
      (let* ((url (read-string "URL: "))
              (title (sacha-page-title url)))
        (list url (read-string "Title: " title)))))
    (with-current-buffer (find-file-noselect sacha-org-bookmark-file)
      (goto-char (point-max))

```

```
(unless (bolp) (insert "\n"))
(insert "** " title "\n" link "\n")
(save-buffer)))
```

```
(with-eval-after-load
 'embark
 (keymap-set embark-org-link-map "s" #'sacha-org-bookmark-save-link))

(with-eval-after-load 'org
  (org-link-set-parameters
   "bookmark"
   :complete #'sacha-org-bookmark-complete
   :insert-description #'sacha-org-link-insert-description))
```

8.26 Org Babel

```
(setq org-edit-src-auto-save-idle-delay 5)
```

8.26.1 Make it easier to split my literate config into files

```
;; ##autoload
(defun sacha-emacs-suggest-file ()
  (let ((elem (org-element-context)))
    (seq-find (lambda (o)
      (save-excursion
        (goto-char (org-element-begin elem))
        (re-search-forward
         (concat "^ *(\(\\(cl-\\)?defun *"
          (regexp-quote (file-name-base o)))
         (org-element-end elem) t)))
      (directory-files "lisp/" t "\\\\.el")))))

;; ##autoload
(defun sacha-emacs-split-into-file (filename)
  "Prepare this block for splitting into FILENAME."
  (interactive (list
    (let* ((default (sacha-emacs-suggest-file))
      (relative
        (and default
          (file-relative-name
            default
            (file-name-directory (buffer-file-name))))))
      (if default
        (read-file-name
         (format "File (%s): " relative)
         "lisp/"
         relative)
        (read-file-name "File: " "lisp/")))))
  (let ((elem (org-element-context)))
    (save-restriction
      (narrow-to-region (org-element-begin elem)
        (org-element-end elem))
      (goto-char (point-min))
      (forward-line)
```

```

(while (re-search-forward "^ ←
↳ *((\\(cl-defun\\|defun\\|define-minor-mode\\|define-derived-mode\\)) " nil t)
  (unless (save-match-data
            (save-excursion
              (forward-line -1)
              (looking-at ";;;###autoload\\n"))))
    (replace-match
      (concat ";;;###autoload\\n"
              (match-string 0))))))

(goto-char (point-min))
(forward-line)
(when (re-search-forward ←
↳ "^((\\(setq\\|use-package\\|with-eval-after-load\\|keymap-global-set\\|keymap-set\\|keymap-global-set\\))" ←
↳ nil t)
  (goto-char (match-beginning 0))
  (sacha-org-demarcate-block))
;; Add the tangle
(goto-char (point-min))
(unless (looking-at "#\\|+begin_src")
  (re-search-forward "#\\|+begin_src" nil t))
(unless (save-excursion (re-search-forward ":tangle" nil (line-end-position)))
  (goto-char (line-end-position))
  (insert " :tangle " (file-relative-name filename (file-name-directory ←
↳ (buffer-file-name))))))

```

Now it should be a little bit easier to jump around my source code!

8.26.2 Linking to Org Babel source in a comment, and making that always use file links org

I've been experimenting with these default header args for Org Babel source blocks.

```

(setq org-babel-default-header-args
  '(:session . "none")
    (:results . "drawer replace")
      (:comments . "link") ;; add a link to the original source

    (:exports . "both")
    (:cache . "no")
    (:eval . "never-export") ;; explicitly evaluate blocks instead of evaluating them ←
↳ during export
    (:hlines . "no")
    (:tangle . "no"))) ;; I have to explicitly set up blocks for tangling

```

In particular, `:comments link` adds a comment before each source block with a link to the file it came from. This allows me to quickly jump to the actual definition. It also lets me use `org-babel-detangle` to copy changes back to my Org file.

I also have a custom link type to make it easier to link to sections of my configuration file ([Links to my config](#) (p. 320)). Org Mode prompts for the link type to use when more than one function returns a link for storing, so that was interrupting my tangling with lots of interactive prompts. The following piece of advice ignores all the custom link types when tangling the link reference. That way, the link reference always uses the `file:` link instead of offering my custom link types.

```

(advice-add #'org-babel-tangle--unbracketed-link
  :around (lambda (old-fun &rest args)
    (let ←
↳ (org-link-parameters)

```



(**apply** old-fun args))))



8.26.3 DONE Org Mode: Tangle Emacs config snippets to different files and add boilerplate emacs.org

I want to organize the functions in my Emacs configuration so that they are easier for me to test and so that other people can load them from my repository. Instead of copying multiple code blogs from my blog posts or my exported Emacs configuration, it would be great if people could just include a file from the repository. I don't think people copy that much from my config, but it might still be worth making it easier for people to borrow interesting functions. It would be great to have libraries of functions that people can evaluate without worrying about side effects, and then they can copy or write a shorter piece of code to use those functions.

In Prot's configuration ([The custom libraries of my configuration](#)), he includes each library as in full, in a single code block, with the boilerplate description, keywords, and (`provide '...`) that make them more like other libraries in Emacs.

I'm not quite sure my little functions are at that point yet. For now, I like the way that the functions are embedded in the blog posts and notes that explain them, and the org-babel `:comments` argument can insert links back to the sections of my configuration that I can open with `org-open-at-point-global` or `org-babel-tangle-jump-to-org`.

nil

Aha, I can fiddle with `org-babel-post-tangle-hook` to insert the boilerplate after the blocks have been written. Then I can add the `lexical-binding: t` cookie and the structure that makes it look more like the other libraries people define and use. It's always nice when I can get away with a small change that uses an existing hook. For good measure, let's even include a list of links to the sections of my config that affect that file.

```

;;;###autoload
(defun sacha-dotemacs-link-for-section-at-point (&optional combined)
  "Return the link for the current section."
  (let* ((custom-id (org-entry-get-with-inheritance "CUSTOM_ID"))
        (title (org-entry-get (point) "ITEM"))
        (url (if custom-id
                  (concat "dotemacs:" custom-id)
                  (concat sacha-emacs-config-url ":-:text=" (url-hexify-string title)))))
    (if combined
        (org-link-make-string
         url
         title)
        (cons url title))))

(eval-and-compile
  (require 'org-core nil t)
  (require 'org-macs nil t)
  (require 'org-src nil t))
(declare-function 'org-babel-tangle--compute-targets "ob-tangle")
(defun sacha-org-collect-links-for-tangled-files ()
  "Return a list of ((filename (link link link link)) ...)."
  (let* ((file (buffer-file-name))
        results)
    (org-babel-map-src-blocks (buffer-file-name)
      (let* ((info (org-babel-get-src-block-info))
            (link (sacha-dotemacs-link-for-section-at-point)))
        (cons (list file link link link) results)))))

```

```

    (mapc
      (lambda (target)
        (let ((list (assoc target results #'string=)))
          (if list
              (cl-pushnew link (cdr list) :test 'equal)
              (push (list target link) results))))
        (org-babel-tangle--compute-targets file info))))
;; Put it back in source order
(nreverse
 (mapcar (lambda (o)
           (cons (car o)
                 (nreverse (cdr o)))))
         results)))
(defvar sachas-emacs-config-module-links nil "Cache for links from tangled files.")

;;;###autoload
(defun sachas-emacs-config-update-module-info ()
  "Update the list of links."
  (interactive)
  (setq sachas-emacs-config-module-links
        (seq-filter
         (lambda (o)
           (string-match "sacha-" (car o)))
         (sachas-org-collect-links-for-tangled-files)))
  (setq sachas-emacs-config-modules-info
        (mapcar (lambda (group)
                  `((, (file-name-base (car group))
                     (commentary
                      .
                      ,(replace-regexp-in-string
                        "^"
                        ";; "
                        (concat
                          "Related Emacs config sections:\n\n"
                          (org-export-string-as
                           (mapconcat
                            (lambda (link)
                              (concat "- " (cdr link) "\\\n\n " (org-link-make-string (car ↵
                              ↵ link)) "\n"))
                            (cdr group)
                            "\n"
                            'ascii
                            t))))))
                  sachas-emacs-config-module-links)))

;;;###autoload
(defun sachas-emacs-config-prepare-to-tangle ()
  "Update module info if tangling my config."
  (when (string-match "Sacha.org" (buffer-file-name))
    (sachas-emacs-config-update-module-info)))

```

Let's set up the functions for tangling the boilerplate.

```

(defvar sachas-emacs-config-modules-dir "~/sync/emacs/lisp/")
(defvar sachas-emacs-config-modules-info nil "Alist of module info.")
(defvar sachas-emacs-config-url "https://sachachua.com/dotemacs")

;;;###autoload

```

```

(defun sacha-org-babel-post-tangle-insert-boilerplate-for-sacha-lisp ()
  (when (file-in-directory-p (buffer-file-name) sacha-emacs-config-modules-dir)
    (goto-char (point-min))
    (let ((base (file-name-base (buffer-file-name))))
      (insert (format ";;; %s.el — %s -*- lexical-binding: t -*-

;; Author: %s <%s>
;; URL: %s

;;; License:
;;
;; This file is not part of GNU Emacs.
;;
;; This is free software; you can redistribute it and/or modify
;; it under the terms of the GNU General Public License as published by
;; the Free Software Foundation; either version 3, or (at your option)
;; any later version.
;;
;; This is distributed in the hope that it will be useful,
;; but WITHOUT ANY WARRANTY; without even the implied warranty of
;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
;; GNU General Public License for more details.
;;
;; You should have received a copy of the GNU General Public License
;; along with GNU Emacs; see the file COPYING. If not, write to the
;; Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
;; Boston, MA 02110-1301, USA.

;;; Commentary:
;;
%s

;;; Code:

\n\n"

                                base
                                (or
                                  (assoc-default 'description
                                                  (assoc-default base sacha-emacs-config-modules-info ↵
                                  ↵ #'string=))
                                  ""))
                                user-full-name
                                user-mail-address
                                sacha-emacs-config-url
                                (or
                                  (assoc-default 'commentary
                                                  (assoc-default base sacha-emacs-config-modules-info ↵
                                  ↵ #'string=))
                                  "")))
      (goto-char (point-max))
      (insert (format "\n(provide '%s)\n;;; %s.el ends here\n"
                                base
                                base))
      (save-buffer))))

```

```

(setq sacha-emacs-config-url "https://sachachua.com/dotemacs")
(with-eval-after-load 'org
  (add-hook 'org-babel-pre-tangle-hook #'sacha-emacs-config-prepare-to-tangle)
  (add-hook 'org-babel-post-tangle-hook ↵

```



```
↪ #'sacha-org-babel-post-tangle-insert-boilerplate-for-sacha-lisp))
```

This also means that if I tangle my config from outside Emacs, I need to set things up a little.

```
(setq load-path (cl-remove-if (lambda (p) (string-match-p "lisp/org$" p)) load-path))
(add-to-list 'load-path "~/vendor/org-mode/lisp")
(add-to-list 'load-path "~/vendor/org-mode/contrib/lisp")
(load "~/vendor/org-mode/lisp/org-loaddefs.el" nil t)
(setq user-lisp-directory "~/sync/emacs/lisp")
(add-to-list 'load-path "~/sync/emacs/lisp")
(load "~/sync/emacs/lisp/.user-lisp-autoloads.el" nil t)
(package-initialize)
(setq safe-local-variable-directories '("/home/sacha/sync/orgzly/" "/home/sacha/sync/emacs/"))
(require 'ert)
(setq sacha-exporting t)
;; ###autoload
(defun sacha-dotemacs-link-for-section-at-point (&optional combined)
  "Return the link for the current section."
  (let* ((custom-id (org-entry-get-with-inheritance "CUSTOM_ID"))
        (title (org-entry-get (point) "ITEM"))
        (url (if custom-id
                  (concat "dotemacs:" custom-id)
                  (concat sacha-emacs-config-url ":-:text=" (url-hexify-string title)))))
    (if combined
        (org-link-make-string url title)
        (cons url title))))

(eval-and-compile
  (require 'org-core nil t)
  (require 'org-macs nil t)
  (require 'org-src nil t))
(declare-function 'org-babel-tangle--compute-targets "ob-tangle")
(defun sacha-org-collect-links-for-tangled-files ()
  "Return a list of ((filename (link link link link)) ...)."
  (let* ((file (buffer-file-name))
        results)
    (org-babel-map-src-blocks (buffer-file-name)
      (let* ((info (org-babel-get-src-block-info))
              (link (sacha-dotemacs-link-for-section-at-point)))
        (mapc
          (lambda (target)
            (let ((list (assoc target results #'string=)))
              (if list
                  (cl-pushnew link (cdr list) :test 'equal)
                  (push (list target link) results))))
            (org-babel-tangle--compute-targets file info))))
      ;; Put it back in source order
      (nreverse
        (mapcar (lambda (o)
                  (cons (car o)
                        (nreverse (cdr o))))
                  results))))
  (defvar sacha-emacs-config-module-links nil "Cache for links from tangled files.")
;; ###autoload
(defun sacha-emacs-config-update-module-info ()
```

```

"Update the list of links."
(interactive)
(setq sachas-emacs-config-module-links
  (seq-filter
    (lambda (o)
      (string-match "sacha-" (car o)))
    (sachas-org-collect-links-for-tangled-files)))
(setq sachas-emacs-config-modules-info
  (mapcar (lambda (group)
    `((, (file-name-base (car group))
      (commentary
        .
        ,(replace-regexp-in-string
          "^"
          ";; "
          (concat
            "Related Emacs config sections:\n\n"
            (org-export-string-as
              (mapconcat
                (lambda (link)
                  (concat "- " (cdr link) "\\\n\n " (org-link-make-string (car ↵
↵ link)) "\n"))
                (cdr group)
                "\n")
              'ascii
              t))))))
    sachas-emacs-config-module-links)))

;;;###autoload
(defun sachas-emacs-config-prepare-to-tangle ()
  "Update module info if tangling my config."
  (when (string-match "Sachas.org" (buffer-file-name))
    (sachas-emacs-config-update-module-info)))
(defvar sachas-emacs-config-modules-dir "~/sync/emacs/lisp/")
(defvar sachas-emacs-config-modules-info nil "Alist of module info.")
(defvar sachas-emacs-config-url "https://sachaschua.com/dotemacs")

;;;###autoload
(defun sachas-org-babel-post-tangle-insert-boilerplate-for-sachas-lisp ()
  (when (file-in-directory-p (buffer-file-name) sachas-emacs-config-modules-dir)
    (goto-char (point-min))
    (let ((base (file-name-base (buffer-file-name))))
      (insert (format ";;; %s.el — %s -*- lexical-binding: t -*-

;; Author: %s <%s>
;; URL: %s

;;; License:
;;
;; This file is not part of GNU Emacs.
;;
;; This is free software; you can redistribute it and/or modify
;; it under the terms of the GNU General Public License as published by
;; the Free Software Foundation; either version 3, or (at your option)
;; any later version.
;;
;; This is distributed in the hope that it will be useful,
;; but WITHOUT ANY WARRANTY; without even the implied warranty of
;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```

```

;; GNU General Public License for more details.
;;
;; You should have received a copy of the GNU General Public License
;; along with GNU Emacs; see the file COPYING.  If not, write to the
;; Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
;; Boston, MA 02110-1301, USA.

;;; Commentary:
;;
%s
;;; Code:

\n\n"

      base
      (or
        (assoc-default 'description
          (assoc-default base sach emacs-config-modules-info ↵
        ↵ #'string=))
        "")
      user-full-name
      user-mail-address
      sach emacs-config-url
      (or
        (assoc-default 'commentary
          (assoc-default base sach emacs-config-modules-info ↵
        ↵ #'string=))
        "")))
      (goto-char (point-max))
      (insert (format "\n(provide '%s)\n;;; %s.el ends here\n"
        base
        base))
      (save-buffer)))
(setq sach emacs-config-url "https://sachachua.com/dotemacs")
(with-eval-after-load 'org
  (add-hook 'org-babel-pre-tangle-hook #'sach emacs-config-prepare-to-tangle)
  (add-hook 'org-babel-post-tangle-hook ↵
    ↵ #'sach org-babel-post-tangle-insert-boilerplate-for-sach lisp))
(with-eval-after-load 'org
  (org-link-set-parameters
    "dotemacs"
    :complete #'sach org-dotemacs-complete
    :store #'sach org-dotemacs-store
    :insert-description #'sach org-dotemacs-insert-description
    :export #'sach org-dotemacs-export
    :follow #'sach org-dotemacs-open))

```

You can see the results at [.emacs.d/lisp](https://sachachua.com/dotemacs). For example, the function definitions in this post are at [lisp/sach emacs.el](https://sachachua.com/dotemacs).

8.26.4 Org Babel: Detangle just the current block

By default, Org Babel updates all the blocks tangled in the current file, which I'm a little bit nervous about. So I wrote some functions that let me narrow to just this block of code and also update just this block of code back in my literate config file.

[2026-03-31 Tue]

```
;; ###autoload
```

```
(defun sacha-org-babel-detangle-current-block ()
  "Detangle just the current block."
  (interactive)
  (save-restriction
    (sacha-org-babel-detangle-narrow-to-block)
    (org-babel-detangle)))

;;;###autoload
(defun sacha-org-babel-detangle-narrow-to-block ()
  "Narrow to just the current block."
  (interactive)
  (let ((start (if (save-excursion (re-search-backward (concat "^;; " org-link-bracket-re) nil <-
    ↵ t))
    (match-beginning 0)
    (point-min)))
    (end (if (save-excursion (re-search-forward (concat "^;; " org-link-bracket-re) nil t))
    (match-end 0)
    (point-max))))
    (narrow-to-region start end)))
```

8.26.5 Format source

[gists/format-org-mode-source-blocks.el](#) at 118c5a579a231862f4d1a548afe071e450af4e03 - gists - Forgejo

```
;;;###autoload
(defun sacha-format-all-advice ()
  (ignore-errors ; in case there's no language support
    (format-all-buffer)))
```

```
(use-package format-all :if sacha-laptop-p :defer t)
(with-eval-after-load 'org
  (advice-add #'org-edit-src-exit :before #'sacha-format-all-advice))
```

8.26.6 Run source blocks in an Org Mode subtree by custom ID

org

I like the way [Org Mode](#) lets me logically group functions into headings. If I give the heading a [CUSTOM_ID](#) property (which is also handy for exporting to HTML, as it turns into a link anchor), I can use that property to find the subtree. Then I can use [org-babel-execute-subtree](#) to execute all source blocks in that subtree, which means I can mix scripting languages if I want to.

Here's the code:

```
;;;###autoload
(defun sacha-org-execute-subtree-by-custom-id (id &optional filename)
  "Prompt for a CUSTOM_ID value and execute the subtree with that ID.
If called with \\[universal-argument], prompt for a file, and then prompt for the ID."
  (interactive (if current-prefix-arg
    (let ((file <-
    ↵ (read-file-name "Filename: ")))
    (list
    ↵
    ↵ (with-current-buffer (find-file-noselect file)
    ↵
    ↵ (completing-read
```

```

↩      "Custom ID: "
↩
↩      (org-property-values "CUSTOM_ID"))
                                         file))
                                         (list
                                         (completing-read ↩
↩ "Custom ID: " (org-property-values "CUSTOM_ID"))))
  (with-current-buffer (if filename (find-file-noselect filename) (current-buffer))
    (let ((pos (org-find-property "CUSTOM_ID" id)))
      (if pos
          (org-babel-execute-subtree)
          (if filename (error "Could not find %s in %s" id filename)
              (error "Could not find %s" id))))))

```

For example, in [Using Org Mode, Emacs Lisp, and TRAMP to parse meetup calendar entries and generate a crontab](#), I have a Emacs Lisp source block that generates a crontab on a different computer, and a shell source block that installs it on that computer.

Technical notes: `org-babel-execute-subtree` narrows to the current subtree, so if I want anything from the rest of the buffer, I need to widen the focus again. Also, it's wrapped in a `save-restriction` and a `save-excursion`, so someday I might want to figure out how to handle the cases where I want to change what I'm looking at.

`elisp: links` in Org Mode let me call functions by clicking on them or following them with `C-c C-o` (`org-open-at-point`). This means I can make links that execute subtrees that might even be in a different file. For example, I can define links like these:

- `[[elisp:(sacha-org-execute-subtree-by-custom-id "update" "~/sync/emacs-calendar/README.org")][Update Emacs calendar]]`
- `[[elisp:(sacha-org-execute-subtree-by-custom-id "crontab" "~/sync/emacs-calendar/README.org")][Update Emacs meetup crontab]]`

That could be a good starting point for a dashboard.

Related: [Execute a single named Org Babel source block](#) (p. 257)

8.26.7 Execute named babel block

```

;;;###autoload
(defun sacha-org-execute-src-block-by-name (name)
  (interactive (list (completing-read "Block: " (org-babel-src-block-names))))
  (save-excursion
    (goto-char (point-min))
    (when (re-search-forward (format "^#\\+NAME:[ \\t]+%s[ \\t]*$" (regexp-quote name)) nil t)
      (org-babel-execute-src-block))))

```

8.26.8 JSON

From <https://isamert.net/2022/01/04/dealing-with-apis-jsons-and-databases-in-org-mode.html>

```

;;;###autoload
(defun sacha-org-babel-execute:json (body params)
  (let ((jq (cdr (assoc :jq params)))
        (node (cdr (assoc :node params))))
    (cond

```

```
(jq
  (with-temp-buffer
    ;; Insert the JSON into the temp buffer
    (insert body)
    ;; Run jq command on the whole buffer, and replace the buffer
    ;; contents with the result returned from jq
    (shell-command-on-region (point-min) (point-max) (format "jq -r \"%s\"" jq) nil 't)
    ;; Return the contents of the temp buffer as the result
    (buffer-string)))
(node
  (with-temp-buffer
    (insert (format "const it = %s;" body))
    (insert node)
    (shell-command-on-region (point-min) (point-max) "node -p" nil 't)
    (buffer-string))))))
```

```
(defalias 'org-babel-execute:json #'sacha-org-babel-execute:json)
```

JQ

```
(use-package jq-mode
  :vc (:url "https://github.com/ljos/jq-mode")
  :defer t
  :config
  (org-babel-do-load-languages 'org-babel-load-languages
    '(((jq . t))))
```

8.26.9 Fix block indentation

```
;; ##autoload
(defun sacha-org-fix-block-indentation ()
  "Fix the indentation of the current src block."
  (interactive)
  (org-edit-special)
  (indent-region (point-min) (point-max))
  (org-edit-src-exit))
```

8.26.10 Let's try `literate-elisp`

```
(use-package literate-elisp :if sacha-laptop-p :defer t)
```

Then I should be able to use `literate-elisp-load-file` and still be able to jump to functions by definition.

8.27 Publishing

8.27.1 HTML

Org Mode: JS for translating times to people's local timezones

```
(setq org-html-datetime-formats '("%F" . "%FT%T%z"))
```

Changing Org Mode underlines to the HTML mark element

ORG

[2025-03-19 Wed]: Changed org-html to ox-html.

Apparently, [HTML has a mark element](#) that is useful for highlighting. ox-html.el in Org Mode doesn't seem to export that yet. I don't use `_` to underline things because I don't want that confused with links. Maybe I can override `org-html-text-markup-alist` to use it for my own purposes...

```
(with-eval-after-load 'ox-html
  (setf (alist-get 'underline org-html-text-markup-alist)
        "<mark>%s</mark>"))
```

Okay, let's try it with:

Let's see `_` how that works.

Let's see `how` that works. Oooh, that's promising.

Now, what if I want something fancier, like the way it can be nice to use different-coloured highlighters when marking up notes in order to make certain things jump out easily? A custom link might come in handy.

```
;; ##autoload
(defun sach-org-highlight-export (link desc format _)
  (pcase format
    ((or '11ty 'html)
     (format "<mark%s>%s</mark>"
             (if link
                 (format " class=\"%s\"" link)
                 link)
             desc))))
```

```
(with-eval-after-load 'org
  (org-link-set-parameters "hl" :export 'sach-org-highlight-export))
```

A [green highlight](#) might be good for ideas, while [red](#) might be good for warnings. (Idea: I wonder how to font-lock them differently in Emacs...)

I shouldn't rely only on the colours, since people reading through RSS won't get them and also since some people are colour-blind. Still, the highlights could make my blog posts easier to skim on my website.

Of course, now I want to port [Prot's excellent colours from the Modus themes](#) over to CSS variables so that I can have colours that make sense in both light mode and dark mode. Here's a snippet that exports the colours from one of the themes:

```
(require 'modus-operandi-theme)
(format ":root {\\n%s\\n}\\n"
  (mapconcat
    (lambda (entry)
      (format "  --modus-%s: %s;"
              (symbol-name (car ←
                             ↪ entry))
              (if (stringp (cadr ←
                             ↪ entry))
                  (cadr ←
                             ↪ entry)
                  (cadr ←
                             ↪ entry))))
    ""
    "\n"))
```

```
↩ "var(--modus-%s)" (symbol-name (cadr entry))))))
      modus-operandi-palette
      "\n"))
```

(format ↩)

So now my style.css has:

```
/* Based on Modus Operandi by Protesilaos Stavrou */
:root {
  // ...
  --modus-bg-red-subtle: #ffcfbf;
  --modus-bg-green-subtle: #b3fabf;
  --modus-bg-yellow-subtle: #fff576;
  // ...
}
@media (prefers-color-scheme: dark) {
  /* Based on Modus Vivendi by Protesilaos Stavrou */
  :root {
    // ...
    --modus-bg-red-subtle: #620f2a;
    --modus-bg-green-subtle: #00422a;
    --modus-bg-yellow-subtle: #4a4000;
    // ...
  }
}
mark { background-color: var(--modus-bg-yellow-subtle) }
mark.green { background-color: var(--modus-bg-green-subtle) }
mark.red { background-color: var(--modus-bg-red-subtle) }
```

Interesting, interesting...

Include inline SVGs in Org Mode HTML and Markdown exports

EMACS:ORG:IMAGES

- [2025-01-10 Fri]: Fixed descriptions.
- [2024-10-14 Mon]: Fixed path when inlining file URLs.
- [2024-10-07 Mon]: Now I can specify `#+ATTR_HTML :data-link t` to make it link instead of include.
- [2024-09-26 Thu]: Whoops, forgot to make sure ox-11ty is also covered.

In my Org Mode HTML and Markdown exports, I usually want to include SVGs inline so that I can use links. Sometimes I also want to use Javascript and CSS to modify elements within the images. I used to use a [sacha-include](#): (p. 289) link to do this, but I realized that I can also modify this behaviour by making my own functions that call `org-html-link` or `org-md-link` and then put those functions in `org-export-backend-transcoders`.

Here is an example of an SVG:

```
digraph g {
  rankdir=LR;
  node [fontcolor="#000000",fontname="Roboto,Arial,sans-serif"];
  edge [fontcolor="#000000",fontname="Roboto,Arial,sans-serif"];
  Graphviz -> "Org Mode" [label="SVG"];
  "Org Mode" -> {HTML Markdown};
  Graphviz[URL="https://graphviz.org",fontcolor="blue"];
}
```

The following code overrides HTML and Markdown exports to include SVGs.


```

;;;###autoload
(defun sach-ox-link-path (link _ info)
  (let* ((raw-path (org-element-property :path link))
         (setq raw-path
                (org-export-file-uri
                 (org-publish-file-relative-name raw-path info)))
         ;; Possibly append `:html-link-home' to relative file
         ;; name.
         (let ((home (and (plist-get info :html-link-home)
                           (org-trim
                            (plist-get info :html-link-home))))
                (when (and home
                           (plist-get info :html-link-use-abs-url)
                           (not (file-name-absolute-p raw-path))
                           (setq raw-path (concat (file-name-as-directory home) raw-path)))
                  raw-path)))

    raw-path))

;;;###autoload
(defun sach-org-html-link (link desc info)
  (if (and
       (string= (org-element-property :type link) "file")
       (not (plist-get (org-export-read-attribute :attr_html
              (org-element-parent-element link))
                       :data-link))
       (org-export-inline-image-p link (plist-get info :html-inline-image-rules)))
      (let ((path (org-element-property :path link))
            (attr (org-export-read-attribute :attr_html
              (org-element-parent-element link))))
        (if (string= (file-name-extension path) "svg")
            (with-temp-buffer
              (set-buffer-multibyte t)
              (insert-file-contents path)
              (if attr
                  (replace-regexp-in-string
                   "<svg "
                   (concat " <svg "
                           (buffer-string)
                           " ")
                   (buffer-string)))
              (org-html-link link desc info)))
            (org-html-link link desc info)))

  (org-html-link link desc info))

;;;###autoload
(defun sach-org-md-link (link desc info)
  (if (and (string= (org-element-property :type link) "file")
           (not (plist-get (org-export-read-attribute :attr_html
              (org-element-parent-element link))
                           :data-link))
           (let ((path (org-element-property :path link))
                 (if (string= (file-name-extension path) "svg")

```

```

                                (with-temp-buffer
                                  (insert-file-contents-literally path)
                                  (buffer-string))
                                (org-md-link link desc info)))
      (org-md-link link desc info)))

;;;###autoload
(defun sacha-org-11ty-link (link desc info)
  (if (and (string= (org-element-property :type link) "file")
            (not (plist-get (org-export-read-attribute :attr_html ↵
↵ (org-element-parent-element link))
                                ↵
                                :data-link))
            (not desc))
      (let ((path (org-element-property :path link))
            (attr (org-export-read-attribute :attr_html ↵
↵ (org-element-parent-element link))))
        (if (string= (file-name-extension path) "svg")
            (with-temp-buffer
              (set-buffer-multibyte t)
              (insert-file-contents path)
              (if attr
                  (replace-regexp-in-string
                    "<svg "
                    (concat
                      "<svg "
↵
                      " ")
                    (buffer-string))
                  (buffer-string)))
            (org-11ty-link link desc info)))
      (org-11ty-link link desc info)))

```

```

(with-eval-after-load 'ox-html
  (setf
    (alist-get 'link (org-export-backend-transcoders (org-export-get-backend 'html)))
    'sacha-org-html-link))
(with-eval-after-load 'ox-md
  (setf
    (alist-get 'link (org-export-backend-transcoders (org-export-get-backend 'md)))
    'sacha-org-md-link))
(with-eval-after-load 'ox-11ty
  (setf
    (alist-get 'link (org-export-backend-transcoders (org-export-get-backend '11ty)))
    'sacha-org-11ty-link))

```

Org Mode: Export HTML, copy files, and serve the results via simple-httpd so that media files work

[2026-03-23 Mon]: Update Oh, ignore all of this! For some reason, when I export the regular Org Mode way, my media files work. Maybe it was just a weird hiccup!

In Org Mode, when you use "Export to HTML - As HTML file and open", the resulting HTML file is loaded using a `file://` URL. This means you can't load any media files. In my post about [pronunciation practice](#), I wanted to test the playback without waiting for my 11ty-based static site generator to churn through the files.

`simple-httpd` lets you run a web server from Emacs. By default, the `httpd-root` is `~/public_html` and `httpd-port` is `8085`, but you can configure it to be somewhere else. Here I set it up to create a new temporary directory, and to delete that directory afterwards.

```
(use-package simple-httpd
  :config
  (setq httpd-root (make-temp-file "httpd" t))
  :hook
  (httpd-stop . sachas-simple-httpd-remove-temporary-root)
  (kill-emacs . httpd-stop))
```

```
;; ##autoload
(defun sachas-simple-httpd-remove-temporary-root ()
  "Remove `httpd-root' only if it's a temporary directory."
  (when (file-in-directory-p httpd-root temporary-file-directory)
    (delete-directory httpd-root t)))
```

The following code exports your Org buffer or subtree to a file in that directory, copies all the referenced local files (if they're newer) and updates the links in the HTML, and then serves it via `simple-httpd`. Note that it just overwrites everything without confirmation, so if you refer to files with the same name, only the last one will be kept.

```
(with-eval-after-load 'ox
  (org-export-define-derived-backend 'sachas-html-served 'html
    :menu-entry
    '(?s "Export to HTML and Serve"
      ((?b "Buffer" sachas-org-serve-buffer)
       (?s "Subtree" sachas-org-serve-subtree)))))
```

```
;; ##autoload
(defun sachas-org-serve-buffer (&optional async _subtreep visible-only body-only ext-plist)
  (sachas-org-export-and-serve nil))

;; ##autoload
(defun sachas-org-serve-subtree (&optional async _subtreep visible-only body-only ext-plist)
  (sachas-org-export-and-serve t))

;; Based on org-11ty--copy-files-and-replace-links
;; Might be a good idea to use something DOM-based instead
(defun sachas-html-copy-files-and-replace-links (info &optional destination-dir)
  (let ((file-regexp "\\(?:src\\|href\\|poster\\)=\\\"\\\\\\\\(file:\\|)?.*?\\\\\\\\\"")
        (destination-dir (or destination-dir (file-name-directory (plist-get info :file-path))))
        file-all-urls file-name beg
        new-file file-re
        unescaped)
    (unless (file-directory-p destination-dir)
      (make-directory destination-dir t))
    (unless (file-directory-p destination-dir)
      (error "%s is not a directory." destination-dir))
    (save-excursion
      (goto-char (point-min))
      (while (re-search-forward file-regexp nil t)
        (setq file-name (or (match-string 1) (match-string 2)))
        (unless (or (string-match "^#" file-name)
                    (get-text-property 0 'changed file-name))
          (setq file-name
```

```

        (replace-regexp-in-string
         "\\?.+" ""
         (save-match-data (if (string-match "^file:" file-name)
                               (substring file-name 7)
                               file-name))))
      (setq unescaped
            (replace-regexp-in-string
             "%23" "#"
             file-name))
      (setq new-file (concat
                      (if info (plist-get info :permalink) "")
                      (file-name-nondirectory unescaped))
            (unless (org-url-p file-name)
              (let ((new-file-name (expand-file-name (file-name-nondirectory unescaped)
                                                       destination-dir)))
                (condition-case err
                  (when (or (not (file-exists-p new-file-name))
                            (file-newer-than-file-p unescaped new-file-name))
                    (copy-file unescaped new-file-name t))
                  (error nil))
                (when (file-exists-p new-file-name)
                  (save-excursion
                    (goto-char (point-min))
                    (setf file-re (concat "\\(?:\?:file://\\)?" (regexp-quote file-name) "\\)"))
                    (while (re-search-forward file-re nil t)
                      (replace-match
                       (propertize
                        (save-match-data (replace-regexp-in-string "#" "%23" new-file))
                        'changed t))
                      t t nil)))
              1)))))))))

;;###autoload
(defun sacha-org-export-and-serve (&optional subtreep)
  "Export current org buffer (or subtree if SUBTREEP) to HTML and serve via simple-httpd."
  (interactive "P")
  (require 'simple-httpd)
  (httpd-stop)
  (unless httpd-root (error "Set `httpd-root'."))
  (unless (file-directory-p httpd-root)
    (make-directory httpd-root t))
  (unless (file-directory-p httpd-root)
    (error "%s is not a directory." httpd-root))
  (let* ((out-file (expand-file-name (concat (file-name-base (buffer-file-name)) ".html")
                                     httpd-root))
         (html-file (org-export-to-file 'sacha-html-served out-file nil subtreep)))
    ;; Copy all the files and rewrite all the links
    (with-temp-file out-file
      (insert-file-contents out-file)
      (sacha-html-copy-files-and-replace-links
       `(:permalink "/") httpd-root))
    (httpd-start))

```



```

        (let ((filename (org-entry-get (point) "EXPORT_ELEVENTY_FILE_NAME")))
          (dolist (dir (list sacha-11ty-site-dir sacha-11ty-local-dir ←
↪ sacha-11ty-remote-dir sacha-11ty-base-dir))
            (condition-case nil
              (when (and dir (file-directory-p ←
↪ (expand-file-name filename dir)))
                (delete-directory ←
↪ (expand-file-name filename dir) t))
              (error nil)))
            (org-delete-property "EXPORT_ELEVENTY_FILE_NAME")
            (org-delete-property "EXPORT_DATE")
            (org-delete-property "EXPORT_ELEVENTY_PERMALINK"))))
      ((derived-mode-p '(html-mode web-mode))
       ;; called from an index.html or page.html, maybe?
       (let* ((file (buffer-file-name))
              (json-file (concat (file-name-sans-extension ←
↪ (buffer-file-name))
↪ ".11tydata.json")
              (json-data (and (file-exists-p json-file) ←
↪ (json-read-file json-file))))
         ;; delete the published files
         (when (alist-get 'permalink json-data)
           (dolist (dir (list sacha-11ty-site-dir sacha-11ty-local-dir ←
↪ sacha-11ty-remote-dir))
             (condition-case nil
               (when (and dir (file-directory-p ←
↪ (expand-file-name (concat "." (alist-get 'permalink ←
↪ json-data)) dir)))
                 (delete-directory ←
↪ (expand-file-name (concat "." (alist-get 'permalink json-data)) dir) t))
               (error nil))))
           ;; delete the .json and the .html file
           (when (file-exists-p json-file)
             (delete-file json-file))
           (kill-buffer (current-buffer))
           (delete-file file))))))

(defalias 'sacha-org-11ty-delete-current-post #'sacha-org-11ty-unpublish-current-post)

;; ##autoload
(defun sacha-org-11ty-copy-permalink ()
  (interactive)
  (kill-new (concat "https://sachachua.com" (org-entry-get (point) ←
↪ "EXPORT_ELEVENTY_PERMALINK"))))

;; ##autoload
(defun sacha-org-11ty-browse-local ()
  (interactive)
  (unless (seq-find (lambda (o) (string-match "--serve" (assoc-default 'args (cdr o) nil ←
↪ "")))
    (proced-process-attributes)
    (let ((default-directory "~/proj/static-blog"))
      (setq sacha-org-11ty-serve-process (start-process "serve" nil "make" ←
↪ "serve"))
      (browse-url "http://localhost:8080/blog"))

```

```

;;;###autoload
(defun sacha-org-11ty-serve-stop ()
  (interactive)
  (if (process-live-p sacha-org-11ty-serve-process)
      (stop-process sacha-org-11ty-serve-process)
      (when-let ((proc (seq-find (lambda (o) (string-match "--serve" (assoc-default o 'args (cdr o) nil ""))))
        (call-process "kill" nil nil nil (number-to-string) (car proc))))))

;;;###autoload
(defun sacha-org-11ty-prepare-subtree ()
  (interactive)
  (unless (or (org-entry-get (point) "EXPORT_DATE")
              (org-entry-get-with-inheritance "DATE"))
    (org-entry-put (point) "EXPORT_DATE" (format-time-string "%Y-%m-%dT%T%z")))
  (let ((path (concat "blog/" (format-time-string "%Y/%m/")
                     (sacha-make-slug (org-get-heading t t t))
                     "/")))
    (unless (org-entry-get (point) "EXPORT_ELEVENTY_PERMALINK")
      (org-entry-put (point) "EXPORT_ELEVENTY_PERMALINK" (concat "/" path)))
    (unless (org-entry-get (point) "EXPORT_ELEVENTY_FILE_NAME")
      (org-entry-put (point) "EXPORT_ELEVENTY_FILE_NAME" path)))

  (with-eval-after-load '11ty
    (advice-add
      'org-11ty-export-to-11tydata-and-html
      :before
      (lambda (&optional _ subtreep &rest _)
        (when (and subtreep (not (org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK")))
          (sacha-org-11ty-prepare-subtree))))))

;;;###autoload
(defun sacha-org-11ty-rename-subtree ()
  (interactive)
  (let ((new-path (concat "blog/" (format-time-string "%Y/%m/")
                          (sacha-make-slug (org-get-heading t t t))
                          "/")))
    (when (not (string= new-path (org-entry-get (point) "EXPORT_ELEVENTY_FILE_NAME")))
      (when
        (file-exists-p (expand-file-name
                        (org-entry-get (point) "EXPORT_ELEVENTY_FILE_NAME")
                        sacha-11ty-base-dir))
          (rename-file (expand-file-name
                        (org-entry-get (point) "EXPORT_ELEVENTY_FILE_NAME")
                        sacha-11ty-base-dir)
                        new-path
                        t)
          (org-entry-put (point) "EXPORT_ELEVENTY_FILE_NAME" new-path))
      (org-entry-put (point) "EXPORT_ELEVENTY_FILE_NAME" new-path))
    (expand-file-name
      (org-entry-get (point) "EXPORT_ELEVENTY_FILE_NAME")
      sacha-11ty-base-dir))

```

```

new-path
↵

    ↵ sachal1ty-base-dir)))
      (org-entry-put (point) "EXPORT_ELEVENTY_PERMALINK" (concat "/" path))
      (org-entry-put (point) "EXPORT_ELEVENTY_FILE_NAME" path))))

;;;###autoload
(defun sachal1ty-convert-to-njk ()
  (interactive)
  (let* ((filename (buffer-file-name))
         (old-buffer (current-buffer))
         (new-name (concat (file-name-base filename) ".njk")))
    (save-buffer)
    (rename-file filename new-name)
    (find-file new-name)
    (kill-buffer old-buffer)))

;;;###autoload
(defun sachal1ty-browse-page ()
  (interactive)
  (if (org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK")
      (browse-url (concat "http://localhost:8080" (org-entry-get-with-inheritance ↵
        ↵ "EXPORT_ELEVENTY_PERMALINK"))))
      (let* ((json-object-type 'plist)
              (data (json-read-file (concat (file-name-base (buffer-file-name)) ".11tydata.json"))))
        (browse-url (concat "http://localhost:8080" (plist-get data :permalink))) )))

;;;###autoload
(defun sachal-org-11ty-pathname ()
  (if (derived-mode-p 'org-mode)
      (file-name-directory (org-entry-get-with-inheritance ↵
        ↵ "EXPORT_ELEVENTY_FILE_NAME"))
      (let ((url (thing-at-point 'url)))
        (when url
          (url-file-directory (url-filename (url-generic-parse-url ↵
            ↵ url)))))))

;;;###autoload
(defun sachal-org-11ty-find-post (url)
  (interactive (list (sachal-org-11ty-pathname)))
  ;; check in posts.org
  (find-file "~/sync/orgzly/posts.org")
  (let ((pos (org-find-property "EXPORT_ELEVENTY_PERMALINK" url)))
    (when pos (goto-char pos))))

;;;###autoload
(defun sachal-org-11ty-find-file (file)
  (interactive
   (list
    (completing-read
     (if (sachal-org-11ty-pathname)
         (format "Post (%s): " (concat "/" (sachal-org-11ty-pathname)))
         "Post: ")
     (mapcar (lambda (o) (replace-regexp-in-string ↵
        ↵ "^~/proj/static-blog\\|index.html$" "" o))
             (directory-files-recursively ↵
        ↵ "~/proj/static-blog/blog" "index\\.html" nil))
     nil nil nil nil (concat "/" (sachal-org-11ty-pathname))))))
  (find-file
   (expand-file-name

```



```

        "index.html"
        (expand-file-name
         (concat "." file)
         "~/proj/static-blog")))))

;; ##autoload
(defun sacha-org-11ty-post-to-mastodon (&optional post-automatically)
  (interactive (list current-prefix-arg))
  (let ((message (concat (org-entry-get (point) "ITEM") " https://sachachua.com" ←
    ↪ (org-entry-get (point) "EXPORT_ELEVENTY_PERMALINK"))))
    (if post-automatically
      (sacha-mastodon-toot-public-string message)
      (mastodon-toot)
      (insert message))))

;; https://kitchingroup.cheme.cmu.edu/blog/2013/05/05/Getting-keyword-options-in-org-files/
;; ##autoload
(defun sacha-org-keywords ()
  "Parse the buffer and return a cons list of (property . value).
This is extracted from lines like:
#+PROPERTY: value"
  (org-element-map (org-element-parse-buffer 'element) 'keyword
    (lambda (keyword) (cons (org-element-property :key keyword)
      (org-element-property :value keyword)))))

;; ##autoload
(defun sacha-11ty-copy-file-and-insert-into-org (filename caption)
  (interactive (list (read-file-name "File: ")
    (read-string "Caption: ")))
  (let ((path (expand-file-name
    (file-name-nondirectory filename)
    (expand-file-name
      (org-entry-get-with-inheritance
        "EXPORT_ELEVENTY_FILE_NAME")
        (assoc-default ←
    ↪ "ELEVENTY_BASE_DIR" (sacha-org-keywords))))
    (copy-file filename path t)
    (insert "#+CAPTION: " caption "\n"
      (org-link-make-string (concat "file:" path)) ←
    ↪ "\n"))))

;; ##autoload
(defun sacha-org-replace-with-permalink ()
  (interactive)
  (let* ((elem (org-element-context))
    (path (org-element-property :path elem))
    (description (org-element-property :description elem))
    (type (org-element-property :type elem))
    (permalink (org-entry-get (point) "EXPORT_ELEVENTY_PERMALINK" ←
    ↪ t))
    (base-url sacha-blog-base-url))
    (when (member type '("file" "audio" "video"))
      (delete-region (org-element-begin elem) (org-element-end elem))
      (insert (org-link-make-string (concat
        (if (string= type "file") "" ←
        ↪ (concat type ":"))

```



```

      (let ((props (org-collect-keywords '("TITLE")))
            (id (org-entry-get-with-inheritance "CUSTOM_ID")))
        (org-link-store-props
         :link
         (concat
          "topic:" (file-name-base (buffer-file-name))
          (if id
              (concat "#" id)
              ""))
         :description
         (save-excursion
          (if id (progn
                  (goto-char ←
                        (org-find-property "CUSTOM_ID" id))
                  (org-entry-get (point) ←
                                "ITEM"))
              (car (assoc-default "TITLE" props #'string=)))))))

```

```

(with-eval-after-load 'org
  (org-link-set-parameters
   "topic"
   :follow #'sacha-org-topic-open
   :store #'sacha-org-topic-store
   :insert-description #'sacha-org-link-insert-description
   :export #'sacha-org-topic-export
   :complete #'sacha-org-topic-complete))

```

Linking to blog posts

- [2024-01-19 Fri]: added link description

```

(defvar sacha-blog-base-url "https://sachachua.com/")

;;;###autoload
(defun sacha-org-blog-complete ()
  "Select a blog post and return its URL."
  (sacha-blog-url (sacha-consult-blog-posts-by-title)))

;;;###autoload
(defun sacha-org-blog-export (link desc format _)
  (let ((path (concat (if (eq format '11ty) "/" sacha-blog-base-url)
                      (replace-regexp-in-string "\\..html$" "/"
                      (replace-regexp-in-string "^/\\|index.html$" ""
                      link)))))
    (pcase format
      ((or 'html '11ty) (format "<a href=\"%s\">%s</a>" path (or desc link)))
      ('latex (format "\\href{%s}{%s}" path desc))
      ('texinfo (format "@uref{%s,%s}" path desc))
      ('ascii (if desc (format "%s (%s)" desc path)
                  path))))))

```



```

↪ (alist-get 'title o))
                                     "\n"))
                                     (sacha-blog-posts)))

```

2. Making it easier to add a category to a blog post embark I want to be able to quickly add a category to a post when I'm looking at the file or when I'm working with a link.

```

;; ##autoload
(defun sacha-11ty-complete-blog-post ()
  (completing-read
   "Post: "
   (mapcar (lambda (o)
             (file-name-directory (file-relative-name o sacha-11ty-base-dir)))
           (directory-files-recursively (expand-file-name "blog" sacha-11ty-base-dir) ↪
                                         ↪ "index\\.html" nil))))
;; ##autoload
(defun sacha-11ty-ripgrep ()
  (interactive)
  (consult-ripgrep (expand-file-name "blog" sacha-11ty-base-dir)))

;; ##autoload
(defun sacha-11ty-post-categories (file)
  (assoc-default 'categories
                 (let ((json-object-type 'alist)
                       (json-array-type 'list))
                   (json-read-file (sacha-11ty-json-filename file))))))

;; ##autoload
(defun sacha-11ty-add-category-tag (cat)
  (interactive (list (sacha-11ty-complete-category "Category: ")))
  (org-set-tags (cons cat (org-get-tags))))

;; ##autoload
(defun sacha-11ty-complete-category (prompt &optional categories)
  (let ((all-categories
        (json-read-file
         (expand-file-name "siteCategories.json"
                           (expand-file-name "_data" sacha-11ty-base-dir)))))
    (completing-read
     (if categories
         (format "%s(current: %s) "
                 prompt
                 (string-join categories ", "))
         prompt)
     (mapcar (lambda (c) (assoc-default 'slug c))
             all-categories))))

;; ##autoload
(defun sacha-11ty-json-filename (path)
  (concat (file-name-sans-extension (sacha-11ty-html-filename path)) ".11tydata.json"))

;; ##autoload
(defun sacha-11ty-change-details (file modify-func)
  (let* ((json-object-type 'alist)
         (json-array-type 'list)
         (json-file (sacha-11ty-json-filename file))
         (json (funcall modify-func (json-read-file json-file))))
    (when json

```

```
(with-temp-file json-file
  (insert (json-encode json)))
(json-file)))

;;;###autoload

(defun sachu-11ty-add-category-to-post (file new-category)
  (interactive (list (buffer-file-name)
                     (sachu-11ty-complete-category "Add category: "
                                                    (sachu-11ty-post-categories file))))
  (sachu-11ty-change-details
   file
   (lambda (json)
     (let ((categories (assoc-default 'categories json)))
       (if categories
           (unless (member new-category categories)
             (setcdr (assoc 'categories json)
                      (cons new-category categories)))
           (setq json (cons (cons 'categories (cons new-category categories)) json)))
       json))))
```

Then it makes sense to be able to work with blog URLs from an Embark action:

[illegible]

```

                                path
                                (assoc-default ←
↪ 'permalink path))))))

;;;###autoload
(defun sach-embark-org-blog-open-in-browser (path)
  (interactive (list (sach-consult-blog-posts-by-title)))
  (browse-url (sach-blog-url path)))

;;;###autoload
(defun sach-embark-blog-insert-link (post)
  (interactive (list (sach-consult-blog-posts-by-title)))
  (when (looking-back "\\]\\]" ) ; end of a link, add a space
    (insert ", "))
  (sach-org-insert-link-dwim (sach-blog-url post)
↪
↪ (and (listp post) (assoc-default 'title post))))

;;;###autoload
(defun sach-11ty-exported-org-filename (info)
  "Return the exported .org file for INFO, or nil if there isn't any."
  (when (stringp info) (setq info (sach-blog-post-info-for-url info)))
  (let-alist info
    (let* ((org (and .permalink (expand-file-name "index.org" (expand-file-name (concat ←
↪ ". " .permalink) sach-11ty-base-dir)))))
      (cond
        ((and org (file-exists-p org))
         org)
        (.inputPath
         (expand-file-name .inputPath sach-11ty-base-dir)))))

;;;###autoload
(defun sach-blog-post--position (info &optional find-file)
  "Return the file position marker for a blog post INFO.
FIND-FILE is the file open function, defaulting to `consult--file-action'."
  (when (stringp info) (setq info (sach-blog-post-info-for-url info)))
  (when info
    (let (pos
          (files (delq nil
↪
↪ (list "~/sync/emacs/Sacha.org"
↪
↪ "~/sync/orgzly/posts.org"
↪
↪ (alist-get 'source_path info)
↪
↪ (sach-11ty-exported-org-filename info)))
          (line-number (alist-get 'line_number info 0)))
      (when-let* ((source-path
                  (seq-find
↪ (lambda (filename)
↪
↪ (with-current-buffer (find-file-noselect filename)
↪
↪ (save-excursion
↪
↪ (save-restriction

```



```

↳                                     (widen)
↳                                     (if (assoc-default 'anchor info)
↳                                     (when-let* ((pos (org-find-property
↳ "CUSTOM_ID" (assoc-default 'anchor info))))
↳                                     (goto-char pos)
↳                                     (setq line-number
↳ (line-number-at-pos nil t))
↳                                     (buffer-file-name))
↳                                     (setq pos (org-find-property "EXPORT_ELEVENTY_PERMALINK"
↳ (assoc-default 'permalink info)))
↳                                     (when pos
↳                                     (progn
↳                                     (goto-char pos)
↳                                     (when line-number
↳                                     (forward-line line-number))
↳                                     (setq line-number
↳ (line-number-at-pos nil t))
↳                                     (buffer-file-name))))))
files)))
      (consult--marker-from-line-column
      (funcall (or find-file #'consult--file-action)
↳ (file-truename source-path)
      line-number 0))))

;;;###autoload
(defun sach-blog-post-info-for-url (url &optional all-posts)
  "Return the alist for URL.
  The alist will have the following keys: permalink, date, title,
  categories, inputPath. If URL has an anchor, add it as an anchor
  attribute.
  "
  (let (anchor entry)
    (if (listp url)
      url
      (when (string-match "#\\(.*\\)" url)
        (setq anchor (match-string 1 url))
        (setq url (replace-match "" nil nil url)))
      (setq url (replace-regexp-in-string
        (concat "^" (regexp-quote sach-11ty-base-dir)
          "\\|/"
          (regexp-quote (expand-file-name sach-11ty-base-dir))
          "\\|/"
          (regexp-quote (file-truename sach-11ty-base-dir))
          "\\|index\\.org$\\|\\.org$") "" url))
      (when (string-match (regexp-quote sach-blog-base-url) url)
        (setq url (substring url (match-end 0)))))

```

```

    (when (string-match "\\?" url)
      (setq url (substring url 0 (match-beginning 0))))
    (unless (string-match "^/" url)
      (setq url (concat "/" url)))
    (setq entry
      (seq-find (lambda (o) (string= (alist-get 'permalink o) url))
        (or all-posts (sacha-blog-posts))))
    (if anchor
      (cons `(anchor . ,anchor) entry)
      entry))))

(ert-deftest sacha-blog-post-info-for-url--handle-hash ()
  "Tests `sacha-blog-post-info-for-url'."
  (should
    (equal
      (sacha-blog-post-info-for-url ↵
        ↵ "/blog/2026/04/yayemacs-10-emacs-coaching-with-prot-packaging-emacs-lisp/#projects-experiment-with-le
        '((anchor
          . ↵
        ↵ "projects-experiment-with-learning-from-prot-yayemacs-10-emacs-coaching-with-prot-packaging-emacs-lis
          (permalink
            . "/blog/2026/04/yayemacs-10-emacs-coaching-with-prot-packaging-emacs-lisp/")
          (date . "2026-04-04T02:23:03.000Z")
          (title
            . "#YayEmacs 10: Emacs coaching with Prot: Emacs workflows and streaming")
          (categories "emacs" "yay-emacs")
          (inputPath
            . ↵
        ↵ "./blog/2026/04/yayemacs-10-emacs-coaching-with-prot-packaging-emacs-lisp/index.html")))))

(ert-deftest sacha-blog-post-info-for-url ()
  (should
    (equal
      (sacha-blog-post-info-for-url
        ↵
        ↵ "https://sachachua.com/blog/2025/10/added-multiple-timezone-support-to-casual-timezone-planner/"
        )
      '((permalink . ↵
        ↵ "/blog/2025/10/added-multiple-timezone-support-to-casual-timezone-planner/"
        (date . "2025-10-08T13:53:11.000Z")
        (title . "Added multiple timezone support to casual-timezone-planner")
        (categories "emacs")
        (inputPath . ↵
        ↵ "./blog/2025/10/added-multiple-timezone-support-to-casual-timezone-planner/index.html")))))
  (should
    (equal
      (sacha-blog-post-info-for-url
        ↵
        ↵ "/blog/2021/03/org2blog-add-a-note-to-the-bottom-of-blog-posts-exported-from-sacha-config-file/"
        '((permalink . ↵
        ↵ "/blog/2021/03/org2blog-add-a-note-to-the-bottom-of-blog-posts-exported-from-sacha-config-file/" ↵
        ↵ (date . "2021-03-25T00:00:00.000Z") (title . "Add a note to the bottom of blog ↵
        ↵ posts exported from my config file") (categories "emacs" "org") (inputPath . ↵
        ↵ "./blog/2021/03/org2blog-add-a-note-to-the-bottom-of-blog-posts-exported-from-sacha-config-file.html"

;;; ##autoload
(defun sacha-blog-post--state ()
  "Blog post search state function, managing preview window and cleanup."
  ;; These functions are closures captured when the state is initialized by consult--read

```

```

(let ((preview (consult--jump-preview))
      (open (consult--temporary-files))
      (jump (consult--jump-state)))
  ;; The returned lambda is the actual preview function called by Consult
  (lambda (action cand)
    (unless cand
      (funcall open))
    (funcall
      preview action
      (and (or (eq action 'preview))
            (sacha-blog-post--position
              (car (split-string cand " "))
              (and (not (eq action 'return)) open)))))))

;;;###autoload
(defun sacha-11ty-current-post ()
  "Return the current blog post info if any."
  (cond
   ((derived-mode-p 'org-mode)
    (when (org-entry-get-with-inheritance "EXPORT_ELEVENTY_FILE_NAME")
      (let ((filename (org-entry-get-with-inheritance "EXPORT_ELEVENTY_FILE_NAME")))
        `((filename . ,filename)
          (permalink . ,(org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK"))
          (title . ,(save-excursion
                     (save-restriction
                      (widen)
                      (goto-char (org-find-property "EXPORT_ELEVENTY_FILE_NAME" ←
→ (org-entry-get-with-inheritance "EXPORT_ELEVENTY_FILE_NAME")))
                      (org-entry-get (point) "ITEM"))))
          (date . ,(org-entry-get-with-inheritance "EXPORT_DATE"))))))
   ((derived-mode-p '(html-mode web-mode))
    ;; called from an index.html or page.html, maybe?
    (let* ((file (buffer-file-name))
           (json-file (concat (file-name-sans-extension ←
→ ".11tydata.json")
                              (file-exists-p json-file)
                              (json-read-file json-file))))
      (cons
       (cons 'filename file)
       json-data))))))

(defun sacha-consult--blog-posts ()
  (mapcar
   #'sacha-blog-format-for-completion
   (delq
    nil
    (append
     (list (sacha-11ty-current-post))
     ;;todo: make a function to get the current blog post context
     (sort (sacha-blog-posts) :key (lambda (o) (alist-get 'date o)) :lessp ←
→ #'string< :reverse t)
     nil))))))

;;;###autoload
(defun sacha-consult-blog-posts-by-title (&optional query)
  (interactive)

```

```

(let* ((options (sacha-consult--blog-posts))
      (val (consult--read
                options
                :prompt "Search blog posts (exact): "
                :category 'sacha-blog
                :sort nil
                :require-match nil
                ;; Preview no longer working: ↵
                ↵ text properties not included?
                ;; :state (sacha-blog-post--state)
                :initial query)
      ))
  (or (and (stringp val) (string-match "^https://\\|^/" val)
          (sacha-blog-post-info-for-url val))
      (get-text-property 0 'consult--candidate val)
      (assoc-default val options #'string=)))

;;;###autoload
(defun sacha-blog-format-for-completion (result)
  (let-alist result
    (let* ((title (or .title "No Title"))
          (date (or .date "2000-01-01"))
          (year (substring date 0 4))
          (categories (string-join .categories ", "))
          (permalink .permalink)
          (final-display
            (format "%-5s %s [%s]"
                    (propertize year 'face 'font-lock-comment-face) title categories)))
      (put-text-property 0 1 'consult--candidate result final-display)
      (cons final-display result))))

```

```

(with-eval-after-load 'embark
  (add-to-list 'embark-target-finders #'sacha-embark-org-blog-target)
  (defvar-keymap embark-sacha-blog-actions
    :parent embark-general-map
    :doc "Shortcuts for my blog"
    "h" #'sacha-blog-edit-html
    "j" #'sacha-blog-edit-json
    "e" #'sacha-blog-find-org
    "c" #'sacha-embark-org-blog-add-category
    "i" #'sacha-embark-blog-insert-link
    "b" #'sacha-embark-org-blog-open-in-browser)
  (add-to-list 'embark-keymap-alist '(sacha-blog . embark-sacha-blog-actions)))

```

Then I want to be able to add a category to all the blog posts in a region:

```

;;;###autoload
(defun sacha-11ty-add-category-to-all-posts-in-region (category beg end)
  (interactive (list (sacha-11ty-complete-category "Category: ")
                    (min ↵
                      (point) (mark))
                    (max ↵
                      (point) (mark)))))
  (goto-char beg)
  (while (re-search-forward org-link-bracket-re end t)
    (sacha-11ty-add-category-to-post
      (sacha-11ty-html-filename (org-element-property :raw-link ↵
        (org-element-context))))

```

```
category)))
```

embark-11ty

11TY:ORG:EMACS:EMBARK

```
(defvar sacha-11ty-base-dir "~/proj/static-blog/")

;;;###autoload
(defun sacha-blog-edit-org (info)
  (interactive (list (sacha-consult-blog-posts-by-title)))
  (unless (listp info) (setq info (sacha-blog-post-info-for-url info)))
  (org-goto-marker-or-bmk (sacha-blog-post--position info)))

;;;###autoload
(defun sacha-blog-find-html (url)
  "Go to the HTML file for URL."
  (interactive (list (sacha-complete-blog-post-url)))
  (setq url (sacha-org-link-as-url url))
  (when (string-match "https://sachachua\\.com/\\(blog/.*/\\)" url)
    (find-file
     (expand-file-name
      "index.html"
      (expand-file-name (match-string 1 url)
                        sacha-11ty-base-dir)))))

;;;###autoload
(defalias 'sacha-blog-find-org #'sacha-blog-edit-org)
```

```
(with-eval-after-load 'embark
  (define-key embark-url-map "v" #'sacha-blog-find-org)
  (define-key embark-org-link-map "v" #'sacha-blog-find-org))
```

Moving my Org post subtree to the 11ty directory 11TY:ORG:EMACS:BLOGGING I sometimes want to move the Org source for my blog posts to the same directory as the 11ty-exported HTML. This should make it easier to update and reexport blog posts in the future. The following code copies or moves the subtree to the 11ty export directory.

```
;;;###autoload
(defun sacha-org-11ty-copy-subtree (&optional do-cut subtreep)
  "Copy the subtree for the current post to the 11ty export directory.
With prefix arg, move the subtree."
  (interactive (list current-prefix-arg))
  (let* ((info (org-combine-plists
    ↪ (org-export--get-export-attributes '11ty subtreep) ↪
    ↪ (org-export--get-buffer-attributes) ↪
    ↪ '11ty subtreep)))
    (file-properties
     (seq-filter (lambda (entry)
       ↪ (string-match (regexp-opt
       ↪ "ELEVENTY_COLLECTIONS"
       ↪ "ELEVENTY_BASE_DIR"
       ↪ "ELEVENTY_BASE_URL"
```

```

↪                                     ↪ "TITLE" ↪
↪                                     ↪ ↪
↪ "ELEVENTY_CATEGORIES" ↪ ↪
↪ ↪ ↪
↪ "ELEVENTY_LAYOUT")) ↪ ↪
↪                                     ↪ ↪
↪                                     ↪ (car entry)))
↪                                     ↪ (org-element-map ↪ ↪
↪ (org-element-parse-buffer) 'keyword ↪ ↪
↪                                     ↪ (lambda (el) (cons ↪ ↪
↪ (org-element-property :key el) ↪ ↪
↪                                     ↪ ↪
↪                                     ↪ (org-element-property :value el))))))
↪                                     ↪ (entry-properties (org-entry-properties))
↪                                     ↪ (filename (expand-file-name
↪                                     ↪ "index.org"
↪                                     ↪ (expand-file-name
↪                                     ↪ (plist-get ↪ ↪
↪ info :file-name) ↪ ↪
↪                                     ↪ (plist-get ↪ ↪
↪ info :base-dir))))
↪                                     ↪ (parent-pos
↪                                     ↪ (org-find-property
↪ "EXPORT_ELEVENTY_FILE_NAME"
↪                                     ↪ (org-entry-get-with-inheritance ↪ ↪
↪ "EXPORT_ELEVENTY_FILE_NAME"))
↪                                     ↪ body)
↪ (unless (string= (buffer-file-name)
↪                                     ↪ filename)
↪ (unless (file-directory-p (file-name-directory filename))
↪ (make-directory (file-name-directory filename) t))
↪ ;; find the heading that sets the current EXPORT_ELEVENTY_FILE_NAME
↪ (if parent-pos
↪ (save-excursion
↪ (goto-char parent-pos)
↪ (org-copy-subtree 1 (if do-cut 'cut)))
↪ (setq body (buffer-string)))
↪ (with-temp-file filename
↪ (org-mode)
↪ (if subtreep
↪ (progn
↪ (insert
↪ (or
↪ (mapconcat
↪ (lambda (o) (format "#+%s: %s" (car o) (cdr o)))
↪ file-properties
↪ "\n")
↪ ""
↪ "\n")
↪ (org-yank))
↪ (insert body))))))

```

Then this adds a link to it:

```
;; ###autoload
```

```

(defun sacha-org-export-filter-body-add-index-link (info)
  (when (and
          (plist-get info :file-name)
          (plist-get info :base-dir)
          (file-exists-p (expand-file-name
                          "index.org"
                          (expand-file-name
                           (plist-get info :file-name)
                           (plist-get info :base-dir))))))
    (goto-char (point-max))
    (insert
     (format "<div><a href=\"%sindex.org\">View Org source for this post</a></div>"
             (plist-get info :permalink)))))

```

```

(with-eval-after-load 'ox-11ty
  (add-to-list 'org-11ty-process-export-functions #'sacha-org-export-filter-body-add-index-link))

```

Then I want to wrap the whole thing up in an export function:

```

(defvar sacha-org-11ty-export-and-copy nil "*Non-nil means copy to site after specified delay ←
  ↳ (ex: \"5s\").")
(defvar sacha-org-11ty-export-and-copy-browse nil "Non-nil means browse after copying.")

;; ###autoload
(defun sacha-org-11ty-export (&optional async subtreep visible-only body-only ext-plist)
  (when (and subtreep (not (org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK"))
            (sacha-org-11ty-prepare-subtree))
    (let* ((info (org-11ty--get-info subtreep visible-only))
           (file (org-11ty--base-file-name subtreep visible-only))
           (permalink-slug (sacha-make-slug (plist-get info :permalink)))
           (org-html-footnotes-section
            (format
             "<div id=\"%s-footnotes\">\n<h3 ←
             ↳ class=\"%footnotes\">%s</h3>\n<div id=\"%s-text-footnotes\">\n\n%s\n</div>\n</div>"
             permalink-slug
             permalink-slug)))
      (unless (or (string= (plist-get info :input-file)
                           (expand-file-name
                            "index.org"
                            (expand-file-name
                             (plist-get info :file-name)
                             (plist-get info :base-dir))))
                  (plist-get (org-11ty--front-matter info) :no_source))
        (save-window-excursion
          (sacha-org-11ty-copy-subtree nil subtreep)))
      (org-11ty-export-to-11tydata-and-html async subtreep visible-only body-only ↳
        ↳ ext-plist)
    (when sacha-org-11ty-export-and-copy
      (message "%s" "Scheduling copy ... ")

```

```

(run-at-time sach-org-11ty-export-and-copy nil
  (lambda (url)
    (sach-org-11ty-copy-just-this-post
      url))
    (plist-get info :permalink))))))

;;;###autoload
(defun sach-org-11ty-export-and-copy (&rest args)
  "Export and copy to website."
  (let ((sach-org-11ty-export-and-copy "10"))
    (apply #'sach-org-11ty-export args)))

;;;###autoload
(defun sach-org-11ty-export-copy-browse (&rest args)
  "Export and copy to website."
  (let ((sach-org-11ty-export-and-copy "10")
        (sach-org-11ty-export-and-copy-browse t))
    (apply #'sach-org-11ty-export args)))

```

Now to figure out how to override the export menu. Totally messy hack!

```

(with-eval-after-load 'ox-11ty
  ;; Only on my computer
  (map-put (caddr (org-export-backend-menu (org-export-get-backend '11ty)))
    ?c (list "To Org, 11tydata.json, HTML" ←
    ↪ 'sach-org-11ty-export))
  (map-put (caddr (org-export-backend-menu (org-export-get-backend '11ty)))
    ?1 (list "...and copy to site" ←
    ↪ 'sach-org-11ty-export-copy-browse))
  )

```

Let's add a minor mode for automatically exporting.

```

;;;###autoload
(define-minor-mode sach-org-11ty-auto-export-mode
  ""
  :lighter ""
  (if sach-org-11ty-auto-export-mode
    (progn
      (setq-local sach-org-11ty-export-and-copy "10")
      (add-hook 'after-save-hook #'sach-org-11ty-export-and-copy ←
      ↪ nil t))
    (setq-local sach-org-11ty-export-and-copy nil)
    (remove-hook 'after-save-hook #'sach-org-11ty-export-and-copy t)))

```

```

;;;###autoload
(defun sach-org-11ty-update-modified ()
  "Update modified date."
  (interactive)
  (if (or (org-before-first-heading-p)
    (not (org-entry-get-with-inheritance ←
    ↪ "ELEVENTY_PERMALINK"))))
    (sach-org-set-file-property "MODIFIED" (format-time-string "%Y-%m-%d"))
    (save-excursion
      (goto-char (org-find-property "ELEVENTY_PERMALINK" ←
      ↪ (org-entry-get-with-inheritance "ELEVENTY_PERMALINK"))
      (org-entry-put (point) "MODIFIED" (format-time-string "%Y-%m-%d")))))

```


Listing exported Org posts

11TY:ORG:EMACS:BLOGGING Sometimes I want to

process those exported files, but exclude some posts that are mostly long lists of links (Emacs News, monthly/weekly/yearly posts). This function returns a list of possibly-relevant filenames.

```
;; ##autoload
(defun sachablog-org-files-except-reviews (after-date)
  "Return a list of recent .org files except for Emacs News and weekly/monthly/yearly reviews.
  AFTER-DATE is in the form yyyy, yyyy-mm, or yyyy-mm-dd."
  (setq after-date (or after-date "2020"))
  (let ((after-month (substring after-date 0 7))
        (posts (sachablog-posts)))
    (seq-keep
     (lambda (filename)
       (when (not (string-match "[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-emacs-news" filename))
         (when (string-match "/blog/\\([0-9]+\\)/\\([0-9]+\\)/" filename)
           (let ((month (match-string 2 filename))
                 (year (match-string 1 filename)))
             (unless (string> after-month
                              (concat year "-" month))
              (let ((info (sachablog-post-info-for-url (replace-regexp-in-string
                                                         "~~/proj/static-blog/\\index\\.org$\\|\\.org$" "" filename) posts)))
                (let-alist info
                  (when (and
                       info
                       (string> .date after-date)
                       (not (seq-intersection .categories
                                                '("emacs-news" "weekly" "monthly" "yearly")
                                                'string=)))
                    filename))))))))))
    (sort
     (directory-files-recursively "~/proj/static-blog/blog" "\\org$")
     :lessp #'string<
     :reverse t))))
```

Include Mastodon, HN, Reddit fields in front matter

```
;; ##autoload
(defun sachablog-org-11ty-add-mastodon-to-front-matter (front-matter info)
  (plist-put front-matter :mastodon (plist-get info :mastodon))
  (plist-put front-matter :hn (plist-get info :hn))
  (plist-put front-matter :reddit (plist-get info :reddit)))
```

```
(with-eval-after-load 'ox-11ty
  (cl-pushnew
   '(:mastodon "MASTODON" nil nil)
   (org-export-backend-options (org-export-get-backend '11ty)))
  (cl-pushnew
   '(:hn "HN" nil nil)
   (org-export-backend-options (org-export-get-backend '11ty)))
  (cl-pushnew
   '(:reddit "REDDIT" nil nil)
   (org-export-backend-options (org-export-get-backend '11ty)))
  (add-hook 'org-11ty-front-matter-functions
    (lambda ()
      (sachablog-org-11ty-add-mastodon-to-front-matter))))
```

Comments

```
;;###autoload
(defun sacha-11ty-add-blog-comment (new-comment url)
  "Add COMMENT to URL.
COMMENT should be an alist with author, date (ISO8901 format), and message (HTML)."
  (let* ((filename (sacha-11ty-comment-file url))
         (comments (sacha-11ty-comments url))
         (comment-list (alist-get 'comments (alist-get 'disqus ←
← comments))))
    (existing (and
← (alist-get ←
← 'postId new-comment)
← (seq-find ←
← (lambda (o)
← (string= (alist-get 'postId o)
← (alist-get ←
← 'postId new-comment)))
← comment-list))))
    (cond
      (existing
        ;; I think this is how you replace
        (setcar (member existing comment-list)
          new-comment))
      (comment-list
        (push new-comment (alist-get 'comments (alist-get 'disqus comments)))
        (cl-incf (alist-get 'commentCount (alist-get 'disqus comments))))
      (t
        (map-put! (alist-get 'disqus comments)
          'comments
          (list new-comment))
        (cl-incf (alist-get 'commentCount (alist-get 'disqus comments))))))
    (with-temp-file filename
      (insert
        (json-encode comments))
      (json-pretty-print (point-min) (point-max)))
    filename))
```

8.27.3 $\LaTeX$

```
(setq org-latex-listings t)
(setq org-export-with-tags 'not-in-toc)
(add-to-list 'org-latex-packages-alist '(" " "listings"))
(setq org-latex-listings-options
  '(("breaklines" "true")
    ("breakatwhitespace" "true")
    ("basicstyle" "\\small\\ttfamily")))
(setq org-latex-toc-include-unnumbered t)
(setq org-latex-pdf-process
  '("xelatex -shell-escape -interaction nonstopmode -output-directory %o %f"
    "xelatex -shell-escape -interaction nonstopmode -output-directory %o %f"
    "xelatex -shell-escape -interaction nonstopmode -output-directory %o %f"))
(setq org-latex-prefer-user-labels t)
```

8.27.4 TODO Copy Tasker task

```
;; ##autoload
(defun sachatasker-org-insert (url)
  (interactive "MTasknet URL: ")
  (let* ((parts (url-path-and-query (url-generic-parse-url url)))
         (params (url-parse-query-string (cdr parts)))
         (xml-url (format
                    ↵
                    ↵ "https://tasknet.com/_ah/api/datashare/v1/sharedata/%s/%s?a=0&xml=true"
                    ↵ (url-hexify-string (car ↵
                    ↵ (assoc-default "user" params 'string=)))
                    ↵ (url-hexify-string ↵
                    ↵ (replace-regexp-in-string ↵
                    ↵ "\\\+" " " ↵
                    ↵ (car (assoc-default "id" params ↵
                    ↵ 'string=))))))
        (json-object-type 'alist)
        (data (plz 'get
                   ↵
                   ↵ xml-url
                   ↵ :as
                   ↵ #'json-read))
        filename)
    (setf filename (expand-file-name (concat (alist-get 'fileName data)
                                              ↵
                                              ↵ ". "
                                              ↵
                                              ↵ (alist-get 'extension data)
                                              ↵
                                              ↵ ".xml")
                                      ↵
                                      ↵ sachadownload-dir))
      (with-temp-file filename
        (insert (alist-get 'shareData data)))
        (sachainsert-file-as-org-include filename)
        (insert (org-link-make-string url "Import via Tasknet")))))
```

8.27.5 **DONE** Remove heading from TOC

```
;;###autoload
(defun sacha-org-html-toc (depth info &optional scope)
  "Build a table of contents.

DEPTH is an integer specifying the depth of the table. INFO is
a plist used as a communication channel. Optional argument SCOPE
is an element defining the scope of the table. Return the table
of contents as a string, or nil if it is empty."

  (let ((toc-entries
        (mapcar (lambda (headline)
                  (cons (org-html--format-toc-headline headline info)
                        (org-export-get-relative-level headline info))))
              (org-export-collect-headlines info depth scope))))
    (when toc-entries
```

```

    (let* ((toc-id-counter (plist-get info :org-html--toc-counter))
           (toc (concat (format "<div class=\"text-table-of-contents toc-id%s\" ↵
↵ role=\"doc-toc\">"
                           (if toc-id-counter (format "-%d" toc-id-counter) "")
                           (org-html--toc-text toc-entries)
                           "</div>\n"))))
           (plist-put info :org-html--toc-counter (1+ (or toc-id-counter 0)))
           (if scope toc
               (let ((outer-tag (if (org-html--html5-fancy-p info)
                                     "nav"
                                     "div")))
                 (concat (format "<%s class=\"table-of-contents toc-id%s\" role=\"doc-toc\">\n"
                                outer-tag
                                (if toc-id-counter (format "-%d" toc-id-counter) ""))
                           ;; (let ((top-level (plist-get info ↵
                           ;; (format "<h%d>%s</h%d>\n"
                           ;; top-level
                           ;; (org-html--translate "Table of ↵
↵ Contents" info)
                           ;; top-level))
                           toc
                           (format "</%s>\n" outer-tag)))))))))

;; (with-eval-after-load 'org
;;   (advice-add 'org-html-toc :override #'sacha-org-html-toc))

```

8.27.6 Counting words without blocks

```

;;;###autoload
(defun sacha-org-simplify-text (text)
  "Don't include source blocks or links."
  (with-temp-buffer
    (insert text)
    (org-mode)
    (goto-char (point-min))
    (while (re-search-forward org-link-any-re nil t)
      (replace-match
        (if (save-match-data (string-match "audio\\|\\vtime\\|\\video" (match-string 0)))
            ""
            (or (match-string 3 "(link)")))) )
    (goto-char (point-min))
    (while (re-search-forward "^ *#\\|+begin" nil t)
      (let* ((block (org-element-context))
             (text
              (if (or (eq (org-element-type block) 'quote-block)
                    (string= (org-element-property :type block)
                          "media-post"))
                  (buffer-substring
                    (org-element-contents-begin block)
                    (org-element-contents-end block))
                  "(block)")))
            (delete-region (org-element-begin block)
                           ↵
                           (org-element-end block))
            (insert text "\n"))
      (while (re-search-forward "\n\n+" nil t)

```

```

                (replace-match "\n"))
            (string-trim
             (buffer-string))))

;;;###autoload
(defun sacha-org-subtree-text-without-blocks ()
  "Don't include source blocks or links. "
  (let (list)
    (save-excursion
      (save-restriction
        (org-back-to-heading)
        (org-narrow-to-subtree)

        (org-map-entries
         (lambda ()
           (push (buffer-substring-no-properties
                  (line-beginning-position)
                  (line-end-position))
                 list)
           (org-end-of-meta-data t)
           (unless (looking-at org-heading-regexp)
             (push
              (buffer-substring
               (point)
               (save-excursion
                 (or (outline-next-heading)
                     (point-max))))
              list))))
         nil 'tree)))
    (sacha-org-simplify-text (string-join (nreverse list) "\n"))))

;;;###autoload
(defun sacha-org-subtree-count-words-without-blocks ()
  (interactive)
  (let ((text (sacha-org-subtree-text-without-blocks)))
    (with-temp-buffer
      (insert text)
      (message "%s" (count-words--buffer-format)))))

;;;###autoload
(defun sacha-org-subtree-copy-words-without-blocks ()
  (interactive)
  (kill-new (sacha-org-subtree-text-without-blocks)))

```

8.27.7 Org Mode: Including portions of files between two regular expressions

org:emacs

- 2023-10-11 Wed: Include images inline.
- 2023-09-10: Use `consult-line` instead of `consult--line`.

I'd like to refer to snippets of code, but lines are too fragile to use as references for code and posts that I want to easily update. I'd like to specify a `from-regexp` and a `to-regexp` instead in order to collect the lines between those regexps (including the ones with the regexps themselves). `org-export-expand-include-keyword` looked a bit hairy to extend since it uses regular expressions to match parameter values. For this quick experiment, I decided to make a custom link type instead. This allows me to refer to parts of code with a link like this:

```
[[sacha-include:~/proj/static-blog/assets/css/style.css::from-regexp=Start of copy
code&to-regexp=End of copy code&wrap=src js]]
```

which will turn into this snippet from my stylesheet:

```
Start of copy code */
pre.src { margin: 0; line-height: 1 }
.org-src-container {
    position: relative;
    margin: 0;
    padding: 1.75rem 0 1.75rem 1rem;
}
div > details { padding: 20px; border: 1px solid var(--modus-border) }
.org-src-container > details { padding: 0; border: none }
details > .org-src-container { padding: 0; border: none }
summary { position: relative; }
summary .org-src-container { padding: 0 }
summary .org-src-container pre.src { margin: 0 }
.org-src-container button.copy-code, summary button.copy-code {
    position: absolute;
    top: 0;
    right: 0;
}
/*
```

Here's the Emacs Lisp code to do that. `sacha-include-complete` function reuses `sacha-include-open` to narrow to the file, and `sacha-include-complete` uses `consult-line` so that we can specify the prompt.

```
(org-link-set-parameters
 "sacha-include"
 :follow #'sacha-include-open
 :store #'sacha-include-store
 :export #'sacha-include-export
 :complete #'sacha-include-complete)
```

```
;; ##autoload
(defun sacha-include-open (path &optional _)
  "Narrow to the region specified in PATH."
  (require 'org-protocol)
  (let (params start end)
    (if (string-match "^\\(.*+?\\)\\(\\(?:::\\|\\|?\\|\\)\\(\\(.*+\\)\\)" path)
        (setq params (save-match-data <-
                      (org-protocol-convert-query-to-plist (match-string 2 path)))
          path (match-string 1 path)))
      (find-file path)
      (if (plist-get params :name)
          (when (org-babel-find-named-block (plist-get params :name))
            (goto-char (org-babel-find-named-block (plist-get <-
              params :name))))
          (let ((block (org-element-context)))
            (narrow-to-region (org-element-begin block)
                              <-
                              (org-element-end block))))
      (setq start
              (or
               (and
                (plist-get params :from-regexp)
                (progn
                 (goto-char (point-min))
```

```

                                (when (re-search-forward ↵
↵ (url-unhex-string (plist-get params :from-regexp)))
                                ↵
↵ (line-beginning-position))))
                                (progn
                                (goto-char (point-min))
                                (point))))
                                (setq end
                                (or
                                (and
                                (plist-get params :to-regexp)
                                (progn
                                (when (re-search-forward ↵
↵ (url-unhex-string (plist-get params :to-regexp)))
                                (line-end-position))))
                                (progn
                                (goto-char (point-max))
                                (point))))
                                (when (or (not (= start (point-min)))
                                (not (= end (point-max))))
                                (narrow-to-region start end))))))

;;;###autoload
(defun sacha-include-store ()
  "Store a link to Org Babel named blocks."
  (when-let ((elem (and (derived-mode-p 'org-mode) (org-element-at-point))))
    (when (and (org-element-type-p elem 'src-block)
                (org-element-property :name elem))
      (org-link-store-props :type "sacha-include"
                            :link (concat "sacha-include:" (buffer-file-name) "?name="
                                             (org-element-property :name elem))
                            :text (org-entry-get (point) "ITEM"))))

;;;###autoload
(defun sacha-include-export (path _ format _)
  "Export PATH to FORMAT using the specified wrap parameter."
  (require 'org-protocol)
  (let (params body start end)
    (when (string-match "^\\(.*+?\\|\\|\\(?:::\\|\\|\\|?\\|\\|\\(.*+\\|\\)" path)
      (setq params (save-match-data (org-protocol-convert-query-to-plist ↵
↵ (match-string 2 path)))
      path (match-string 1 path)))
    (with-temp-buffer
      (insert-file-contents-literally path)
      (when (string-match "\\|.org$" path)
        (org-mode))
      (if (plist-get params :name)
          (when (org-babel-find-named-block (plist-get params ↵
↵ :name))
          (goto-char (org-babel-find-named-block ↵
↵ (plist-get params :name)))
          (let ((block (org-element-context)))
            (setq start (org-element-begin block)
                    end ↵
↵ (org-element-end block))))
          (goto-char (point-min))
          (when (plist-get params :from-regexp)
            (re-search-forward (url-unhex-string (plist-get params ↵
↵ :from-regexp))))

```

```

                                (goto-char (match-beginning 0)))
                                (setq start (point))
                                (setq end (point-max))
                                (when (plist-get params :to-regexp)
                                  (re-search-forward (url-unhex-string (plist-get params ↵
↵ :to-regexp)))
                                (setq end (match-beginning 0)))
                                (setq body (buffer-substring start end)))
                                (with-temp-buffer
                                  (when (plist-get params :wrap)
                                    (let* ((wrap (plist-get params :wrap))
                                             block args)
                                      (when (string-match "\\<\\(\\S-+\\)\\( +.\\)*?" wrap)
                                        (setq block (match-string 1 wrap))
                                        (setq args (match-string 2 wrap))
                                        (setq body (format "#+BEGIN_%s%s\n%s\n#+END_%s\n"
↵
↵ block (or args ""))
↵
↵ body
↵ block))))
                                  (when (plist-get params :summary)
                                    (setq body (format "#+begin_my_details ↵
↵ %s\n%s\n#+end_my_details\n"
↵
↵ (plist-get params :summary)
↵
↵ body)))
                                  (insert body)
                                  (message "BODY: %s" body)
                                  (org-export-as format nil nil t)))

;;###autoload
(defun sacha-include-complete ()
  "Include a section of a file from one line to another, specified with regexps."
  (interactive)
  (require 'consult)
  (let ((file (read-file-name "File: ")))
    (save-window-excursion
      (find-file file)
      (concat "sacha-include:"
↵
↵ file
↵ "?from-regexp="
↵ (let ((curr-line (line-number-at-pos
↵
↵ (point)
↵
↵ consult-line-numbers-widen))
↵ (prompt "From ↵
↵ line: "))
↵
↵ (goto-char (point-min))
↵ (consult-line)
↵ (url-hexify-string
↵ (regexp-quote ↵
↵ (buffer-substring (line-beginning-position) (line-end-position))))
↵ "to-regexp="
↵ (let ((curr-line (line-number-at-pos
↵
↵

```



```

↪                                (point)                                ↪
↪                                consult-line-numbers-widen))          ↪
↪                                (prompt "To ↪
↪ line: "))
                                (goto-char (point-min))
                                (consult-line
                                nil (point))
                                (url-hexify-string
                                (regexp-quote ↪
↪ (buffer-substring (line-beginning-position) (line-end-position))))
                                "&wrap=src " (replace-regexp-in-string ↪
↪ "-mode$" "" (symbol-name major-mode))))))

```

This code displays the images inline.

```

(eval-and-compile
  (require 'org-macs nil t))

;;###autoload
(defun sach-org-display-included-images (&optional include-linked refresh beg end)
  "Display inline images for sach-org-include types."
  (interactive "P")
  (when (display-graphic-p)
    (when refresh
      (org-remove-inline-images beg end)
      (when (fboundp 'clear-image-cache) (clear-image-cache)))
    (let ((end (or end (point-max))))
      (org-with-point-at (or beg (point-min))))
      (let* ((case-fold-search t)
              (file-extension-re "\\\\.svg")
              (file-types-re (format "\\[\\[sach-org-include:"))
              (while (re-search-forward file-types-re end t)
                (let* ((link (org-element-lineage (save-match-data ↪
↪ (org-element-context)) 'link t))
                                (inner-start (match-beginning 1))
                                (path
                                  (cond
                                    ((not link) nil)
                                    ;; file link without a ↪
                                  ((or (not ↪
↪ (org-element-contents-begin link)) include-linked)
                                (org-element-property ↪
↪ :path link))
                                ((not inner-start) nil)
                                (t (org-with-point-at ↪
↪ inner-start
                                (and ↪
↪ (looking-at
                                (if (char-equal ?< (char-after inner-start))
                                ↪
                                ↪
                                org-link-angle-re
                                ↪
                                ↪
                                org-link-plain-re))
                                ↪
                                ;; File name must fill the whole

```

```

↪      ;; description.
↪
↪      (= (org-element-contents-end link)
↪
↪          (match-end 0))
↪
↪      (progn
↪          (setq linktype (match-string 1))
↪          (match-string 2))))))
↪          (when (string-match "\\(\\.+\\)\\?" path)
↪              (setq path (match-string 1 path)))
↪          (when (and path (string-match-p file-extension-re path))
↪              (let ((file (expand-file-name path)))
↪                  ;; Expand environment variables.
↪                  (when file (setq file (substitute-in-file-name file)))
↪                  (when (and file (file-exists-p file))
↪                      (let ((width ←
↪
↪      (org-display-inline-image--width link))
↪
↪          (old ←
↪
↪      (get-char-property-and-overlay
↪
↪          (org-element-begin link)
↪
↪          'org-image-overlay)))
↪          (if (and (car-safe ←
↪
↪      old) refresh)
↪              (image-flush (overlay-get (cdr old) 'display))
↪              (let ((image ←
↪
↪      (org--create-inline-image file width)))
↪              (when ←
↪
↪      image
↪
↪          (let ((ov (make-overlay
↪
↪          (org-element-begin link)
↪
↪          (progn
↪              (goto-char
↪
↪          (org-element-end link))
↪              (skip-chars-backward " \t")
↪              (point))))))
↪              ;; FIXME: See bug#59902. We cannot rely
↪              ;; on Emacs to update image if the file
↪              ;; has changed.
↪              (image-flush image)
↪              (overlay-put ov 'evaporate t)
↪
↪      (overlay-put ov 'display image)
↪
↪      (overlay-put ov 'face 'default)
↪
↪      (overlay-put ov 'org-image-overlay t)
↪
↪      (overlay-put

```

```

↪          ov 'modification-hooks ↪
↪          (list 'org-display-inline-remove-overlay)) ↪
↪          (when (boundp 'image-map) ↪
↪              (overlay-put ov 'keymap image-map)) ↪
↪          (push ov org-inline-image-overlays)))))))))))))

```

8.27.8 ox-epub

```

(use-package ox-epub
  :if sachalaptop-p
  :defer t
  :config
  (setq org-epub-style-default
    (concat org-epub-style-default "\n p.sacha-verse { white-space: pre }\n")))

```

8.27.9 DONE Add a note to the bottom of blog posts exported from my config file

Update: 2021-04-18: Tweaked the code so that I could add it to the main `org-export-filter-body-functions` list now that I'm using Eleventy and ox-11ty.el instead of Wordpress and org2blog.

I occasionally post snippets from my Emacs configuration file, drafting the notes directly in my literate config. I figured it might be a good idea to include a link to my config at the end of the posts, but I didn't want to scatter redundant links in my config file itself. Wouldn't it be cool if the link could be automatically added whenever I post a subtree from my config file? I think the code below accomplishes that.

```

;;;###autoload
(defun sachalorg-export-filter-body-add-emacs-configuration-link (string backend info)
  (when (and (member backend '(md html 11ty)) (plist-get info :input-file) (string-match ↪
    ↪ "\\\\.emacs\\.d/Sacha\\.org\\|sync/emacs/Sacha\\.org" (plist-get info :input-file)))
    (concat string
      (let ((id (org-entry-get-with-inheritance ↪
        ↪ "CUSTOM_ID"))))
        (format
          (if (eq backend 'md)
              "\nThis is part of my ↪
        ↪ [Emacs configuration](https://sachachua.com/dotemacs%s)\n"
              "\n<div class=\"note\">This ↪
        ↪ is part of my <a href=\"https://sachachua.com/dotemacs%s\">Emacs ↪
        ↪ configuration.</a></div>")
          (if id (concat "#" id) ""))))))

```

```

(with-eval-after-load 'ox
  (add-to-list 'org-export-filter-body-functions ↪
    ↪ #'sachalorg-export-filter-body-add-emacs-configuration-link))

```

8.27.10 Copy linked file and change link

```

;;;###autoload
(defun sacha-org-copy-linked-file-and-change-link (destination)
  (interactive (list
    (cond
      ((and (not
        ↪ current-prefix-arg) (file-directory-p "images/"))
        "images/")
      ↪
        ↪ ((org-entry-get-with-inheritance "EXPORT_ELEVENTY_FILE_NAME")
        (expand-file-name
        ↪
        ↪ (org-entry-get-with-inheritance "EXPORT_ELEVENTY_FILE_NAME")
        (cadar ↪
        ↪ (org-collect-keywords '("ELEVENTY_BASE_DIR")))))
        (t
        (read-file-name ↪
        ↪
        ↪
        ↪ (file-name-nondirectory (org-element-property :path (org-element-context))))))
        (let* ((elem (org-element-context))
        (path (org-element-property :path elem))
        (description (org-element-property :description elem))
        (copy-file path destination t)
        (delete-region (org-element-begin elem) (org-element-end elem))
        (insert (org-link-make-string
        (concat "file:"
        ↪
        ↪ (file-relative-name
        ↪ (if ↪
        ↪ (file-directory-p destination)
        ↪
        ↪ (expand-file-name (file-name-nondirectory path)
        ↪
        ↪
        ↪ destination)
        ↪
        ↪ destination)))
        description))))

;;;###autoload
(defun sacha-embark-org-copy-linked-file-and-change-link (url destination)
  (interactive (list
    (read-string "Link: ")
    (read-file-name (format "Copy ↪
    ↪ %s to: "
    ↪
    ↪
    ↪ (file-name-nondirectory (org-element-property :path (org-element-context))))))
    (sacha-org-copy-linked-file-and-change-link destination))

;;;###autoload
(defun sacha-org-copy-linked-files (destination beg end)
  (interactive (list
    (if (and (not ↪
    ↪ current-prefix-arg) (file-directory-p "images/"))
    "images/"

```

```

↪ (format "Copy %s to: "                                (read-file-name ↪
                                                         ↪
                                                         ↪
↪ (file-name-nondirectory (org-element-property :path (org-element-context))))))
                                                         (if (region-active-p) ↪
↪ (region-beginning) (point-min))
                                                         (if (region-active-p) ↪
↪ (region-end) (point-max)))
  (goto-char beg)
  (while (re-search-forward "\\(file\\):" end t)
    (let* ((elem (org-element-context))
            (path (org-element-property :path elem))
            (description (org-element-property :description elem)))
      (when path
        (unless (string-match (concat "^" (regexp-quote ↪
↪ (expand-file-name destination)))) (expand-file-name path))
        (sacha-org-copy-linked-file-and-change-link ↪
↪ destination))))))

```

```

(with-eval-after-load 'embark-org
  (keymap-set embark-org-link-map "r l" ↪
  ↪ #'sacha-embark-org-copy-linked-file-and-change-link))

```

8.27.11 Org Mode: Create a quick timestamped note and capture a screenshot

emacs:org

I wanted to be able to quickly create timestamped notes and possibly capture a screenshot. Prompting for a value inside an `org-capture-template` disrupts my screen a little, so maybe this will make it as easy as possible. I could probably do this without going through `org-capture-templates`, but I wanted to take advantage of the fact that Org Mode will deal with the date tree and finding the right position itself.

This uses the `sacha-combined-screenshots` and `sacha-latest-screenshot` functions from [Using image-dired to browse the latest screenshots from multiple directories](#).

```

;;;###autoload
(defun sacha-org-insert-screenshot (file &optional note)
  (interactive (list
    (if current-prefix-arg
      (consult--read
        (sacha-combined-screenshots)
        :sort nil
        :require-match t
        :category 'file)
      ↪
    ↪ (sacha-latest-screenshot))))
  (cond
    ((derived-mode-p 'mastodon-toot-mode)
     (mastodon-toot--attach-media file (or note (read-string "Caption: "))))
    ((derived-mode-p 'subed-mode)
     (insert "NOTE\n" (org-link-make-string (concat "file:" file)) "\n"))
    (t
     (save-window-excursion
      (if (string-match "webm" file)
        (progn
          (mpv-play file)
          (insert "#+begin_media-post\n"

```

```

(org-link-make-string (concat "video:" file "?caption=" (or note ↵
↵ (read-string "Caption: "))))) "\n"
      "#+end_media-post\n"))
      (with-current-buffer (find-file-noselect file) (display-buffer ↵
↵ (current-buffer)))
      (insert "#+CAPTION: " (or note (read-string "Caption: ")) "\n"
      (org-link-make-string (concat ↵
↵ "file:" file)
      (concat "file:" file)
      )))))))

;;;###autoload
(defun sacha-copy-last-screenshot-to-file (new-filename)
  (interactive (list (read-file-name (format "Copy %s to: " (file-name-nondirectory ↵
↵ (sacha-latest-screenshot)))))
  (copy-file (sacha-latest-screenshot) new-filename))

;;;###autoload
(defun sacha-copy-last-screenshot-and-insert-into-org (new-filename caption)
  (interactive (list (read-file-name (format "Copy %s to: " (file-name-nondirectory ↵
↵ (sacha-latest-screenshot)))))
  (read-string "Caption: "))
  (copy-file (sacha-latest-screenshot) new-filename t)
  (insert "#+CAPTION: " caption "\n"
  (org-link-make-string (concat "file:" (file-relative-name new-filename))) "\n"))

;;;###autoload
(defun sacha-convert-latest-recording ()
  (interactive)
  (let* ((latest (expand-file-name (sacha-latest-screenshot)))
  (pcase (file-name-extension latest)
    ("mkv"
      (message "Converting %s ..." latest)
      (let ((new-file (new-file (concat (file-name-sans-extension latest) ".webm"))))
        (make-process :name "ffmpeg"
          :buffer "*ffmpeg*"
          :command
            (list "ffmpeg" "-i" latest "-y" new-file)
          :sentinel
            (lambda (proc status)
              (when (string-match "finished" status)
                (message "%s done." new-file))))))
      ("svg"
        (let ((new-file (concat (file-name-sans-extension latest) ".png")))
          (call-process "convert" nil nil nil latest new-file)
          (kill-new new-file)
          (message "Converted %s to %s" latest new-file))))))

;;;###autoload
(defun sacha-org-capture-prefill-template (template &rest values)
  "Pre-fill TEMPLATE with VALUES."
  (setq template (or template (org-capture-get :template)))
  (with-temp-buffer
    (insert template)
    (goto-char (point-min))
    (while (re-search-forward
      (concat "%\\("
      "\\[[\\(.+\\)\\]\\|\\|")

```

```

" <\\([^\n]+\>\\| "
" \\([tTuUaLiAcxkKInfF]\\|\\| "
" \\(:[-a-zA-Z]+\\|\\| "
" \\^\\({\\([^\n]}*\\|\\| "
" ?\\([gGtTuUCLp]\\|\\| ?\\| "
" %\\|\\|\\|([1-9][0-9]*\\| "
" \\| " nil t)

(if (car values)
    (replace-match (car values) nil t))
(setq values (cdr values)))
(buffer-string)))

;;;###autoload
(defun sacha-capture-timestamped-note (time note)
  "Disable Helm and capture a quick timestamped note."
  (interactive (list (current-time) (read-string "Note: ")))
  (let ((helm-completing-read-handlers-alist '((org-capture . nil)))
        (entry (org-capture-select-template "p")))
    (org-capture-set-plist entry)
    (org-capture-get-template)
    (org-capture-set-target-location)
    (org-capture-put
     :template (org-capture-fill-template
                  (sacha-org-capture-prefill-template (org-capture-get :template)
                                                         (format-time-string "%H:%M:%S,%3N"
                                                         note))))
    (org-capture-place-template)
    (org-capture-finalize)))

;;;###autoload
(defun sacha-capture-timestamped-note-with-screenshot (time note)
  "Include a link to the latest screenshot."
  (interactive (list (current-time) (read-string "Note: ")))
  (kill-new (sacha-latest-screenshot))
  (sacha-capture-timestamped-note time note))

```

8.27.12 Cleaning up export

Timestamps and section numbers make my published files look more complicated than they are. Let's turn them off by default, and let's use fancy HTML5.

```

(setq org-html-doctype "html5")
(setq org-html-html5-fancy t)
(setq org-export-with-section-numbers nil)
(setq org-html-include-timestamps nil)
(setq org-export-with-sub-superscripts nil)
(setq org-export-with-toc nil)
(setq org-html-toplevel-hlevel 2)
(setq org-export-htmlize-output-type 'css)
(setq org-export-with-broken-links t)
(setq org-ascii-text-width 10000)
(setq-default tab-width 2)
(setq org-publish-project-alist
  '(("stream"
    :base-directory "~/proj/stream"
    )
    ("emacs-config"
    :base-directory "~/.config/emacs"

```

```

:publishing-directory "~/config/emacs"
:publishing-function sacha-org-html-publish-to-html-trustingly
)
("book-notes"
:base-directory "c:/sacha/Dropbox/books"
:publishing-directory "c:/sacha/Dropbox/books/html"
:publishing-function sacha-org-html-publish-to-html-trustingly
:makeindex t)

("topics"
:base-directory "~/sync/topics"
:publishing-directory "/tmp/topics"
:publishing-function sacha-org-11ty-publish-from-project)))

```

This makes it easier to publish my files:

```

(defun sacha-org-11ty-publish-from-project (_ from-file _)
  (with-current-buffer (find-file-noselect from-file)
    (save-excursion
      (goto-char (point-min))
      (when (save-excursion (re-search-forward "ELEVENTY_PERMALINK" nil t))
        (sacha-org-11ty-export))))))
; (load "~/proj/dev/emacs-chats/build-site.el" t)
; (load "~/proj/dev/emacs-notes/build-site.el" t)

```

If a file is in a publishing project, publish it.

```

;;;###autoload
(defun sacha-org-publish-maybe ()
  (require 'ox-publish)
  (interactive)
  (save-excursion
    (if (org-publish-get-project-from-filename
        (buffer-file-name (buffer-base-buffer))) 'up)
        (org-publish-current-file t)
        (sacha-org-html-export-trustingly))))

```

Make it easy to publish and browse a file.

```

;;;###autoload
(defun sacha-org-publish-and-browse ()
  (interactive)
  (save-buffer)
  (sacha-org-publish-maybe)
  (browse-url (org-export-output-file-name ".html" nil default-directory)))

```

(I'm not sure I have an Apps keybinding at the moment...)

```

(keymap-global-set "<apps> b" 'sacha-org-publish-and-browse)

```

8.27.13 Publish without prompting

I want to be able to export without having to say yes to code blocks all the time.

```

;;;###autoload
(defun sacha-org-html-export-trustingly ()

```



```
(interactive)
(let ((org-confirm-babel-evaluate nil))
  (org-html-export-to-html)))

;;###autoload
(defun sacha-org-html-publish-to-html-trustingly (plist filename pub-dir)
  (let ((org-confirm-babel-evaluate nil))
    (org-html-publish-to-html plist filename pub-dir)))
```

8.27.14 Special blocks

Example markup:

```
#+begin_my_details Summary of details block
Details that will be displayed when the summary is clicked.
#+end_my_details
```

```
(use-package org-special-block-extras
  :if sacha-laptop-p
  :hook (org-mode . org-special-block-extras-mode)
  :init (setq org-special-block-add-html-extra nil)
  :config
  ;; Use short names like 'defblock instead of the fully qualified name
  ;; 'org-special-block-extras--defblock
  (setcdr org-special-block-extras-mode-map nil)
  (org-defblock my_details (title "Details" title-color "Green" open "")
    "Top level (HTML & 11ty)OSPE-RESPECT-NEWLINES? Enclose contents in a folded up box."
    (message "my_details %s %s %s" title title-color open))
  (cond
    ((eq backend '11ty)
     (format
      "{% details \"%s\" %s%}\n%s\n{% enddetails %}"
      title (if (string= open "") "" " ", \"open\"") contents))
    ((eq backend 'html)
     (format
      "<details class=\"code-details\"
        style =\"padding: 1em;
          border-radius: 15px;
          font-size: 0.9em;
          box-shadow: 0.05em 0.1em 5px 0.01em #00000057;\"%s>
        <summary>
          <strong>
            <font face=\"Courier\" size=\"3\" color=\"%s\">
              %s
            </font>
          </strong>
        </summary>
        %s
      </details>"
      (if (string= open "") "" " open") title-color title contents))))
  (defalias 'org-block/details #'org-block/my_details))

(org-defblock
  columns nil nil
  "Top level (HTML & wp & 11ty)OSPE-RESPECT-NEWLINES? Split into columns using ↵
↵ Foundation."
  (if (member backend '(html 11ty))
```

```

                (format "<div class=\"row\">%s</div>" contents)
            contents))
(org-defblock
  column50 nil nil
  "Top level (HTML & wp & 11ty)OSPE-RESPECT-NEWLINES? Split into columns."
  (if (member backend '(html 11ty))
      (format "<div class=\"columns small-12 medium-6 large-6\">%s</div>"
              contents)
      contents))
  (org-defblock short (yt nil video nil audio nil thumbnail nil)
    "Top level (HTML &
    ↪ 11ty)OSPE-RESPECT-NEWLINES? Mark up a YouTube short."
    (let ((yt-link (and yt (format ↪
    ↪ "<a href=\"https://youtube.com/watch?v=%s\">watch this on YouTube</a>"
    ↪
    ↪
    ↪ (sacha-org-yt-id yt))))
    ↪
    ↪ (video-link (and video
    ↪
    ↪ (format "<a ↪
    ↪ href=\"%s\">download the video</a>"
    ↪
    ↪
    ↪ (org-export-file-uri video))))
    ↪
    ↪ (audio-link (and audio
    ↪
    ↪ (format "<a ↪
    ↪ href=\"%s\">download the audio</a>"
    ↪
    ↪
    ↪ (org-export-file-uri audio))))
    (concat
      "<div ↪
    ↪ class=\"row\"><div class=\"columns\"><div style=\"width: 400px\">"
      (if video
        ↪
        ↪ (sacha-org-video-export (concat "video:" (expand-file-name video) "?thumbnail=" (or ↪
        ↪ thumbnail ""))
        ↪
        ↪ nil backend nil)
        ↪
        ↪ (sacha-org-yt-export yt nil backend nil)
        ↪ "</div></div><div ↪
    ↪ class=\"columns\">"
        contents
        "<p>You can "
        (cond
          ((and yt-link ↪
        ↪ video-link audio-link) (format "%s, %s, or %s." yt-link video-link audio-link))
          ((and yt-link ↪
        ↪ video-link) (format "%s or %s." yt-link video-link))
          ((and yt-link ↪
        ↪ audio-link) (format "%s or %s." yt-link audio-link))
          ((and ↪
        ↪ video-link audio-link) (format "%s or %s." video-link audio-link))
          (video-link ↪

```

```

↳ (format "%s." video-link))

                                          (audio-link ↵
↳ (format "%s." audio-link)))

                                          "</p></div></div>"))))
      (org-defblock visual_book_note (title nil post nil image nil)
        "Top level (HTML & ↵
↳ 11ty)OSPE-RESPECT-NEWLINES? Mark up a visual book note thumbnail."
        (format
          "<figure class=\"book\">
<a href=\"%s\">
<div><img src=\"%s\" alt=\"%s\" /></div>
<figcaption>%s</figcaption>
</a>
</figure>"
          post image title title))
      (org-defblock gallerylist ()
        "Top level (HTML & ↵
↳ 11ty)OSPE-RESPECT-NEWLINES? Mark up a visual book note thumbnail."
        (if (eq backend '11ty)
          (concat "{% ↵
↳ gallerylist %}\n" contents "\n{% endgallerylist %}\n")
          (concat "<div ↵
↳ class=\"gallerylist\">" contents "</div>"))))

```

And here's a little thing to convert a two-level list into my collapsible sections:

```

;; ###autoload
(defun sacha-org-convert-list-to-collapsible-details ()
  (interactive)
  (let ((list (org-list-to-lisp t)))
    (mapc (lambda (o)
      (when (stringp (car o))
        (insert
          (format
            "#+begin_my_details %s :open ↵
↳ t\n%s#+end_my_details\n"
            (car o)
            (mapconcat
              (lambda (s)
                (concat "- " ↵
↳ (string-trim (org-ascii--indent-string (car s) 2)) "\n"))
              (cdr (cadr o))))))
        (cdr list))))

```

8.27.15 Abbreviations

There's an `abbr` HTML tag that I can use to provide inline abbreviations and definitions.

```

;; ###autoload
(defun sacha-org-abbr-export (path desc backend info)
  "Export abbr links for Org mode.
PATH is the expansion/title.
DESC is the abbreviation text (optional).
BACKEND is the export backend.
INFO is a plist holding contextual information."
  (pcase backend
    ;; HTML export

```

```

((or 'html '11ty)
 (if desc
  (format "<abbr title=\"%s\" tabindex=\"%0\">%s</abbr>"
    (org-html-encode-plain-text path)
    (org-html-encode-plain-text desc))
  (format "<abbr>%s</abbr>"
    (org-html-encode-plain-text path))))
('org
 (org-link-make-string (concat "abbr:" path) desc))
;; LaTeX export
('latex
 (if desc
  (format "\\abbr[%s]{%s}"
    (org-latex-encode-plain-text path)
    (org-latex-encode-plain-text desc))
  (format "\\abbr{%s}"
    (org-latex-encode-plain-text path))))

;; ASCII/plain text export
('ascii
 (if desc
  (format "%s (%s)" desc path)
  path))

;; Default for other backends
(_
 (if desc
  (format "%s (%s)" desc path)
  path))))

```

```

(with-eval-after-load 'org
  (org-link-set-parameters "abbr" :export #'(sacha-org-abbr-export)))

```

8.27.16 Adding a custom header argument to Org Mode source blocks and using that argument during export org:emacs

I sometimes want to put long source blocks in a `<details><summary>... </summary>... </details>` block when I export to HTML, so that they're tucked away in a collapsible block. I tried using <https://github.com/alhassy/org-special-block-extras> to define my own `#+begin_my_details "summary text" ... #+end_my_details` block, but source blocks inside `my_details` doesn't get font-locked properly while in the Org file. I wanted to add a `:summary` attribute to the regular `src` blocks, and to change the HTML export to wrap the code in `details` if the summary was specified.

```

(eval-and-compile
 (require 'org-macs nil t))
;; ##autoload
(defun sacha-org-html-src-block (src-block _contents info)
  (let* ((result
          (org-html-src-block
            src-block
            ;; todo: apply filter functions
            _contents
            info))
    (block-info
      (org-with-point-at (org-element-property :begin ↵
        ↵ src-block)

```

```

                                (org-babel-get-src-block-info)))
                                (summary (assoc-default :summary (elt block-info 2))))
    (if (member summary '("%summary" ""))
        result
        (format "<details><summary>%s</summary>%s</details>"
                  summary
                  result))))

;; ##autoload
(defun sacha-org-11ty-src-block (src-block _contents info)
  (let* ((result (org-11ty-src-block src-block _contents info))
         (block-info
          (org-with-point-at (org-element-property :begin ↵
↵ src-block)
                                (org-babel-get-src-block-info)))
         (summary (assoc-default :summary (elt block-info 2))))
    (if (member summary '("%summary" ""))
        result
        (format "<details><summary>%s</summary>%s</details>"
                  summary
                  result))))

```

```

(setq org-babel-exp-code-template "#+begin_src %lang%switches%flags :summary ↵
↵ %summary\n%body\n#+end_src")
(with-eval-after-load 'ox-html
  (map-put!
   (org-export-backend-transcoders (org-export-get-backend 'html))
   'src-block 'sacha-org-html-src-block))
(with-eval-after-load 'ox-11ty
  (map-put!
   (org-export-backend-transcoders (org-export-get-backend '11ty))
   'src-block 'sacha-org-11ty-src-block))

```

So now I can use it by specifying blocks like this:

```

#+begin_src emacs-lisp :summary "Code for adding a :summary argument and using it during export"
;; code goes here
#+end_src

```

It took me a bit of digging around to figure this out. When I added the `:summary` attribute, `org-babel-get-src-block-info` found it when I was in the Org file, but by the time `sacha-org-html-src-block` was called, the block had been replaced with a copy that didn't have the header argument. I dug around using edebug's `d` command for displaying the backtrace, stepping through various functions. I found out that in the process for exporting source code blocks, `org-babel-exp-code` replaces the source block with the value of `org-babel-exp-code-template`, substituting certain values. Adding the `summary` flag to that and retrieving the summary information using `org-babel-get-src-block-info` worked. I originally used `advice-add` to override `org-html-src-block`, but I think I'll try replacing the transcoder.

Adding custom header arguments could be useful for different export-related tweaks (someone wanted to [create an argument for highlighting certain lines](#) but hadn't figured it out in that thread). If there's a more elegant way to do this, I'd love to find out!

8.27.17 Stylesheet / header

Might as well take advantage of my stylesheet:

```
(setq org-html-head "
  <link rel=\"stylesheet\" type=\"text/css\" ↵
  ↵ href=\"https://sachachua.com/assets/css/style.css\"></link>
  <link rel=\"stylesheet\" type=\"text/css\" ↵
  ↵ href=\"https://sachachua.com/assets/css/org-export.css\"></link>
  <script ↵
  ↵ src=\"https://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js\"></script>")
(setq org-html-htmlize-output-type 'css)
(setq org-src-fontify-natively t)
```

8.27.18 Footer

Make it easy to scroll to the top:

```
(setq org-html-preamble "<a name=\"top\" id=\"top\"></a>")
(setq org-html-postamble "
  <style type=\"text/css\">
    .back-to-top {
      position: fixed;
      bottom: 2em;
      right: 0px;
      text-decoration: none;
      color: #000000;
      background-color: rgba(235, 235, 235, 0.80);
      font-size: 12px;
      padding: 1em;
      display: none;
    }

    .back-to-top:hover {
      background-color: rgba(135, 135, 135, 0.50);
    }
  </style>

  <div class=\"back-to-top\">
    <a href=\"#top\">Back to top</a> | <a href=\"mailto:sacha@sachachua.com\">E-mail me</a>
  </div>

  <script type=\"text/javascript\">
    var offset = 220;
    var duration = 500;
    jQuery(window).scroll(function() {
      if (jQuery(this).scrollTop() > offset) {
        jQuery('.back-to-top').fadeIn(duration);
      } else {
        jQuery('.back-to-top').fadeOut(duration);
      }
    });
  </script>
  <script src=\"https://sachachua.com/assets/js/misc.js\"></script>")
```

8.27.19 Copy region

Sometimes I want a region's HTML in my kill-ring/clipboard without any of the extra fluff:

```
; ; ; ##autoload
```

```
(defun sacha-org-copy-region-as-html (beg end &optional level)
  "Make it easier to copy code for Wordpress posts and other things."
  (interactive "r\np")
  (let ((org-export-html-preamble nil)
        (org-html-toplevel-hlevel (or level 3)))
    (kill-new
      (org-export-string-as (buffer-substring beg end) 'html t))))
```

Sometimes I want a subtree:

```
;;###autoload
(defun sacha-org-copy-subtree-as-html ()
  (interactive)
  (sacha-org-copy-region-as-html
   (org-back-to-heading)
   (org-end-of-subtree)))
```

8.27.20 UTF-8 checkboxes

This snippet turns - [X] into and - [] into , but leaves [-] alone.

```
(setq org-html-checkbox-type 'unicode)
(setq org-html-checkbox-types
  '((unicode (on . "<span class=\"task-done\">6#x2611;</span>")
              (off . "<span class=\"task-todo\">8#x2610;</span>")
              (trans . "<span class=\"task-in-progress\">[-]</span>"))))
```

8.27.21 Beamer

```
(with-eval-after-load 'org
  (require 'ox-latex)
  (add-to-list 'org-latex-classes
    '("beamer"
      "\\documentclass[presentation]{beamer}"
      ("\\section{%s}" . "\\section*{%s}")
      ("\\subsection{%s}" . "\\subsection*{%s}")
      ("\\subsubsection{%s}" . "\\subsubsection*{%s}"))
    (add-to-list 'org-latex-classes
      '("memoir"
        "\\documentclass{memoir}"
        ("\\section{%s}" . "\\section*{%s}")
        ("\\subsection{%s}" . "\\subsection*{%s}")
        ("\\subsubsection{%s}" . "\\subsubsection*{%s}")))))
```

8.27.22 PlantUML

```
(setq org-plantuml-jar-path (expand-file-name "/usr/share/plantuml/plantuml.jar"))
(add-to-list 'org-src-lang-modes '("plantuml" . plantuml))
```

8.27.23 ox-hugo

```
(use-package ox-hugo
```

```

:ensure t           ;Auto-install the package from Melpa (optional)
:defer t
:after ox)

```

8.27.24 Org Mode: Asynchronous export and tangle of a large file

org

I have a pretty large [Emacs configuration file](#). It's annoying to wait 11 seconds for it to export to HTML or 12 seconds to tangle. Fortunately, Org Mode allows me to export asynchronously. I tried it out from `org-export-dispatch` (`C-c C-e`) by using the `C-a` option. It worked pretty well, but it was a bit slow because it loaded my full configuration. Fortunately, there's a way to use a smaller configuration that focuses on just the packages needed.

```

(setq org-export-async-init-file "~/config/emacs/org-async-export-config.el")
(setq org-export-async-debug t)

```

I've named the source blocks, and this block assembles the config from those named blocks by using `noweb`.

```

<<org-async-variables>>
<<startup>>
<<system-info>>
<<package-setup>>
<<org-package-setup>>
(require 'ol)
<<org-babel-default-header-args>>
<<org-styles>>
<<org-special-blocks>>
<<org-clean-up-export>>
<<org-sacha-include-link>>
<<org-blog-link>>
<<org-dotemacs-link>>
<<org-yt-link>>
<<org-video-link>>
<<org-audio-link>>
<<org-captions-link>>
<<org-project-link>>
<<org-elisp-link>>
<<org-irc-link>>
<<org-protocol-link>>
<<org-journal-link>>
<<org-package-link>>
<<org-defun-link>>
<<org-defvar-link>>
<<sacha-mastodon-store-link>>
<<org-sketch-link>>
<<org-copy-link>>
<<org-config-link>>

```

I want my config file to be tangled and exported to HTML regularly so that I don't forget to do so. The following code exports my config, but only if I saved it myself instead of when I auto-save it by focusing away from Emacs.

```

(defmacro sacha-org-debounce-idle-timer (seconds var body &rest args)
  `(progn
    (defvar ,var nil "Timer.")
    (when (timerp ,var) (cancel-timer ,var))

```



```

    (setq ,var (run-with-idle-timer ,seconds nil ,body ,@args))))
(defvar sacha-unfocusing nil "Non-nil when I'm in the middle of unfocusing.")
;;;###autoload
(defun sacha-org-async-export-and-tangle (&optional filename)
  (async-start
   `(lambda ()
      ;; make async emacs aware of packages (for byte-compilation)
      (package-initialize)
      (setq package-enable-at-startup nil)
      (require 'org)

      (setq-default tab-width 8)
      (setq org-babel-default-header-args
        '(:session . "none")
          (:results . "drawer replace")
          (:comments . "link") ;; add a link to ↔

      ↪ the original source
          (:exports . "both")
          (:cache . "no")
          (:eval . "never-export") ;; explicitly evaluate blocks instead ↪
      ↪ of evaluating them during export
          (:hlines . "no")
          (:tangle . "no")))) ;; I have to explicitly set up blocks for ↪

      ↪ tangling
      (org-babel-tangle-file ,(buffer-file-name))
      )
    (lambda (&rest results) (message "Tangled.")))
  (org-export-to-file 'html (or filename "index.html") t))
(defun sacha-org-export-and-tangle-if-saved-in-focus ()
  (interactive)
  (when (frame-focus-state)
    (message "Scheduling export...")
    (sacha-org-debounce-idle-timer
     10
     sacha-export-org-config
     (lambda (buf)
       (with-current-buffer buf
         (sacha-org-async-export-and-tangle "index.html"))
       (current-buffer)))))

;;;###autoload
(define-minor-mode sacha-org-export-and-tangle-when-saved-in-focus-mode
  "Toggle a mode for exporting and tangling when saved.
Interactively with no argument, this command toggles the mode.
A positive prefix argument enables the mode, any other prefix
argument disables it. From Lisp, argument omitted or nil enables
the mode, `toggle' toggles the state."
  :group 'my
  (if sacha-org-export-and-tangle-when-saved-in-focus-mode
      (add-hook 'after-save-hook #'sacha-org-export-and-tangle-if-saved-in-focus nil t)
      (remove-hook 'after-save-hook #'sacha-org-export-and-tangle-if-saved-in-focus t)))

;;;###autoload
(defun sacha-org-save-and-tangle-sacha-config ()
  (when (string= (buffer-file-name) (expand-file-name "~/sync/emacs/Sacha.org")) ↪
    ↪ (sacha-org-export-and-tangle-when-saved-in-focus-mode 1)))

;;;###autoload
(defun sacha-emacs-export (&optional sync)
  (interactive (list current-prefix-arg))

```

```

(with-current-buffer (find-file-noselect "~/sync/emacs/Sacha.org")
  (if sync
      (progn
        (org-babel-tangle)
        (org-export-to-file 'html "index.html"))
      (async-start
        `(lambda ()
           (load-file "~/sync/emacs/batch-tangle.el")
           (org-babel-tangle-file "~/sync/emacs/Sacha.org")
           (load-file "~/sync/emacs/Sacha.el")
           (find-file "~/sync/emacs/Sacha.org")
           (org-export-to-file 'html "index.html"))
          (lambda (&rest results) (message "Tangled and exported."))))))
; (use-package org
;   :hook ((org-mode . sacha-org-save-and-tangle-sacha-config)))

```

Let's see if this makes it easier for me to tweak things.

8.27.25 Plain text

For Emacs News, I want to export links without punctuation, and I want to indent lists by 4 spaces instead of 2. This makes my Emacs News export like this:

- Org Mode:
 - Stop renting your life - own it <https://curtismchale.ca/2025/08/23/stop-renting-your-life-own-it/>
 - TIL: Org Mode Can Sort Lists Automatically! https://www.reddit.com/r/emacs/comments/1musbn8/til_o
 - Randy Ridenour: Creating Attendance Sheets with Org Mode <https://randyridenour.net/posts/2025-08->

instead of:

- Org Mode:
 - [Stop renting your life - own it] (<<https://curtismchale.ca/2025/08/23/stop-renting-your-life-own-i>
 - [TIL: Org Mode Can Sort Lists Automatically!] (<https://www.reddit.com/r/emacs/comments/1musbn8/til_o
 - [Randy Ridenour: Creating Attendance Sheets with Org Mode] (<<https://randyridenour.net/posts/2025-0>

```

;;;###autoload
(defun sacha-plain-text-link (link contents info)
  "Export LINK in 'description URL' format."
  (let* ((type (org-element-property :type link))
         (path (org-element-property :path link))
         (raw-link (org-element-property :raw-link link))
         (description (or contents
                           (and (string= type "fuzzy") path)
                           path)))
    (url (cond
          ((member type '("http" "https"))
           (concat type ":" path))
          ((string= type "file") path)
          (t raw-link))))
    (cond
     ((org-export-custom-protocol-maybe link description 'sacha-plain-text info)
      (t
       (if description
            (format "%s %s" description url)
            url))))))
;;;###autoload

```

```

(defun sacha-plain-text-item (item contents info)
  "Transcode an ITEM element with 4-space indentation."
  (replace-regexp-in-string "^\\( \\)+ " "\\1\\1"
                             (org-ascii-item item contents info)))

;;###autoload
(defun sacha-plain-text-export-to-buffer (&optional async subtreep visible-only body-only
                                          ↵ ext-plist)
  "Export current buffer to plain text buffer."
  (interactive)
  (org-export-to-buffer 'sacha-plain-text "*My Plain Text Export*"
                        async subtreep visible-only body-only ext-plist))

;;###autoload
(defun sacha-plain-text-export-to-file (&optional async subtreep visible-only body-only
                                           ↵ ext-plist)
  "Export current buffer to plain text file."
  (interactive)
  (let ((file (org-export-output-file-name ".txt" subtreep)))
    (org-export-to-file 'sacha-plain-text file
                        async subtreep visible-only body-only ext-plist)))

```

```

(with-eval-after-load 'org
  (org-export-define-derived-backend 'sacha-plain-text 'ascii
    :translate-alist '((link . sacha-plain-text-link)
                      (item . sacha-plain-text-item))
    :menu-entry '(?p "Export to custom plain text"
                    ((?p "As plain text buffer" sacha-plain-text-export-to-buffer)
                     (?P "As plain text file" sacha-plain-text-export-to-file))))
  (add-to-list 'org-export-backends 'sacha-plain-text)
  (provide 'ox-sacha-plain-text))

```

8.27.26 PDF

<https://so.nwalsh.com/2020/01/05-latex> , but I use letter paper instead of A4.

```

(setq org-latex-compiler "xelatex")
(setq org-latex-pdf-process
  (list (concat "latexmk -"
                org-latex-compiler
                " -recorder -synctex=1 -bibtex-cond %b"))))
(setq org-latex-default-packages-alist
  '((" " "graphicx" t)
    (" " "grffile" t)
    (" " "longtable" nil)
    (" " "wrapfig" nil)
    (" " "rotating" nil)
    ("normalem" "ulem" t)
    (" " "amsmath" t)
    (" " "textcomp" t)
    (" " "amssymb" t)
    (" " "capt-of" nil)
    (" " "hyperref" nil)))
(setq org-latex-classes
  '(("article"
    "\\RequirePackage{fix-cm}

```

```

\PassOptionsToPackage{svgnames}{xcolor}
\documentclass[11pt]{article}
\usepackage{fontspec}
\setmainfont{Noto Sans}
\setsansfont[Scale=MatchLowercase]{Noto Sans}
\setmonofont[Scale=MatchLowercase]{Hack}
\usepackage{sectsty}
\allsectionsfont{\sffamily}
\usepackage{enumitem}
\setlist[description]{style=unboxed,font=\sffamily\bfseries}
\usepackage{listings}
\lstset{frame=single,aboveskip=1em,
        framesep=.5em,backgroundcolor=\color{AliceBlue},
        rulecolor=\color{LightSteelBlue},framerule=1pt}
\usepackage{xcolor}
\newcommand\basicdefault[1]{\scriptsize\color{Black}\ttfamily#1}
\lstset{basicstyle=\basicdefault{\spaceskip1em}}
\lstset{literate=
    {\$}{\S}1
    {\@}{\raisebox{.125ex}{\copyright}\enspace}1
    {\«}{\guillemotleft}1
    {\»}{\guillemotright}1
    {\AA}{\A}1
    {\AE}{\A}1
    {\E}{\E}1
    {\I}{\I}1
    {\O}{\O}1
    {\OO}{\O}1
    {\U}{\U}1
    {\UU}{\U}1
    {\B}{\ss}2
    {\a}{\a}1
    {\aa}{\a}1
    {\ae}{\a}1
    {\e}{\e}1
    {\i}{\i}1
    {\o}{\o}1
    {\oe}{\o}1
    {\u}{\u}1
    {\ue}{\u}1
    {^}{\textsuperscript1}1
    {^2}{\textsuperscript2}1
    {^3}{\textsuperscript3}1
    \_i{\i}1
    -{\—}1
    '{''}1
    ...{\dots}1
    @{\hookleftarrow}1
    _{\textvisiblespace}1,
    keywordstyle=\color{DarkGreen}\bfseries,
    identifierstyle=\color{DarkRed},
    commentstyle=\color{Gray}\upshape,
    stringstyle=\color{DarkBlue}\upshape,
    emphstyle=\color{Chocolate}\upshape,
    showstringspaces=false,
    columns=fullflexible,
    keepspaces=true}
\usepackage[margin=1in,left=1.5in]{geometry}
\usepackage{parskip}

```

```

\\makeatletter
\\renewcommand{\\maketitle}{%
  \\begingroup\\parindent0pt
  \\sffamily
  \\Huge{\\bfseries\\@title}\\par\\bigskip
  \\LARGE{\\bfseries\\@author}\\par\\medskip
  \\normalsize\\@date\\par\\bigskip
  \\endgroup\\@afterindentfalse\\@afterheading}
\\makeatother
[DEFAULT-PACKAGES]
\\hypersetup{linkcolor=Blue,urlcolor=DarkBlue,
  citecolor=DarkRed,colorlinks=true}
\\AtBeginDocument{\\renewcommand{\\UrlFont}{\\ttfamily}}
[PACKAGES]
[EXTRA]"
("\\section{%s}" . "\\section*{%s}")
("\\subsection{%s}" . "\\subsection*{%s}")
("\\subsubsection{%s}" . "\\subsubsection*{%s}")
("\\paragraph{%s}" . "\\paragraph*{%s}")
("\\subparagraph{%s}" . "\\subparagraph*{%s}"))

("report" "\\documentclass[11pt]{report}"
("\\part{%s}" . "\\part*{%s}")
("\\chapter{%s}" . "\\chapter*{%s}")
("\\section{%s}" . "\\section*{%s}")
("\\subsection{%s}" . "\\subsection*{%s}")
("\\subsubsection{%s}" . "\\subsubsection*{%s}"))

("book" "\\documentclass[11pt]{book}"
("\\part{%s}" . "\\part*{%s}")
("\\chapter{%s}" . "\\chapter*{%s}")
("\\section{%s}" . "\\section*{%s}")
("\\subsection{%s}" . "\\subsection*{%s}")
("\\subsubsection{%s}" . "\\subsubsection*{%s}"))))

```

8.28 Org roam

```

(use-package org-roam
  :if sachalaptop-p
  :ensure t

  :config
  (org-roam-db-autosync-mode)

  :hook
  (after-init . org-roam-mode)

  :custom
  (org-roam-directory "/home/sacha/sync/topics")

  :bind (:map org-roam-mode-map
    (("C-c n l" . org-roam)
     ("C-c n f" . org-roam-find-file)
     ("C-c n g" . org-roam-graph))
    :map org-roam-mode-map
    (("C-c n i" . org-roam-insert)
     ("C-c n I" . org-roam-insert-immediate))))

```

8.29 Links

8.29.1 Convert an inline link into a side note/footnote

```
;;###autoload
(defun sacha-org-convert-link-to-footnote (label)
  "Convert the link at point into a footnote."
  (interactive (list (read-string "Footnote label: ")))
  ;; Get the Org link path at point
  (when (org-in-regexp org-link-bracket-re 1)
    (let ((url (match-string 1)))
      (replace-match (match-string 2) 0)
      (skip-syntax-forward ".")
      (sacha-org-footnote-add (org-link-make-string (org-link-unescape url)
                                                    (sacha-org-link-default-description url)
                                                    ↵ nil))))))

;;###autoload
(defun sacha-org-footnote-add (label &optional text)
  (interactive "MLabel: \n")
  (insert (format "[fn:%s]" label))
  (goto-char (org-footnote-create-definition label))
  (goto-char (line-end-position))
  (when text
    (insert text)))
```

8.29.2 Adding Org Mode link awesomeness elsewhere: sacha-org-insert-link-dwim

org

[2025-09-23 Tue]: Changed my mind, I want the clipboard URL to be used by default. More bugfixes. [2025-09-22 Mon]: Fix bug in `sacha-page-title`. Add `mastodon-toot-mode-map`.

I love so many things about [Org Mode](#)'s links. I can use `C-c C-l` (`org-insert-link`) to insert a link. If I've selected some text, `C-c C-l` turns the text into the link's description. I can define my own custom link types with interactive completion, default descriptions, and export formats. This is so nice, I want it in all the different places I write links in:

- Markdown, like on the [EmacsConf wiki](#); then I don't have to remember Markdown's syntax for links
- [mastodon.el](#) toots
- Oddmuse, like on [EmacsWiki](#)
- HTML/Web mode
- Org Mode HTML export blocks

Some considerations inspired by [Emacs DWIM: do what I mean](#), which I used as a starting point:

- I want Emacs to use the URL from the clipboard.
- If I haven't already selected some text, I want to use the page title or the custom link description as a default description.
- I want to be able to use my custom link types for completion, but I want it to insert the external web links if I'm putting the link into a non-Org Mode buffer (or in a source or export block that isn't Org Mode). For example, let's say I select `dotemacs:sacha-org-insert-link-dwim` with completion. In Org Mode, it should use

that as the link target so that I can follow the link to my config and have it exported as an HTML link. In Markdown, it should be inserted as `[Adding Org Mode niceties elsewhere: sachalog-insert-link-dwim](https://sachalog.com/dotemacs#sachalog-insert-link-dwim)`.

Mostly, this is motivated by my annoyance with having to work with different link syntaxes:

HTML	<code>title</code>
Org	<code>[[https://example.com][title]]</code>
Plain text	<code>title https://example.com</code>
Markdown	<code>[https://example.com](title)</code>
Oddmuse	<code>[https://example.com title]</code>

I want things to Just Work.

[https://sachalog.com/blog/2025/09/adding-org-mode-link-awesomeness-elsewhere-sachalog-insert-link-dwim/sachalog-insert-link-dwim.webm?caption=ScreenCast showing how I insert links](https://sachalog.com/blog/2025/09/adding-org-mode-link-awesomeness-elsewhere-sachalog-insert-link-dwim/sachalog-insert-link-dwim.webm?caption=ScreenCast%20showing%20how%20I%20insert%20links)

Play by play:

1. `vtime:0:00:00` inserting a custom dotemacs link with completion
2. `vtime:0:00:11` inserting a link to a blog post
3. `vtime:0:00:28` selecting text in an HTML export block and adding a link to it
4. `vtime:0:00:48` adding a bookmark link as a plain text link in a Python src block

Here's the `sachalog-insert-link-dwim` function, using `sachalog-link-as-url` from [Copy web link](#) (p. 334) and `sachalog-set-link-target-with-search` from [Using web searches and bookmarks to quickly link placeholders in Org Mode](#) (p. 54):

```
;; ##autoload
(defun sachalog-insert-link-dwim (&optional url title)
  "Like `org-insert-link' but with personal dwim preferences."
  (interactive)
  (let* ((point-in-link (and (derived-mode-p 'org-mode) (org-in-regexp org-link-any-re 1)))
         (point-in-html-block (and (derived-mode-p 'org-mode)
                                     (let ((elem (org-element-context)))
                                       (and (eq (org-element-type elem) 'export-block)
                                             (string= (org-element-property :type elem) "HTML")))))
         (point-in-src-or-export-block (and (derived-mode-p 'org-mode)
                                              (let ((elem (org-element-context)))
                                                (and (member (org-element-type elem) '(src-block export-block))
                                                      (not (string= (org-element-property :type elem) "Org"))
                                                           (≥ (point) (org-element-property :post-affiliated elem))
                                                           (≤ (point) (+ (length (org-element-property :value elem))
                                                                (org-element-property :post-affiliated elem))))))
                                              (region-content (when (region-active-p)
                                                                (buffer-substring-no-properties (region-beginning)
                                                                (region-end)))))))
    (org-insert-link point-in-link point-in-html-block point-in-src-or-export-block)))
```

```

    (region-end))))
    (bookmark-match (when region-content (sacha-org-bookmark-match region-content)))
    (url (cond
          (url url)
          ((sacha-org-in-bracketed-text-link-p) ←
           nil)
          (bookmark-match bookmark-match)
          ((not point-in-link)
           (sacha-org-read-link
            ;; clipboard
            (when (string-match-p "^http" ←
              (current-kill 0))
              (current-kill 0))))))
          (title (or title
                     region-content
                     (when (or ←
                           (string-match (regexp-quote "*new toot*") (buffer-name))
                           (derived-mode-p '(markdown-mode web-mode oddmuse-mode))
                           point-in-html-block
                           point-in-src-or-export-block
                           (not (and (derived-mode-p 'org-mode)
                                      point-in-link)))
                           (read-string "Title: "
                                       (or ←
                                           (sacha-org-link-default-description url nil)
                                           (sacha-page-title ←
                                            url))))))
                     ;; resolve the links; see sacha-org-link-as-url in ←
                     https://sachachua.com/dotemacs#web-link
                     (unless (and (derived-mode-p 'org-mode)
                                   (not (or point-in-html-block ←
                                           point-in-src-or-export-block)))
                       (setq url (sacha-org-link-as-url url)))
                     (when (region-active-p) (delete-region (region-beginning) (region-end)))
                     (cond
                      ((or (string-match (regexp-quote "*new toot*") (buffer-name))
                          (derived-mode-p 'markdown-mode))
                       (insert (format "[%s](%s)" title url)))
                      ((or (derived-mode-p '(web-mode html-mode)) point-in-html-block)
                       (insert (format "<a href=\"%s\">%s</a>" url title)))
                      ((derived-mode-p 'oddmuse-mode)
                       (insert (format "[%s %s]" url title)))
                      ((or point-in-src-or-export-block
                          (not (derived-mode-p 'org-mode)))
                       (insert title " " url)))
                      ((or (derived-mode-p 'message-mode)
                          (derived-mode-p 'notmuch-message-mode))
                       (insert title " " url)))
                      ((and region-content url (not point-in-link))
                       (insert (org-link-make-string url region-content)))
                      ((and url (not point-in-link))

```



```

(insert (org-link-make-string
      url
      (or title
          (read-string "Title: "
                        ↵
                        (or (sacha-org-link-default-description url nil)
                            ↵
                            (sacha-page-title url)))))))
;; bracketed [[plain text]]; see Using web searches and bookmarks to quickly ↵
↵ link placeholders in Org Mode ↵
↵ https://sachachua.com/dotemacs#completion-consult-consult-omni-using-web-searches-and-bookmarks-to-quickly-
((sacha-org-set-link-target-with-search))
;; In Org Mode, edit the link
((call-interactively 'org-insert-link))))

```

Consistent keybindings mean less thinking.

```

(dolist (group '((org . org-mode-map)
                  (markdown-mode . ↵
                  (markdown-mode-map)
                  (mastodon-toot . ↵
                  (mastodon-toot-mode-map)
                  (web-mode . web-mode-map)
                  (oddmuse-mode . ↵
                  (oddmuse-mode-map)
                  (notmuch . notmuch-message-mode-map)
                  (text-mode . text-mode-map)
                  (html-mode . html-mode-map)))
  (with-eval-after-load (car group)
    (keymap-set (symbol-value (cdr group)) "C-c C-l" #'sacha-org-insert-link-dwim)))

```

All right, let's dig into the details. This code gets the page title so that we can use it as the link's description. I like to simplify some page titles. For example, when I link to Reddit or HN discussions, I just want to use "Reddit" or "HN".

```

;;;###autoload
(defun sacha-page-title (url)
  "Get the page title for URL. Simplify some titles."
  (condition-case nil
    (pcase url
      ((rx "reddit.com") "Reddit")
      ((rx "news.ycombinator.com") "HN")
      ((rx "lobste.rs") "lobste.rs")
      (_
       (with-current-buffer (url-retrieve-synchronously url)
         (string-trim
          (replace-regexp-in-string
            "[ \n]+" " "
            (replace-regexp-in-string
              "\\(^Github - \\|:: Sacha Chua\\)" ""
              (or
                (dom-texts (car
                            ↵
                            (dom-by-tag (libxml-parse-html-region
                            ↵
                            (point-min)
                            ↵
                            (point-max))

```

```

↪                                     'title)))
                                     ""))))))
(error nil)))

```

Let's use that as the default for `https:` links too.

```

(defun sacha-org-link-https-insert-description (link desc)
  "Default to the page title."
  (unless desc (sacha-page-title link)))

```

```

(with-eval-after-load 'org
  (org-link-set-parameters "https" :insert-description ↪
    ↪ #'sacha-org-link-https-insert-description))

```

I want to get the default description for a link, even if it uses a custom link type. I extracted this code from `org-insert-link`.

```

;;;###autoload
(defun sacha-org-link-default-description (link desc)
  "Return the default description for an Org Mode LINK.
This uses :insert-description if defined."
  (let* ((abbrevs org-link-abbrev-alist-local)
         (all-prefixes (append (mapcar #'car abbrevs)
                                ↪
                                (mapcar #'car org-link-abbrev-alist)
                                ↪
                                (org-link-types))))
    (type
     (cond
      ((and all-prefixes
            (string-match (rx-to-string `(: string-start (submatch (or ,@all-prefixes)) ↪
            ↪ ":")) link))
       (match-string 1 link))
      ((file-name-absolute-p link) "file")
      ((string-match "\\`\\.\\.\\.\\.\\.\\.\\.\\.?/" link) "file"))))
    (when (org-link-get-parameter type :insert-description)
      (let ((def (org-link-get-parameter type :insert-description)))
        (condition-case nil
          (cond
           ((stringp def) def)
           ((functionp def)
            (funcall def link desc)))
          (error
           nil))))))

```

Now I want an Emacs Lisp function that interactively reads a link with completion, but doesn't actually insert it. I extracted this logic from `org-read-link`.

```

;;;###autoload
(defun sacha-org-read-link (&optional default)
  "Act like `org-insert-link'. Return link."
  (let* ((wcf (current-window-configuration))
         (origbuf (current-buffer))
         (abbrevs org-link-abbrev-alist-local)
         (all-prefixes (append (mapcar #'car abbrevs)

```


getting it from the clipboard), get the default description from the link (using custom links' `:insert-description` or the webpage's title), and either wrap the link around the region or insert it in whatever syntax makes sense.

On a related note, you might also enjoy:

- [Bookmarks](#) (p. 246) - my custom link type for bookmark: that offers completion from resources.org
- [Linking to blog posts](#) (p. 271) - my custom link type for blog posts
- [Using an Emacs Lisp macro to define quick custom Org Mode links to project files; plus URLs and search](#) (p. 332)

And elsewhere:

- [Generating Descriptions of org-insert-link](#) // [Take on Rules](#)
- [Adding More Link Description Defaults](#) // [Take on Rules](#)

8.29.3 IDs

```
(setq org-id-method 'ts)
(setq org-id-link-to-org-use-id 'create-if-interactive-and-no-custom-id)
```

8.29.4 Quick links

```
(setq org-link-abbrev-alist
  '(("google" . "http://www.google.com/search?q=")
    ("gmap" . "http://maps.google.com/maps?q=%s")
  ))
```

8.29.5 Tag links

From <http://endlessparentheses.com/use-org-mode-links-for-absolutely-anything.html?source=rss>

```
(org-add-link-type "tag" 'endless/follow-tag-link)
```

```
(defun endless/follow-tag-link (tag) "Display a list of TODO headlines with tag TAG.
With prefix argument, also display headlines without a TODO keyword." (org-tags-view
  (null current-prefix-arg) tag))
```

8.29.6 Links to my config

```
;;###autoload
(defvar sachas-emacs-config-url)
(defun sachas-org-dotemacs-export (path desc format _)
  "Export dotemacs link."
  (pcase format
    ((or 'html '11ty 'md)
     (format "<a href=\"%s#%s\">%s</a>"
              sachas-emacs-config-url
              path (or desc path)))
    ('latex
     (format "\\hyperref[%s]{%s} (p.~\\pageref{%s})"
              path (or desc path) path)))
```

```

      (ascii
        (if desc
          (format "%s %s%s"
                  desc
                  sach emacs-config-url
                  path)
          (format "%s%s"
                  sach emacs-config-url
                  path))))))

;;;###autoload
(defun sach org-dotemacs-complete ()
  "Prompt for dotemacs."
  (interactive)
  (with-current-buffer (find-file-noselect "~/sync/emacs/Sacha.org")
    (concat "dotemacs:" (org-read-property-value "CUSTOM_ID"))))

;;;###autoload
(defun sach org-dotemacs-insert-description (link &optional description)
  (unless description
    (with-current-buffer (find-file-noselect "~/sync/emacs/Sacha.org")
      (save-restriction
        (save-excursion
          (widen)
          (goto-char (org-find-property "CUSTOM_ID" ↵
            ↵ (replace-regexp-in-string "^dotemacs:" "" link)))
            (org-entry-get (point) "ITEM"))))))))

;;;###autoload
(defun sach org-dotemacs-open (path)
  (with-current-buffer (find-file-noselect "~/sync/emacs/Sacha.org")
    (when-let ((pos (org-find-property "CUSTOM_ID" (replace-regexp-in-string ↵
      ↵ "^dotemacs:" "" path))))
      (switch-to-buffer (current-buffer))
      (goto-char pos))))))

;;;###autoload
(defun sach org-dotemacs-store ()
  (when (and (string= (buffer-file-name)
                      (expand-file-name "~/sync/emacs/Sacha.org"))
            (org-entry-get (point) "CUSTOM_ID"))
    (org-link-store-props
     :link (concat "dotemacs:" (org-entry-get (point) "CUSTOM_ID"))
     :description (org-entry-get (point) "ITEM"))))

```

```

(with-eval-after-load 'org
  (org-link-set-parameters
   "dotemacs"
   :complete #'sach org-dotemacs-complete
   :store #'sach org-dotemacs-store
   :insert-description #'sach org-dotemacs-insert-description
   :export #'sach org-dotemacs-export
   :follow #'sach org-dotemacs-open))

```

TODO add dotemacs completion

1. Config

```
(use-package org
:config
(org-link-set-parameters
"config"
:follow (lambda (id) (org-open-link-from-string (format "%s"
↳ "[~/sync/emacs/Sacha.org::%s]" id)))
:export (lambda (link description format)
(format "<a href=\"%s\">%s</a>" link
↳ description))))
```

8.29.7 YouTube

```
(defvar sach-org-yt-iframe-format
(concat "<div class=\"yt-video\"><iframe width=\"456\"\"
height=\"315\"\"
title=\"YouTube video player\"\"
src=\"https://www.youtube-nocookie.com/embed/%s?enablejsapi=1\"\"
frameborder=\"0\"\"
allowfullscreen>%s</iframe><a href=\"%s\">Watch on YouTube</a></div>"))

;;###autoload
(defun sach-org-yt-id (path)
(cond
((string-match "\\(?:v=\\|tu\\.be\\/\\|live\\/\\)\\([^\&]+\\)" path)
(match-string 1 path))
((string-match "\\(live_stream\\|?channel\\.\\)" path)
(match-string 1 path))
(t path)))

;;###autoload
(defun sach-org-yt-export (path desc format _)
"Export time link or embed."
(pcase format
((or 'html '11ty 'md)
(cond
(desc (format "<a href=\"%s\">%s</a>" path (or desc path)))
(t
(let* ((path-and-query (url-path-and-query (url-generic-parse-url
↳ path)))
(url (car path-and-query))
(params (and (cdr path-and-query)
↳ (url-parse-query-string (cdr path-and-query))))
(id (cond
↳ ((string-match
↳ "\\(?:v=\\|tu\\.be\\/\\|live\\/\\)\\([^\&]+\\)" path)
↳ (match-string
↳ 1 path))
↳ ((string-match
↳ "\\(live_stream\\|?channel\\.\\)" path)
↳ (match-string
↳ 1 path))
↳ (t path)))
(width (or (car (assoc-default "width"
↳ params 'string=)) "456"))
```



```

        (let ((split-time (cdr (reverse (split-string (match-string 1 link) ←
↳ "[hms]")))))
            (concat
              (if (and (elt split-time 2) (not (string= (elt split-time 2) ←
↳ "0")))
                  (concat (elt split-time 2) ":")
                  "")
              (if (elt split-time 1)
                  (concat (if (and (and (elt split-time 2) (not ←
↳ (string= (elt split-time 2) "0")))
                              (elt ←
↳ (< (length (elt split-time 1)) 2))
                              (elt ←
↳ "0" ""))
                  (elt ←
↳ split-time 1) ":")
                  ""))
            (concat (if (and (elt split-time 1) (< (length (elt ←
↳ split-time 0)) 2)) "0" ""
                    (elt split-time 0))))))
(ert-deftest sach-org-yt-insert-description ()
  (should
    (string=
      (sach-org-yt-insert-description "yt:somevideo?t=0h1m2s")
      "1:02"))
  (should
    (string=
      (sach-org-yt-insert-description "yt:somevideo?t=1h2m3s")
      "1:02:03")))
;;###autoload
(defun sach-org-yt-open (path)
  (browse-url path))

```

```

;;###autoload
(defun sach-org-copy-region-as-plain-text (beg end)
  "Copy as plain text."
  (interactive "r")
  (save-restriction
    (narrow-to-region beg end)
    (kill-new (org-export-as 'sach-plain-text nil nil t))))

```

```

(org-link-set-parameters "yt" :complete #'sach-org-yt-complete
↳
↳ :insert-description #'sach-org-yt-insert-description
↳
↳ :export #'sach-org-yt-export
↳
↳ :follow #'sach-org-yt-open)

```

8.29.8 Videos

```

(org-link-set-parameters
  "video"
  :export #'sach-org-video-export
  :follow #'sach-org-video-follow)

```



```
:complete #'sacha-org-video-complete)
```

```
;;;###autoload
(defun sacha-org-video-follow (path _)
  (cond
    ((string-match "\\(https://.+\\):\\([0-9:]+\\)" path)
     (mpv-start (concat (match-string 1 path) "?t=" (sacha-org-yt-convert-time ←
→ (match-string 2 path))))))
    ((string-match "https:" path)
     (mpv-start path))
    ((string-match "\\(.+?\\):\\([0-9:]+\\)" path)
     (mpv-start (expand-file-name (match-string 1 path)
                                   (concat "--start=" (match-string 2 ←
→ path))))))
    (t (mpv-play (expand-file-name (replace-regexp-in-string "\\?.*" "" path))))))

;;;###autoload
(defun sacha-org-video-replace-with-permalink ()
  (interactive)
  (let* ((elem (org-element-context))
         (path (org-element-property :path elem))
         (description (org-element-property :description elem))
         (permalink (org-entry-get (point) "EXPORT_ELEVENTY_PERMALINK" ←
→ t)))
    (delete-region (org-element-begin elem) (org-element-end elem))
    (insert (org-link-make-string (concat "video:https://sachachua.com" permalink ←
→ (file-name-nondirectory path))
                                   description))))

;;;###autoload
(defun sacha-org-video-export (link desc format info)
  "Export PATH to FORMAT using the specified wrap parameter."
  (if desc
      (org-export-string-as (org-link-make-string link desc) format)
      (pcase format
        ((or 'html '11ty 'md)
         (let* ((parsed-url (url-generic-parse-url link))
                (path-and-query (url-path-and-query parsed-url))
                (url
                 (if (string-match "^https://" link)
                     (concat (url-type parsed-url) "://" (url-domain parsed-url) (car ←
→ path-and-query))
                     (concat "file://" (if (file-name-absolute-p (car path-and-query))
                                           (expand-file-name (car ←
→ path-and-query))
                                           (car path-and-query))))))
          (params (and (cdr path-and-query) ←
→ (url-parse-query-string (cdr path-and-query))))
          (body
           (format
            "<video%s%>src=\"%s\" ←
→ %stype=\"%s\">%s%</video>%s"
            (if (string= (or (car ←
→ (assoc-default "controls" params 'string= ('("1")) "1") "0"))
```

```

""
" controls=\"1\""
(if (string= (or (car ←
↪ (assoc-default "autoplay" params 'string= '("0"))) "0") "0")
""
" autoplay=\"1\""
url
(if (assoc-default "thumbnail" ←
↪ params)
(format ←
↪ "poster=\"%s\" "
↪
↪ (car (assoc-default "thumbnail" params)))
""
↪
↪ (mailcap-file-name-to-mime-type (car path-and-query))
(if (assoc-default "captions" ←
↪ params)
(format "<track kind=\"subtitles\" label=\"Captions\" ←
↪ src=\"%s\" srclang=\"en\" default></track>"
(cond
((string= (car (assoc-default "captions" params)) "t")
(concat (file-name-sans-extension url) ".vtt"))
((string-match "^https://" (car (assoc-default "captions" params)))
(car (assoc-default "captions" params)))
(t
↪
↪ (expand-file-name (car (assoc-default "captions" params))))
""
(if (assoc-default "thumbnail" ←
↪ params)
(format ←
↪ "<span>Video not supported. Thumbnail:<br /><img src=\"%s\" alt=\"Thumbnail\" /></span>"
↪
↪ (car (assoc-default "thumbnail" params)))
""
(if (string-match "^https://" url)
""
(format " <a href=\"%s\">Download the video</a>"
(car path-and-query)))
(if (assoc-default ←
↪ "captions-below" params)
"<div ←
↪ class=\"captions\" style=\"display: none\"></div>"
""
))
(when (assoc-default "caption" params)
(setq body (format ←
↪ "<figure>%s<figcaption><div>%s</div></figcaption></figure>"
↪
↪ body
↪
↪ (car (assoc-default "caption" params))))
body))
(_ link)))

;;;###autoload
(defun sach-org-video-complete ()
  "Complete video reference."

```

```
(interactive)
(concat "video:" (read-file-name "File: "))
```

8.29.9 Linking to a specific time in a video

```
(org-link-set-parameters
 "vtime"
 :export #'sacha-org-video-time-export
 :complete #'sacha-org-video-time-complete
 :follow #'sacha-org-video-time-follow)
```

```
;;;###autoload
(defun sacha-org-video-time-follow (path _)
  ;; TODO: Look for the previous video and jump to the specified time
  (mpv-seek path)
  (mpv--enqueue '("set" "pause" "no") #'ignore))

;;;###autoload
(defun sacha-org-video-time-complete ()
  (interactive)
  (concat "vtime:" (format-seconds "%02h:%02m:%02s" (mpv-get-playback-position))))

;;;###autoload
(defun sacha-org-video-time-export (link desc format info)
  "Export PATH to FORMAT using the specified wrap parameter."
  (pcase format
    ((or 'html '11ty 'md)
     (when (string-match "\\([0-9]+:\\)?[0-9]+:[0-9]+" link)
       (format "<span class=\"media-time\" data-start=\"%03f\">%s</span>"
                (save-match-data
                  (/ ←
                    (← (compile-media-timestamp-to-msecs
                       (match-string 0 link)) 1000.0))
                      (match-string 0 link))))
       ('sacha-plain-text
        (when (string-match "\\([0-9]+:\\)?[0-9]+:[0-9]+" link)
          (match-string 0 link)))
       ('org (org-link-make-string (concat "vtime:" link) desc))))
    _
    (match-string 0 link)))
```

```
(defun sacha-org-vtime-item-p ()
  (save-excursion
    (org-list-at-regexp-after-bullet-p
     "\\[\\[\\[\\[\\(vtime:\\([0-9]+:\\)?[0-9]+:[0-9]+\\)\\]\\]\\]\\]")))

;;;###autoload
(defun sacha-org-vtime-insert-item-advice (fn &rest args)
  (let ((itemp (org-in-item-p))
        (pos (point)))
    (unless (or (not itemp)
                 (save-excursion
                   (goto-char itemp)
                   (org-invisible-p)))
      (if (sacha-org-vtime-item-p)
          ;; Insert another vtime link
          (sacha-org-vtime-item)
```

```

      (apply fn args))))))

(defun sach-org-vtime-item ()
  (let ((itemp (org-in-item-p)) (pos (point)))
    (cond
      ;; In a timer list, insert with `org-list-insert-item',
      ;; then fix the list.
      ((and itemp (goto-char itemp) (sach-org-vtime-item-p))
       (let* ((struct (org-list-struct))
              (prevs (org-list-prevs-alist struct))
              (s (concat (org-link-make-string (sach-org-video-time-complete)) " - ")))
         (setq struct (org-list-insert-item pos struct prevs nil s))
         (org-list-write-struct struct (org-list-parents-alist struct))
         (looking-at org-list-full-item-re)
         (goto-char (match-end 0))
         (line-end-position)))
      ;; In a list of another type, don't break anything: throw an error.
      (itemp (goto-char pos) (error "This is not a vtime list")))
      ;; Else, start a new list.
      (t
       (forward-line 0)
       (org-indent-line)
       (insert "- "
              (concat (org-link-make-string (sach-org-video-time-complete)) " - "))))))

```

```

(with-eval-after-load 'org
  (advice-add 'org-insert-item :around 'sach-org-vtime-insert-item-advice))

```

8.29.10 Audio

```

(with-eval-after-load 'org
  (org-link-set-parameters
   "audio"
   :export #'sach-org-audio-export
   :follow #'sach-org-video-follow
   :complete #'sach-org-audio-complete)

  (org-link-set-parameters
   "audioi"
   :export #'sach-org-audio-export
   :follow #'sach-org-video-follow
   :complete #'sach-org-audio-icon-complete))

```

```

;;;###autoload
(defun sach-org-audio-replace-with-permalink ()
  (interactive)
  (let* ((elem (org-element-context))
         (path (org-element-property :path elem))
         (description (org-element-property :description elem))
         (permalink (org-entry-get (point) "EXPORT_ELEVENTY_PERMALINK" ↵
         ↵ t)))
    (delete-region (org-element-begin elem) (org-element-end elem))
    (insert (org-link-make-string (concat "audio:https://sachachua.com" permalink ↵
    ↵ (file-name-nondirectory path))

```

```

↪                                     description))))
;;###autoload
(defun sach-audio-export (link desc format &optional info)
  "Export PATH to FORMAT using the specified wrap parameter."
  (pcase format
    ((or 'html 'lily 'md 'sach-html-served)
     (let* ((parsed-url (url-generic-parse-url link))
            (path-and-query (url-path-and-query parsed-url))
            (params (and (cdr path-and-query) ↪
                          (url-parse-query-string (cdr path-and-query))))
     ↪ (element (or (assoc-default "element" params ↪
                                (url (if (string-match "^https://" link)
                                      (concat (url-type parsed-url) "://" (url-domain parsed-url) (car ↪
                                      path-and-query))
                                (concat "file://"
                                      (if (file-name-absolute-p (car path-and-query))
                                          (expand-file-name (car path-and-query))
                                          (car path-and-query))))))
      (is-icon (or (assoc-default "icon" params 'string=)
                   (string= desc "▶"))))
      (if is-icon
          (format
            " <a href=\"%s\" class=\"audio-icon\"%s>%s</a>"
            (concat (car path-and-query)
              (if (string= (or (assoc-default "nocache" params 'string= "0") "1") "1")
                  (concat "?" (format-time-string ↪
                  "%Y-%m-%d"))
                  ""))
            (if (not (string= "" (or (car (assoc-default "title" params 'string=)) ""))
                (format " title=\"%s\" (htmlize-attr-escape (decode-coding-string (car ↪
                (assoc-default "title" params 'string=)) 'utf-8)))
            ""
            (if (not (string= "" (or (car (assoc-default "style" params 'string=)) ""))
                (format " style=\"%s\" (htmlize-attr-escape (car (assoc-default "style" ↪
                ↪ params 'string=))))
            ""
            desc)
            (format
              "<div class=\"audio\"%s>%s%s%s%s preload=\"metadata\" ↪
              ↪ src=\"%s\" type=\"%s\"><a href=\"%s\">Download the audio</a>%s</div>"
              (if desc (concat desc " ") "")
              element
              (if (string= (or (assoc-default "controls" params 'string= ↪
              ↪ "1") "1") "0")
                  ""
                  " controls=\"1\"")
              (if (string= (or (assoc-default "autoplay" params 'string= ↪
              ↪ "0") "0") "0")
                  ""
                  " autoplay=\"1\"")
              (if (assoc-default "id" params)
                  (format " id=\"%s\" (car (assoc-default ↪
              ↪ "id" params))))
              ""))
    ))

```

```

(car path-and-query)
(if (string= (or (assoc-default "nocache" params 'string= ↵
↵ "0") "1") "1")
      (concat "?" (format-time-string "%Y-%m-%d"))
      "")
      (mailcap-file-name-to-mime-type (car path-and-query))
      (car path-and-query)
      (if (assoc-default "captions" params)
          (format "<track kind=\"captions\" ↵
↵ label=\"Captions\" src=\"%s\" srclang=\"en\" default></track>"
          (if (string= ↵
↵ (car (assoc-default "captions" params)) "t")
              (concat (file-name-sans-extension url) ".vtt")
              ""))
          (expand-file-name (car (assoc-default "captions" params))))))
      element
      (if (assoc-default "captions-below" params)
          "<div class=\"captions\" style=\"display: none\"></div>"
          ""))))))
('org
 (org-link-make-string (concat "audio:" link) desc))
 ('latex
  (let* ((parsed-url (url-generic-parse-url link))
         (path-and-query (url-path-and-query parsed-url))
         (params (and (cdr path-and-query) ↵
↵ (url-parse-query-string (cdr path-and-query))))))
    (org-latex-link (car path-and-query) desc info))
    (_ (or desc link))))))

(ert-deftest sach-a-org-audio-export ()
  (should
   (string-match
    "<audio controls=\"1\" id=\"play-this\" src=\"test.opus\" type=\"audio/ogg\"><a ↵
↵ href=\"test.opus\">Download the audio</a><track kind=\"subtitles\" label=\"Captions\" ↵
↵ src=\"test.vtt\" srclang=\"en\" default></track></audio>"
    (sach-a-org-audio-export
     "test.opus?id=play-this&captions=test.vtt"
     nil
     'html
     nil
     )
    )))

;;;###autoload
(defun sach-a-org-audio-complete ()
  "Complete audio reference."
  (interactive)
  (concat "audio:" (read-file-name "File: ") "?captions=t&captions-below=t"))

;;;###autoload
(defun sach-a-org-audio-icon-complete ()
  "Complete audio reference."
  (interactive)
  (concat "audio:" (read-file-name "File: ") "?icon=t"))

```

8.29.11 Captions

```
(org-link-set-parameters
 "captions"
 :export #'sacha-org-captions-export
 :follow #'find-file
 :complete #'sacha-org-captions-complete)
```

```
;; ##autoload
(defun sacha-org-captions-format (file &optional separator)
  (let ((cues (subed-parse-file file))
        last-speaker)
    (mapconcat
     (lambda (cue)
       (let ((text (elt cue 3))
             break)
         (when (elt cue 4)
           (setq last-speaker nil))
         (when (and (string-match "^\\[\\(\\.+?\\)\\]: " text))
           (if (string= last-speaker (match-string 1 text))
               (setq text (replace-match "" nil nil t))
               (setq last-speaker (match-string 1 text))
               (setq break t)))
         (concat
          (if (and (elt cue 4) (not (string= (elt cue 4) "")))
              (format "<p></p><div "
                    ↪ class="transcript-heading"><span class="audio-time" data-start="%f">%s</span> ↪
                    ↪ <strong>%s</strong></div>"
                    ↪ (floor ↪
                    ↪ ↪
                    ↪ (format-seconds "%02h:%02m:%02s" (floor (/ (elt cue 1) 1000)))
                    ↪ (elt ↪
                    ↪ cue 4))
              "")
          (if break "<p></p>" "")
          (format "<span class='audio-time caption'"
                    ↪ ↪ data-speaker="%s" data-start="%f" data-stop="%f" >%s</span>"
                    ↪ (or last-speaker "")
                    ↪ (/ (elt cue 1) 1000.0)
                    ↪ (/ (elt cue 2) 1000.0)
                    ↪ text))))
     cues
     (or separator " "))))

;; ##autoload
(defun sacha-org-captions-export (link desc format _)
  "Export PATH to FORMAT using the specified wrap parameter."
  (if desc
      (org-export-string-as (org-link-make-string link desc) format)
      (pcase format
        ((or 'html '11ty 'md) (sacha-org-captions-format (car ↪
                    ↪ (url-path-and-query (url-generic-parse-url link))))
         (_ path))))

;; ##autoload
(defun sacha-org-captions-complete ()
  "Complete audio reference."
```

```

(interactive)
(concat "captions:" (read-file-name "Captions: ")))

;;;###autoload
(defun sacha-org-captions-insert-as-html-block (file)
  (interactive "FFile: ")
  (insert "#+begin_export html\n" (sacha-org-captions-format file "\n") "\n#+end_export ↵
↵ html\n"))

```

8.29.12 Using an Emacs Lisp macro to define quick custom Org Mode links to project files; plus URLs and search org:emacs:coding:embark

- [2025-03-27 Thu]: Use sacha-org-project- as the prefix to avoid collisions.
- [2024-09-19 Thu]: Added function for replacing current link, bound to C-. r (sacha-embark-replace-link-with-exported-url)
- [2024-01-12 Fri] Added embark action to copy the exported link URL.
- [2024-01-11 Thu] Switched to using Github links since Codeberg's down.
- [2024-01-11 Thu] Updated sacha-copy-link to just return the link if called from Emacs Lisp. Fix getting the properties.
- [2024-01-08 Mon] Add tip from Omar about embark-around-action-hooks
- [2024-01-08 Mon] Simplify code by using consult--grep-position

Summary (882 words): Emacs macros make it easy to define sets of related functions for custom Org links. This makes it easier to link to projects and export or copy the links to the files in the web-based repos. You can also use that information to consult-ripgrep across lots of projects.

I'd like to get better at writing notes while coding and at turning those notes into blog posts and videos. I want to be able to link to files in projects easily with the ability to complete, follow, and export links. For example, `[[subed:subed.el]]` should become `subed.el`, which opens the file if I'm in Emacs and exports a link if I'm publishing a post. I've been making custom link types using `org-link-set-parameters`. I think it's time to make a macro that defines that set of functions for me. Emacs Lisp macros are a great way to write code to write code.

```

(defvar sacha-project-web-base-list nil "Local path . web repo URLs for easy linking.")

;;;###autoload
(defmacro sacha-org-project-link (type file-path git-url)
  `(progn
     (defun ,(intern (format "sacha-org-project-%s-complete" type)) ()
       ,(format "Complete a file from %s." type)
       (concat ,type ":" (completing-read "File: "
                                           ↵
                                           ↵
                                           (projectile-project-files ,file-path))))
     (defun ,(intern (format "sacha-org-project-%s-follow" type)) (link _)
       ,(format "Open a file from %s." type)
       (find-file
        (expand-file-name
         link
         ,file-path)))
     (defun ,(intern (format "sacha-org-project-%s-export" type)) (link desc ↵
       ↵ format _)
       "Export link to file."
       (setq desc (or desc link))
       (when (and ,git-url link)

```



```

                                (setq link (concat ,git-url (replace-regexp-in-string "^/" "" ↵
↵ link))))
                                (pcase format
                                  ((or 'html '11ty) (format "<a href=\"%s\">%s</a>" ↵
↵ link
↵
                                (or desc link)))
                                  ('md (if desc (format "[%s](%s)" desc link)
                                              (format "<%s>" link)))
                                  ('latex (format "\\href{%s}{%s}" link desc))
                                  ('texinfo (format "@uref{%s,%s}" link desc))
                                  ('ascii (format "%s (%s)" desc link))
                                  (_ (format "%s (%s)" desc link)))
                                (org-link-set-parameters
                                  ,type
                                  :complete (quote ,(intern (format "sacha-org-project-%s-complete" ↵
↵ "sacha-org-project-%s-complete" type)))
                                  :export (quote ,(intern (format "sacha-org-project-%s-export" ↵
↵ type)))
                                  :follow (quote ,(intern (format "sacha-org-project-%s-follow" ↵
↵ type))))
                                (cl-pushnew (cons (expand-file-name ,file-path) ,git-url)
                                              sacha-project-web-base-list
                                              :test 'equal)))

```

Then I can define projects this way:

```

(with-eval-after-load 'org
  (sacha-org-project-link
    "subed"
    "~/proj/subed/subed/"
    "https://github.com/sachac/subed/blob/main/subed/"
    ;; "https://codeberg.org/sachac/subed/src/branch/main/subed/"
  )
  (sacha-org-project-link
    "emacsconf-el"
    "~/proj/emacsconf/lisp/"
    "https://git.emacsconf.org/emacsconf-el/tree/"
  )
  (sacha-org-project-link
    "subed-record"
    "~/proj/subed-record/"
    "https://github.com/sachac/subed-record/blob/main/"
    ;; "https://codeberg.org/sachac/subed-record/src/branch/main/"
  )
  (sacha-org-project-link
    "compile-media"
    "~/proj/compile-media/"
    "https://github.com/sachac/compile-media/blob/main/"
    ;; "https://codeberg.org/sachac/compile-media/src/branch/main/"
  )
  (sacha-org-project-link
    "ox-11ty"
    "~/proj/ox-11ty/"
    "https://github.com/sachac/ox-11ty/blob/master/"
  )
  (sacha-org-project-link
    "11ty"
    "~/proj/static-blog/"
  )

```

```

"https://github.com/sachac/eleventy-blog-setup/blob/master/")
(sacha-org-project-link
 "emacsstv"
 "~/proj/emacsstv.github.io/"
 "https://github.com/emacsstv/emacsstv.github.io/blob/master/")
(sacha-org-project-link
 "quantified"
 "~/proj/quantified/"
 "https://github.com/sachac/quantified/blob/master/")
(sacha-org-project-link
 "emacs-news"
 "~/sync/emacs-news/"
 "https://github.com/sachac/emacs-news/blob/master/")
(sacha-org-project-link
 "speech-input"
 "~/proj/speech-input/"
 "https://codeberg.org/sachac/speech-input/src/branch/main/")
(sacha-org-project-link
 "learn-lang"
 "~/proj/learn-lang/"
 "https://codeberg.org/sachac/learn-lang/src/branch/main/")
(sacha-org-project-link
 "emacs.d"
 "~/sync/emacs/"
 "https://codeberg.org/sachac/.emacs.d/src/branch/gh-pages/"))

```

And I can complete them with the usual `C-c C-l` (`org-insert-link`) process:

[images/completing-custom-links.gif](#)

Sketches are handled by [my Org Mode sketch links](#) (p. 427), but we can add them anyway.

```

(cl-pushnew (cons (expand-file-name "~/sync/sketches/") ←
  ↪ "https://sketches.sachachua.com/filename/")
            sacha-project-web-base-list
            :test 'equal)

```

I've been really liking being able to refer to various emacsconf-el files by just selecting the link type and completing the filename, so maybe it'll be easier to write about lots of other stuff if I extend that to my other projects.

Copy web link

- [2024-01-20 Sat]: Fix Wayback link handling.
- [2024-01-19 Fri]: Add Wayback machine.

Keeping a list of projects and their web versions also makes it easier for me to get the URL for something. I try to post as much as possible on the Web so that it's easier for me to find things again and so that other people can pick up ideas from my notes. Things are a bit scattered: [my blog](#), repositories on [Github](#) and [Codeberg](#), [my sketches](#)... I don't want to think about where the code has ended up, I just want to grab the URL. If I'm going to put the link into an Org Mode document, that's super easy. I just take advantage of the things I've added to `org-store-link`. If I'm going to put it into an e-mail or a toot or wherever else, I just want the bare URL.

I can think of two ways to approach this. One is a command that copies just the URL by figuring it out from the buffer filename, which allows me to special-case a bunch of things:

```

;;;###autoload
(defun sachu-copy-link (&optional filename skip-links)
  "Return the URL of this file.
If FILENAME is non-nil, use that instead.
If SKIP-LINKS is non-nil, skip custom links.
If we're in a Dired buffer, use the file at point."
  (interactive)
  (setq filename (or filename
                      (if (derived-mode-p 'dired-mode) (dired-get-filename))
                      (buffer-file-name)))
  (if-let* ((project-re (concat "\\(" (regexp-opt (mapcar 'car sachu-project-web-base-list)) "\\))"
                        "\\(\\..*\\)"))
            (url (cond ((and (derived-mode-p 'org-mode)
                             (eq (org-element-type) 'link)
                             (not skip-links))
                        (pcase (org-element-property :type)
                          ((or "https" "http")
                           (org-element-property :path)
                           ("yt"
                            (org-element-property :path)
                            ;; if it's a custom link,
                            (_
                             (save-window-excursion
                               (org-open-at-point)
                               (sachu-copy-link nil)))
                           ;; links to my config usually have a
                           CUSTOM_ID property
                           ((string= (buffer-file-name)
                                       (expand-file-name "~/sync/emacs/Sacha.org"))
                            (concat "https://sachachua.com/dotemacs#"
                                     (org-entry-get-with-inheritance "CUSTOM_ID")))
                           ;; blog post drafts have permalinks
                           ((and (derived-mode-p 'org-mode)
                                (org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK"))
                            (concat "https://sachachua.com"
                                     (org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK")))
                           ;; some projects have web repos
                           ((string-match project-re filename)
                            (concat (assoc-default (match-string 1 filename) sachu-project-web-base-list)
                                     (url-hexify-string (match-string 2 filename))))))
                      (progn
                        (when (called-interactively-p 'any)
                          (kill-new url)
                          (message "%s" url))
                        (url-hexify-string (match-string 2 filename)))))))
    (url-hexify-string (match-string 2 filename))))))

```

```

      url)
    (error "Couldn't figure out URL.)))

```

Another approach is to hitch a ride on the Org Mode link storage and export functions and just grab the URL from whatever link I've stored with `org-store-link`, which I've bound to `C-c l`. I almost always have an HTML version of the exported link. We can even use XML parsing instead of regular expressions.

```

;; ##autoload
(defun sach-org-link-as-url (link)
  "Return the final URL for LINK."
  (cond
   ((string-match "^/" link)
    (concat sach-blog-base-url (replace-regexp-in-string "^/" "" link)))
   ((string-match "^\\(https://\\|file:\\)" link)
    link)
   (t
    (dom-attr
     (dom-by-tag
      (with-temp-buffer
        (insert (org-export-string-as link 'html t))
        (xml-parse-region (point-min) (point-max)))
      'a)
     'href))))

;; ##autoload
(defun sach-org-stored-link-as-url (&optional link insert)
  "Copy the stored link as a plain URL.
If LINK is specified, use that instead."
  (interactive (list nil current-prefix-arg))
  (setq link (or link (caar org-stored-links)))
  (let ((url (if link
                  (sach-org-link-as-url link)
                  (error "No stored link"))))
    (when (called-interactively-p 'any)
      (if url
          (if insert (insert url) (kill-new url))
          (error "Could not find URL.)))
    url))

(ert-deftest sach-org-stored-link-as-url ()
  (should
   (string= (sach-org-stored-link-as-url "[[dotemacs:web-link]]")
            "https://sachachua.com/dotemacs#web-link"))
  (should
   (string= (sach-org-stored-link-as-url "[[dotemacs:org-mode-sketch-links][my Org Mode ↩
↪ sketch links]]")
            "https://sachachua.com/dotemacs#org-mode-sketch-links")))

;; ##autoload
(defun sach-embark-org-copy-exported-url-as-wayback (link &rest _)
  (interactive "MLink: ")
  (let ((url (sach-embark-org-copy-exported-url link)))
    (when (not (string-match (regexp-quote "^https://web.archive.org") url))
      (setq url (concat "https://web.archive.org/web/" (format-time-string ↩
↪ "%Y%m%d%H%M%S/"))))

```

```

    url)))
    (when (called-interactively-p 'any)
      (kill-new url)
      (message "Copied %s" url))
    url))

;;;###autoload
(defun sach-embark-org-copy-exported-url (link &rest _)
  (interactive "MLink: \np")
  (let ((url (sach-org-link-as-url link)))
    (when (and (derived-mode-p 'org-mode)
               (org-entry-get-with-inheritance
                "EXPORT_ELEVENTY_PERMALINK"
                (string-match "^/" url))
               ;; local file links are copied to blog directories
               (setq url (concat "https://sachachua.com"
                                (org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK")
                                (replace-regexp-in-string
                                 "[\\?&].*"
                                 ""
                                 (file-name-nondirectory link))))))
      (when (called-interactively-p 'any)
        (kill-new url)
        (message "Copied %s" url))
      url))

;;;###autoload
(defun sach-embark-replace-link-with-exported-url (link &rest _)
  (interactive (list (org-element-property :raw-link (org-element-context))))
  (sach-insert-or-replace-link (sach-org-link-as-url link)))

```

```

(with-eval-after-load 'embark-org
  (mapc (lambda (map)
          (keymap-set map "u" #'sach-embark-org-copy-exported-url)
          (keymap-set map "U" ↵
            ↵ #'sach-embark-org-copy-exported-url-as-wayback)
          (keymap-set map "r e" ↵
            ↵ #'sach-embark-replace-link-with-exported-url))
        (list embark-url-map embark-org-link-map ↵
            ↵ embark-org-link-copy-map)))

```

We'll see which one I end up using. I think both approaches might come in handy.

Quickly search my code Since `sacha-project-web-base-list` is a list of projects I often think about or write about, I can also make something that searches through them. That way, I don't have to care about where my code is.

```

;;;###autoload
(defun sach-consult-ripgrep-code ()
  (interactive)
  (consult-ripgrep (mapcar 'car sach-project-web-base-list)))

```

I can add `.rgignore` files in directories to tell ripgrep to ignore things like `node_modules` or `*.json`.

I also want to search my Emacs configuration at the same time, although links to my config are handled by [my dotemacs link type](#) (p. 320) so I'll leave the URL as nil. This is also the way I can handle other unpublished directories.

```
(cl-pushnew (cons (expand-file-name "~/sync/emacs/Sacha.org") nil)
             sacha-project-web-base-list
             :test 'equal)
(cl-pushnew (cons (expand-file-name "~/proj/static-blog/_includes") nil)
             sacha-project-web-base-list
             :test 'equal)
(cl-pushnew (cons (expand-file-name "~/bin") nil)
             sacha-project-web-base-list
             :test 'equal)
```

Actually, let's throw my blog posts and Org files in there as well, since I often have code snippets. If it gets to be too much, I can always have different commands search different things.

```
(cl-pushnew (cons (expand-file-name "~/proj/static-blog/blog/") "https://sachachua.com/blog/")
             sacha-project-web-base-list
             :test 'equal)
(cl-pushnew (cons (expand-file-name "~/sync/orgzly") nil)
             sacha-project-web-base-list
             :test 'equal)
```

[images/ripgrep-code.gif](#)

I don't have anything bound to `M-s c` (code) yet, so let's try that.

```
(keymap-global-set "M-s c" #'sacha-consult-ripgrep-code)
```

At some point, it might be fun to get Embark set up so that I can grab a link to something right from the consult-ripgrep interface. In the meantime, I can always jump to it and get the link.

1. Tip from Omar: [embark-around-action-hooks](#) [2024-01-07 Sun] I modified oantolin's suggestion from the comments to work with `consult-ripgrep`, since `consult-ripgrep` gives me `consult-grep` targets instead of `consult-location`:

```
;; ##autoload
(cl-defun embark-consult--at-location (&rest args &key target type run &allow-other-keys)
  "RUN action at the target location."
  (save-window-excursion
    (save-excursion
      (save-restriction
        (pcase type
          ('consult-location (consult--jump ←
                               → (consult--get-location target)))
          ('org-heading (org-goto-marker-or-bmk ←
                               → (get-text-property 0 'org-marker target)))
          ('consult-grep (consult--jump ←
                           → (consult--grep-position target)))
          ('file (find-file target)))
        (apply run args))))
```

```
(cl-pushnew #'embark-consult--at-location (alist-get 'org-store-link ←
  → embark-around-action-hooks))
```

In general, I don't want to have to think about where something is on my laptop or where it's published on the Web, I just want to write about it. One step closer, yay Emacs!

[illegible]

So that I can easily add links at point. Formatted as an Org list for now.

339

```

;;;###autoload
(defun sach-get-youtube-info (url)
  (with-current-buffer (url-retrieve-synchronously url)
    (goto-char (point-min))
    (progn
      (list
        :url
        url
        :title
        (when (re-search-forward "<title>\\(.*?\\)</title>" nil t)
          (match-string 1))
        :duration
        (when (re-search-forward "approxDurationMs\\":\\\"\\([0-9]+\\)\\\" nil t)
          (format-seCONDS "%.2h:%.2m:%.2s%z" (/ (string-to-number (match-string 1)) 1000))))
      (kill-buffer))))

;;;###autoload
(defun sach-link-video (list)
  (when (stringp list) (setq list (list :url list)))
  (with-current-buffer (url-retrieve-synchronously (concat ←
    ↪ "https://video.link/bookmarklet?url=" (url-encode-url (plist-get list :url))))
    (save-excursion
      (if (re-search-forward "<input type=\"text\" id=\"safeURL\" readonly=\"readonly\" ↪
        ↪ value=\"\\(.*?\\)\\\" nil t)
          (plist-put list :url (match-string-no-properties 1))
          (plist-put list :url (replace-regexp-in-string "watch" "watch_popup" (plist-get list ↪
        ↪ :url)))))
    (when (string= (or (plist-get list :thumbnail) "") "")
      (save-excursion
        (when (re-search-forward "<img id=\"videoThumb\" src=\"\\(.*?\\)\\\" nil t)
          (plist-put list :thumbnail (match-string-no-properties 1)))))
    list))

;;;###autoload
(defun sach-org-protocol-copy-thumbnail (info)
  "Store and insert the link at point based on INFO."
  (interactive "MURL: ")
  (when (stringp info) (setq info (list :url info)))
  (when (string-match "youtube\\.com" (plist-get info :url))
    (setq info (sach-link-video info)))
  (let ((date (format-time-string "%Y-%m-%d")))
    (kill-new
      (if (string= (plist-get info :videoId) "")
          (format "{<thumbnail image=\"%s\" title=\"%s\" link=\"%s\" date=\"%s\">}}\\n"
            (plist-get info :thumbnail)
            (plist-get info :title)
            (plist-get info :url)
            date)
          (format "{<youtube id=\"%s\" title=\"%s\" link=\"%s\" date=\"%s\">}}\\n"
            (plist-get info :videoId)
            (plist-get info :title)
            (plist-get info :url)
            date))))
  nil)

```



```
(eval-after-load 'org-protocol
  '(add-to-list 'org-protocol-protocol-alist
    ("copy-thumbnail" :protocol "copy-thumbnail" :function ↔
      ↪ sach-org-protocol-copy-thumbnail)))
```

```
(use-package org-protocol-capture-html
  :vc (:url "https://github.com/alphapapa/org-protocol-capture-html"))
```

8.29.15 Fix elisp links

```
;;;###autoload
(defun sach-org-elisp-link-export (link description format &optional arg)
  (pcase format
    ('html (format "<span title=\"%s\">%s</span>" (replace-regexp-in-string "\"" "&quot;" link) ↔
      ↪ description))
    ((or 'icalendar 'ascii) description)
  ))
```

```
(org-link-set-parameters
  "elisp"
  :export 'sach-org-elisp-link-export)
```

8.29.16 IRC

```
(org-link-set-parameters
  "ircs"
  :export #'sach-org-irc-export)
```

```
;;;###autoload
(defun sach-org-ircs-export (link description format)
  "Export an ircs link.
See `org-link-parameters' for details about LINK, DESCRIPTION and
FORMAT."
  (let ((desc (or description link)))
    (pcase format
      ('html (format "<a href=\"%s\">%s</a>" link desc))
      ('md (format "[%s](ircs:%s)" desc link))
      (_ nil))))
```

8.29.17 Dired

```
(setq dired-dwim-target t)
```

```
;;;###autoload
(defun sach-org-get-links-in-region (beg end)
  (save-excursion
    (let (results)
      (goto-char (min beg end))
      (while (re-search-forward org-any-link-re (max beg end) t)
        (add-to-list 'results (org-element-context)))))
```

```

      results)))

;;;###autoload
(defun sacha-org-dired-file-links-in-region (beg end)
  "Display a Dired buffer for the file links in the selected region."
  (interactive "r")
  (let ((files
        (-map
         (lambda (x)
           (expand-file-name (org-link-unescape (plist-get (cadr x) :path))))
         (-filter
          (lambda (x)
            (string= (plist-get (cadr x) :type) "file"))
          (sacha-org-get-links-in-region beg end))))))
    (with-current-buffer (get-buffer-create "*Files*")
      (let ((inhibit-read-only t))
        (erase-buffer)
        (apply 'call-process "ls" nil t nil "-LR" files))
      (dired-virtual "/")
      (switch-to-buffer (current-buffer)))))

```

8.30 Org protocol: following Org links from outside Emacs

org:emacs

`_xor` had an interesting idea: can we use `org-protocol` to link to things inside Emacs, so that we can have a webpage with bookmarks into our Org files? Here's a quick hack that reuses `org-store-link` and `org-link-open`.

```

;;;###autoload
(defun org-protocol-open-link (info)
  "Process an org-protocol://open style url with INFO."
  (org-link-open (car (org-element-parse-secondary-string (plist-get info :link) ↵
    ↵ '(link)))))
;;;###autoload
(defun org-protocol-copy-open-link (arg)
  (interactive "P")
  (kill-new (concat "org-protocol://open?link=" (url-hexify-string (org-store-link arg)))))

(with-eval-after-load 'org-protocol
  (add-to-list 'org-protocol-protocol-alist
    '("org-open" :protocol "open" ↵
    ↵ :function org-protocol-open-link)))

```

To make exporting and following easier, we also need a little code to handle `org-protocol` links inside Org.

```

(defun org-protocol-follow (path &rest _)
  "Follow the org-protocol link for PATH."
  (org-protocol-check-filename-for-protocol (concat "org-protocol:" path) nil nil))

(defun org-protocol-export (path desc format info)
  "Export an org-protocol link."
  (setq path (concat "org-protocol:" path))
  (setq desc (or desc path))
  (pcase format
    ('html (format "<a href=\"%s\">%s</a>" path desc))
    ('11ty (format "<a href=\"%s\">%s</a>" path desc))

```

```
(`latex (org-latex-link path desc info))
(`ascii (org-ascii-link path desc info))
      (`md (org-md-link path desc info))
(_ path)))
```

```
(with-eval-after-load 'org
  (org-link-set-parameters "org-protocol"
    ↪
    ↪ :follow #'org-protocol-follow
    ↪
    ↪ :export #'org-protocol-export))
```

Now I can use `org-protocol-copy-open-link` to copy a link to the current location, and I can put it into my Org files.

Example bare link to the Org manual, which will work only if you have `open` in the `org-protocol-protocol-alist`:

`org-protocol://open?link=%5B%5Binfo%3Aorg%23Protocols%5D%5Borg%23Protocols%5D%5D`

With a description:

8.31 TODO Speed command for adding a custom ID to Org Mode posts

Nudged by [Amit's post about adding custom IDs to Org headings](#), I decided to write a speed command to add a custom ID with a reasonable default, and to make it happen whenever I post something from my Emacs config (like this one). I'm running out of brainspace for speed commands, so I'm going to try sticking it into a hydra so that I can add future things to the hydra instead. I'll probably figure out some kind of [cheat sheet thing](#) for speed commands too.

```
;; ##autoload
(defun sacha-make-slug (s)
  (thread-last
    s
    (learn-lang-replace-accents)
    (downcase)
    (replace-regexp-in-string "[^a-z0-9]+" "-")
    (replace-regexp-in-string "^-\\|-$" "")))

;; ##autoload
(defun sacha-org-set-custom-id (id)
  "Set the CUSTOM_ID property to ID at point."
  (interactive (list
    (let ((default-custom-id (sacha-make-slug (string-join (org-get-outline-path ↪
      ↪ t) " "))))
      (read-string (format "ID (%s): " default-custom-id) nil nil ↪
        ↪ default-custom-id))))
    (org-entry-put (point) "CUSTOM_ID" id)))

(defun sacha-org-assign-custom-ids ()
  (interactive)
  (let ((custom-ids
    (org-map-entries (lambda () (org-entry-get (point) ↪
      ↪ "CUSTOM_ID")) "CUSTOM_ID={.}")))
    (org-map-entries
      (lambda ()
        (let ((slug
```

```

                                (replace-regexp-in-string
                                "^-\\|-$" " "
                                (replace-regexp-in-string ↵
                                ↵
                                (downcase ↵
↵
↵
↵ (string-join (org-get-outline-path t) " "))))
                                (while (member slug custom-ids)
                                (setq slug (read-string "Manually set custom ID: "))
                                (org-entry-put (point) "CUSTOM_ID" slug))
                                "-CUSTOM_ID={.}"))

```

[2026-04-11 Sat] I haven't been using hydras much...

```

(with-eval-after-load 'hydra
  (define-key hydra-base-map (kbd "<down>") 'sacha-hydra-pop)
  (define-key hydra-base-map (kbd "<up>") (lambda () (interactive) (sacha-hydra-go-and-push ↵
    ↵ 'sacha-shortcuts/body)))

  (defhydra sacha-hydra/org-speed-commands ()
    ("i" sacha-org-set-custom-id "CUSTOM_ID" :exit t)
    ("<up>" sacha-hydra/org-mode/body :exit t)
    ("u" (sacha-hydra-go-and-push 'sacha-hydra/org-mode/body) :exit t :hint nil))
  (defhydra sacha-hydra/org-mode (:foreign-keys run)
    ("b" sacha-org-back-to-heading "Heading")
    ("n" org-forward-heading-same-level "Next")
    ("p" org-backward-heading-same-level "Previous")
    ("a" org-archive-subtree-default "Archive")
    ("j" sacha-org-mark-done-and-add-to-journal "Journal" :exit t)
    ("k" org-cut-subtree "Kill")
    ("<up>" (sacha-hydra-go-and-push 'sacha-shortcuts/body) :exit t :hint nil)
    ("u" (sacha-hydra-go-and-push 'sacha-shortcuts/body) :exit t :hint nil)
    ("<f14>" nil "Exit" :exit t))
  (defhydra sacha-hydra/org-link ()
    ("RET" org-open-at-point "Open")
    ("e" org-insert-link "Edit")
    ("c" sacha-caption-show "Captions")
    ("w" sacha-org-link-element-copy-link "Copy link")
    ("u" (sacha-hydra-go-and-push 'sacha-hydra/org-mode/body) :exit t :hint nil)
    ("<up>" (sacha-hydra-go-and-push 'sacha-hydra/org-mode/body) :exit t :hint nil))
  (defhydra sacha-hydra/org-src ()
    ("e" org-babel-execute-src-block "Exec")
    ("E" sacha-org-execute-src-block-by-name "Exec by name")
    ("i" org-edit-special "Edit")
    ("d" org-babel-demarcate-block "Demarcate")
    ("g" org-babel-goto-named-src-block "Goto")
    ("r" org-babel-open-src-block-result "Result")
    ("x" org-babel-expand-src-block "Expand")
    ("t" (org-babel-tangle '(4)) "Tangle at point")
    ("T" (org-babel-tangle '(16)) "Tangle target file")
    ("u" (sacha-hydra-go-and-push 'sacha-hydra/org-mode/body) :exit t :hint nil)
    ("<up>" (sacha-hydra-go-and-push 'sacha-hydra/org-mode/body) :exit t :hint nil)
    )
  ;; Not in a lisp/ file because it's very idiosyncratic
  (defun sacha-hydra/dwim ()
    (interactive)

```

```

(if (derived-mode-p 'org-mode)
  (let ((context (org-element-context)))
    (cond
      ((and (bolp) (looking-at org-outline-regexp))
       (sacha-hydra/org-speed-commands/body))
      ((org-in-src-block-p) (sacha-hydra/org-src/body))
      ((eq (org-element-type context) 'link) (sacha-hydra/org-link/body))
      (t (sacha-hydra/org-mode/body)))
    (sacha-shortcuts/body)))
(define-key org-mode-map (kbd "<f14>") 'sacha-hydra/dwim)
(keymap-global-set "<f14>" 'sacha-hydra/dwim))

```

8.32 Journal

```

(defvar sacha-journal-category-map
  '(("Gross" . "Gross motor")
    ("Fine" . "Fine motor")
    ("8 - Kaizen" . "Kaizen")
    ("9 - Us" . "Us")
    ("Self-care" . "Self-care and independence"))
  "Alist of string replacements for journal categories.")
(defvar sacha-journal-categories
  '("Kaizen" "Us" "Field trip" "Gross motor" "Fine motor"
    "Sensory" "Language" "Music" "Art"
    "Self-care and independence" "Eating" "Sleep" "Emotion"
    "Household" "Social" "Pretend" "Cognition" "World" "Other" "Oops" "Thoughts" "Consulting" ↵
    ↵ "Track" "Uncategorized")
  "List of categories to display.
   Unknown categories will be added to the end.")

;; ###autoload
(defun sacha-journal-date (o) (elt o 3))
;; ###autoload
(defun sacha-journal-note (o) (car o))
;; ###autoload
(defun sacha-journal-week-highlight (o) (elt o 4))
;; ###autoload
(defun sacha-journal-category (o) (elt o 1))
;; ###autoload
(defun sacha-journal-pictures (o) (when (string> (elt o 2) "") (split-string (elt o 2) ",")))
;; ###autoload
(defun sacha-journal-id (o) (elt o 7))
;; ###autoload
(defun sacha-journal-status (o) (elt o 8))
;; ###autoload
(defun sacha-journal-other (o) (elt o 9))
;; ###autoload
(defun sacha-journal-zidstring (o) (elt o 11))
;; ###autoload
(defun sacha-org-group-journal-entries (filtered &optional category-map categories)
  (setq category-map (or category-map sacha-journal-category-map))
  (setq categories (or categories sacha-journal-categories))
  (let* ((grouped (-group-by 'sacha-journal-category filtered))
        (mapped-list
         (mapcar
          (lambda (o)
            (cons (or (assoc-default (car o) category-map) (car o))

```

```

        (cdr o)))
      grouped))
    (sorted-list
      (delq nil
        (append
          (mapcar (lambda (cat)
                    (when (assoc-default cat mapped-list)
                      (cons cat (assoc-default cat mapped-list))))
            categories)
          (-remove (lambda (o) (member (car o) categories)) mapped-list))))
    sorted-list))

;;;###autoload
(defun sacha-org-date-to-string (date &optional base-date)
  "Return the Org date specified by DATE.
  This is relative to BASE-DATE if specified."
  (org-read-date nil nil date nil (when base-date (org-read-date nil t base-date))))

(ert-deftest sacha-org-date-to-string ()
  (should (string= (sacha-org-date-to-string "+1" "2018-08-01") "2018-08-02")))

;;;###autoload
(defun sacha-org-filter-journal-csv (filename &optional from to highlight base-date)
  "Return a list of matching entries."
  (setq from (and from (substring (sacha-org-date-to-string from base-date) 0 10))
    to (and to (substring (sacha-org-date-to-string to base-date) 0 10)))
  (let* ((data (pcsv-parse-file filename))
    (filtered
      (-filter
        (lambda (o)
          (let ((date (sacha-journal-date o)))
            (and (or (null from) (not (string< date from)))
              (or (null to) (string< date to))
              (and (not (string= (sacha-journal-status o) "Deleted"))
                (not (string-match "^!" (sacha-journal-note o)))
                (string-equal
                  "true"
                  (cond
                    ((null highlight) "true")
                    ((string-equal highlight "week") (sacha-journal-week-highlight o))
                    (t "true"))))))))
        data)))
    filtered))

;;;###autoload
(defun sacha-journal-read-category (&optional initial)
  (consult--read sacha-journal-categories :sort nil :prompt "Category: " :initial initial))

;;;###autoload
(defun sacha-journal-guess-category ()
  (when (derived-mode-p 'org-mode)
    (save-excursion
      (org-back-to-heading)
      (org-end-of-meta-data)
      (let ((text (buffer-substring-no-properties (point) (org-end-of-subtree))))
        (if (string-match "#gardening" text)
            "Household")))))

;;;###autoload

```



```

(defun sacha-journal-search-query-async (query-str next)
  (let* ((url-request-method "GET")
         (url-request-extra-headers (cons '("Content-Type" . "application/json") ←
         ↪ url-request-extra-headers)))
    (url-retrieve
     (format "%s/api/entries?q=%s&limit=50&sort=date&regex=1"
             sacha-journal-url
             query-str)
     (lambda (status)
       (goto-char (point-min))
       (re-search-forward "^$" nil t)
       (setq sacha-journal-search-cache
             (mapcar (lambda (o)
                       (cons
                        (format "%s %s"
                                (plist-get o :ZIDString)
                                (plist-get o :Note))
                        o))
                     (let ((json-object-type 'plist))
                       (json-read))))
              (funcall next 'flush)
              (if sacha-journal-search-cache (funcall next sacha-journal-search-cache))))))

;;;###autoload
(defun sacha-journal--async-search (next)
  (lambda (action)
    (cond
     ((eq action 'setup) ;; Should figure out how to start
      (sacha-journal-search-query-async "" next))
     ((and (stringp action) (not (string= action "")))
      (sacha-journal-search-query-async action next))
     (t (funcall next action)))))

;;;###autoload
(defun sacha-journal-completing-read ()
  (interactive)
  (consult--read
   (thread-first (consult--async-sink)
                  (consult--async-refresh-immediate)
                  (sacha-journal--async-search)
                  (consult--async-throttle)
                  (consult--async-split))
   :sort nil
   :prompt "Entry: "
   :category 'journal))

;;;###autoload
(defun sacha-journal-id-from-string (s)
  (when (string-match "^[-0-9]+" s) (match-string 0 s)))

;;;###autoload
(defun sacha-journal-view (s)
  (interactive (list (sacha-journal-completing-read)))
  (sacha-org-journal-open (sacha-journal-id-from-string s)))

;;;###autoload
(defun sacha-journal-sketch-large (zid)
  "Create a large sketch based on ZID."
  (interactive (list (sacha-journal-completing-read)))

```

```
(let ((filename (expand-file-name (format "%s.psd"
                                         (sacha-journal-id-from-string zid))
                                   sacha-sketch-inbox-directory)))
  (unless (file-exists-p filename)
    (copy-file sacha-sketch-large-template-file filename))
  (sacha-org-sketch-open filename)))
```

I should probably figure out how to switch this over to my Consult-based workflow:

```
;;;###autoload
(defun sacha-journal-format-entry (type o)
  (cond
   ((eq type 'org-link-zid-only)
    (org-link-make-string (format "journal:%s" (cdr (assoc 'ZIDString o)))))
   ((eq type 'list-item-with-zid)
    (format "- %s (%s)\n"
            (assoc-default 'Note o)
            (org-link-make-string
             (format "journal:%s" (assoc-default 'ZIDString o)))))
   ((eq type 'list-item)
    (format "- %s\n" (assoc-default 'Note o)))
   ((eq type 'text)
    (assoc-default 'Note o))))

;;;###autoload
(defun sacha-journal-format-entries (type list)
  (mapconcat
   (lambda (o) (sacha-journal-format-entry type o))
   (reverse list)
   (cond
    ((eq type 'org-link-zid-only) ", ")
    ((eq type 'list-item-with-zid) "")
    ((eq type 'list-item) "")
    ((eq type 'text) " "))))
```

This lets me define a custom link type.

```
;;;###autoload
(defun sacha-org-journal-open (id &optional arg)
  (browse-url (format "%s/zid/%s" sacha-journal-url id)))

;;;###autoload
(defun sacha-org-journal-export (link description format &optional arg)
  (let* ((path (concat "%s/zid/" sacha-journal-url link))
        (image (concat "%s/zid/" sacha-journal-url link))
        (desc (or description link)))
    (cond
     ((or (eq format 'html) (eq format 'wp))
      (if description
        (format "<a target=\"_blank\" href=\"%s\">%s</a>" path desc)
        (format "<a target=\"_blank\" href=\"%s\"><img src=\"%s\"><br />%s</a>" path image
              desc)))
     ((eq format 'latex) (format "\\href{%s}{%s}" path desc))
     ((eq format 'texinfo) (format "@uref{%s,%s}" path desc))
     ((eq format 'ascii) (format "%s <%s>" desc path))
     (t path))))

;;;###autoload
```

```
(defun sach-org-journal-complete (&optional prefix)
  (cdr (assoc 'ZIDString (helm-comp-read "Entry: " 'sacha-helm-journal-search :volatile t))))
```

```
(with-eval-after-load 'org
  (org-link-set-parameters
   "journal"
   :follow 'sacha-org-journal-open
   :export 'sacha-org-journal-export
   :complete 'sacha-org-journal-complete))
```

```
;;###autoload
(defun sach-org-journal-summarize (from to &optional search category-map categories)
  (sacha-org-group-journal-entries (sacha-journal-get-entries from to search) category-map <=>
    <=> categories))

;;###autoload
(defun sach-org-journal-format-tree (groups &optional include)
  (mapconcat
   (lambda (o)
     (concat "- *" (car o) "*\n"
              (mapconcat
               (lambda (i)
                 (concat "  - "
                          (if (member 'date include) (concat (sacha-journal-date i) " ") "")
                          (replace-regexp-in-string "\\\""" \"\" (sacha-journal-note i))
                          (if (member 'zid include) (concat " " (sacha-journal-zidstring i)) "")
                          ;; (if (string= "" (sacha-journal-category i))
                          ;;   ""
                          ;;   (format " (%s)" (sacha-journal-category i)))
                          "\n"))
                (reverse (cdr o)) "")))
    groups ""))

;;###autoload
(defun sach-org-summarize-journal-csv (from to &optional search category-map categories include)
  (interactive
   (list (org-read-date nil nil nil "From: ")
         (org-read-date nil nil nil "To: ")
         (read-string "Search: ")
         sach-journal-category-map
         sach-journal-categories
         nil))
  (let ((list (sach-org-journal-format-tree
                (sach-org-group-journal-entries
                 (sacha-journal-get-entries from to search)
                 category-map categories)
                include)))
    (if (called-interactively-p 'any) (insert list) list)))
```

```
;;###autoload
(defun sach-read-journal-category ()
  (completing-read "Category: " sach-journal-categories))

;;###autoload
(defun sach-update-journal-entry (old-text new-text category)
  (interactive (list (read-string "Old: ")))
```

```

        (read-string "New: ")
        (sacha-read-journal-category)))
(sacha-send-intent "com.sachachua.journal.categorize"
  (list (cons "text" old-text)
        (cons "newtext" (or new-text old-text))
        (cons "category" (or category "Uncategorized")))))

;;###autoload
(defun sacha-create-journal-entry (new-text category)
  (interactive (list (read-string "Text: ")
                    (sacha-read-journal-category)))
  (sacha-update-journal-entry new-text new-text category))

;;###autoload
(defun sacha-export-journal-entries ()
  "Trigger task to export. Phone must be unlocked."
  (interactive)
  (sacha-send-intent "com.sachachua.journal.export" '(("a" . "b"))))

```

```

(use-package csv
  :commands csv--read-line)

```

```

;;###autoload
(defun sacha-prompt-for-uncategorized-entries ()
  (interactive)
  (let ((key-list '("Note" "Date" "highlight week" "Category" "month" "Time" "Link" "ELECT")))
    x new-text category done)
    (while (and (not (eobp)) (not done))
      (forward-char 1)
      (setq x (csv--read-line key-list))
      (when (string= (assoc-default "Category" x nil "") "")
        (setq text (read-string "Text: " (assoc-default "Note" x nil "")))
        (setq category (completing-read "Category: " (cons "." sacha-journal-categories)))
        (if (string= category ".")
            (setq done t)
            (sacha-update-journal-entry (assoc-default "Note" x nil "") text category))))))

```

8.32.1 Working with journal entries

```

;;###autoload
(defun sacha-journal-insert-matching-entries (from to match)
  (interactive (list (org-read-date "From: ") (org-read-date "To: ") (read-string "Match: ")))
  (insert
   (mapconcat
    (lambda (o)
      (format "- %s %s" (sacha-journal-zidstring o) (sacha-journal-note o)))
    (seq-filter (lambda (o) (string-match match (sacha-journal-other o)))
                (sacha-journal-get-entries from to))
    "\n"))
  ;;###autoload
(defun sacha-journal-convert-to-refs (beg end)
  (interactive "r")
  (save-restriction
    (goto-char beg)
    (narrow-to-region beg end)

```

```

(while (re-search-forward "^- \\([0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]\\) ←
  ↪ .*? $" nil t)
  (replace-match "ref:\\1"))))
;;;###autoload
(defun sachajournal-get-refs-from-region (beg end)
  (interactive "r")
  (save-excursion
    (goto-char beg)
    (cl-loop for pos = (re-search-forward " ←
  ↪ \\([0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]\\) " end t)
      while pos
        collect (match-string 1))))

;;;###autoload
(defun sachajournal-add-tag (tag beg end)
  (interactive "MTag: \nr")
  (let* ((url-request-method "POST")
    (url-request-extra-headers '(("Content-Type" . "application/json")))
    (zids (sachajournal-get-refs-from-region beg end))
    (json-object-type 'plist)
    (url-request-data (json-encode-plist (list :zids zids :tags (split-string tag " " )))))
    (pp (sachajournal-get "api/entries/tag/bulk"))))

;;;###autoload
(defun sachajournal-remove-tag (tag beg end)
  (interactive "MTag: \nr")
  (let* ((url-request-method "DELETE")
    (url-request-extra-headers '(("Content-Type" . "application/json")))
    (zids (sachajournal-get-refs-from-region beg end))
    (json-object-type 'plist)
    (url-request-data (json-encode-plist (list :zids zids :tags (split-string tag " " )))))
    (pp (sachajournal-get "api/entries/tag/bulk"))))

;;;###autoload
(defun sachajournal-post-with-refs (note date other beg end)
  (interactive (list
    (read-string "Note: ")
    (concat (org-read-date "Date: ") " 23:00")
    (read-string "Other: ")
    (min (point) (mark))
    (max (point) (mark))))
  (sachajournal-post note :Date date :Other (concat other "\n"
    (mapconcat (lambda (o) (concat "ref:" o))
      ↪
      ↪ (sachajournal-get-refs-from-region beg end)
        " "))))

;;;###autoload
(defun sachajournal-browse-current-day ()
  (interactive)
  (browse-url
    (format "https://journal.sachachua.com/day/%s"
      (format-time-string "%Y-%m-%d"
        ↪
        ↪ (sachajournal-filename-timestamp (buffer-file-name))))))

```

8.32.2 Tagging journal entries

```

;;;###autoload
(defun sachajournal-list-toggle-monthly-highlight ()
  (interactive)
  (let ((entry (tabulated-list-get-entry)))
    (setf (elt entry 3) (if (string-match "#monthly-highlight" (elt entry 3))
                             ↵
                             (replace-regexp-in-string "?#monthly-highlight" "" (elt entry 3))
                             ↵
                             (string-trim (concat (elt entry 3) " #monthly-highlight")))))
    (sachajournal-update
     (list :ZIDString (elt entry 0)
           :Other (elt entry 3)))
    (tabulated-list-print t t)))

;;;###autoload
(defun sachajournal-list-echo ()
  (interactive)
  (message "%s -- %s" (elt (tabulated-list-get-entry) 2) (elt (tabulated-list-get-entry) ↵
                                                                ↵ 3)))

(defvar-keymap sachajournal-list-mode-map
  :parent tabulated-list-mode-map
  "t" #'sachajournal-list-toggle-monthly-highlight
  "v" #'sachajournal-list-echo)

(define-derived-mode sachajournal-list-mode tabulated-list-mode "Journal"
  "Major mode for journal entries."
  (setq tabulated-list-format [("ZID" 14 t)
                               ↵
                               ("Category" 10 t)
                               ↵
                               ("Note" 80 nil)
                               ↵
                               ("Other" 30 nil)])
  (tabulated-list-init-header)
  (tabulated-list-print t))

;;;###autoload
(defun sachajournal-list (start end filter)
  (interactive (list (org-read-date "Start: ") (org-read-date "End: ")
                    (read-string ↵
                      "Filter: ")))
  (switch-to-buffer (get-buffer-create "*journal*"))
  (setq tabulated-list-entries
        (mapcar
         (lambda (row)
           (list
            (sachajournal-zidstring row)
            (vector
             (sachajournal-zidstring row)
             (sachajournal-category row)
             (replace-regexp-in-string "\n" " " ↵
                                       (sachajournal-note row))
             (replace-regexp-in-string "\n" " " ↵
                                       (sachajournal-other row))))))
         (sachajournal-get-entries start end filter)))
  (sachajournal-list-mode))

```

```

;;;###autoload
(defun sacha-get-image-caption (file)
  (let ((caption (shell-command-to-string (format "exiftool -s -s -s -ImageDescription %s" ↵
    ↵ (shell-quote-argument file))))))
    (when (> (length caption) 0) (format "#+CAPTION: %s" caption))))

;;;###autoload
(defun sacha-insert-image-link-with-caption (file)
  (let ((caption (sacha-get-image-caption file)))
    (insert (or caption "") (org-link-make-string file) "\n")))

;;;###autoload
(defun sacha-caption-current-image ()
  (interactive)
  (let ((link (org-element-link-parser)) caption)
    (when (and link (org-element-property :path link))
      (setq caption (sacha-get-image-caption (org-element-property :path link)))
      (when caption (insert caption)))))

;;;###autoload
(defun sacha-set-image-caption (file caption)
  (interactive (list (if (derived-mode-p 'dired-mode) (dired-get-filename) (buffer-file-name))
    (read-string "Caption: ")))
  (shell-command (format "exiftool -ImageDescription=\"%s\" %s" (shell-quote-argument caption) ↵
    ↵ (shell-quote-argument file))))

(defvar sacha-photo-directory "/mnt/nfs/photos/inbox")
;;;###autoload
(defun sacha-get-photo-rating (file)
  (let ((rating (shell-command-to-string (concat "exiftool -s -s -s -Rating " ↵
    ↵ (shell-quote-argument file))))))
    (string-to-number rating)))

;;;###autoload
(defun sacha-make-photo-list (start end &optional rating require-description)
  (interactive (list (org-read-date "Start: ") (org-read-date "End: ")))
  (-filter
    (lambda (filename)
      (and (string> (file-name-nondirectory filename) start)
        (string> end (file-name-nondirectory filename))
        (if rating (>= (sacha-get-photo-rating filename) rating) t)
        (if require-description (sacha-get-image-caption filename) t)))
    (directory-files sacha-photo-directory t ".*\\.jpg$"))

;;;###autoload
(defun sacha-org-get-photo (id)
  "Open the photo identified by ID."
  (car (directory-files sacha-photo-directory t (concat id ".*\\.jpg"))))

;;;###autoload
(defun sacha-org-open-photo (id)
  (find-file (sacha-org-get-photo id)))

(sacha-make-photo-list "2018-06-10" "2018-06-15" nil t)
(sacha-get-photo-rating (sacha-org-get-photo ↵
  ↵ "2018-06-10-18-16-31"))

```

```

;;;###autoload
(defun sacha-org-significant-moments (start end &optional rating)
  (interactive (list (org-read-date "Start: ") (org-read-date "End: ") 3))
  (let ((result
        (mapconcat (lambda (file)
                      (let ((caption (sacha-get-image-caption file)))
                        (if caption
                          (concat caption (org-link-make-string file) "\n")
                          (concat (org-link-make-string file) "\n")))))
                    (sacha-make-photo-list start end 3)
                    "\n")))
    (if (called-interactively-p 'any) (insert result) result)))

```

8.32.4 Moments

```

;;;###autoload
(defun sacha-journal-moments (date)
  (interactive (list (org-read-date "Start: ")))
  (sacha-journal-post (concat "Moments starting " date " #moment") :Date (concat date " ↵
↵ 23:00") :Category "Thoughts"))

```

8.32.5 Slicing and dicing the journal entries

```

;;;###autoload
(defun sacha-journal-filter-by-category (category list)
  (reverse (seq-filter (lambda (o) (string= (sacha-journal-category o) "Eating")) list)))

;;;###autoload
(defun sacha-journal-group-by-month (list)
  (seq-group-by (lambda (o)
                  (substring ↵
↵ (sacha-journal-date o) 0 7))
                list))

;;;###autoload
(defun sacha-journal-filter-by-month (month-regexp list)
  (seq-filter (lambda (o)
                (string-match month-regexp ↵
↵ (substring (sacha-journal-date o) 5 7)))
              list))

;;;###autoload
(defun sacha-journal-group-by-month-day (list)
  (seq-group-by (lambda (o)
                  (substring ↵
↵ (sacha-journal-date o) 5))
                list))

;;;###autoload
(defun sacha-journal-list-with-day (list)
  (mapconcat (lambda (o)
                (concat " - " (substring ↵
↵ (sacha-journal-date o) 8) " "
↵ (replace-regexp-in-string "#.*" "" (sacha-journal-note o))))
              list
              "\n"))

```



```

;;;###autoload
(defun sachajournal-list-with-year (list)
  (mapconcat (lambda (o)
    (concat " - " (substring (sachajournal-date o) 0 4) " ")
    (sachajournal-note o)))
  list
  "\n"))

;;;###autoload
(defun sachajournal-this-month-by-day (list)
  (mapconcat (lambda (group)
    (format "- %s\n%s"
      (car group)
      (sachajournal-list-with-year (cdr group))))
    (cl-sort (sachajournal-group-by-month-day
      (sachajournal-filter-by-month (format-time-string "%02m")))
      'string<
      :key #'car)
    "\n"))

```

8.33 Attachments

Org lets you attach files to an Org file. Haven't gotten the hang of this yet, but looks interesting.

```

(use-package org-attach
  :ensure nil
  :config
  (setq org-attach-store-link-p 'attached)
  (setq org-attach-auto-tag nil))

```

8.34 HTTP

```

(use-package ob-http :defer t)

```

8.35 Lilypond

```

(use-package lilypond-init
  :if sachalaptop-p
  :load-path "~/vendor/lilypond/elisp"
  :config
  (setq org-babel-lilypond-arrange-mode t
    org-babel-lilypond-commands '("lilypond" "timidity" "timidity")
    org-babel-lilypond-gen-pdf nil
    org-babel-lilypond-display-pdf-post-tangle nil)
  :mode ("\\.ly\\'" . LilyPond-mode))

```

8.36 Diagrams and graphics

Ooooh. Graphviz and Ditaai make it easier to create diagrams from Emacs. See <http://sachachua.com/evil-plans> for examples and source.

The `pikchr-cli` package in Ubuntu 24 is version 0.1.2 and does not have the `--svg-only` argument that `pikchr-mode` uses, so I followed the instructions at <https://pikchr.org/home/doc/trunk/doc/download.md> to download and compile `pikchr.c`.

```
;also includes Org Babel support
(use-package pikchr-mode
  :defer t
  :config
  (setq pikchr-executable "/home/sacha/vendor/pikchr/pikchr"))
```

```
(setq org-ditaa-jar-path "c:/sacha/Dropbox/bin/ditaa.jar")
(use-package org-contrib)
(use-package org
  :config
  (add-hook 'org-babel-after-execute-hook 'org-link-preview)
  (setf org-confirm-babel-evaluate (lambda (lang body)
    ↪                                     ↪
    ↪                                     (pcase (or (buffer-file-name) "")
    ↪                                     ↪
    ↪                                     ((rx (or "vendor" ↪
    ↪                                     ↪
    ↪                                     ↪
    ↪                                     ((rx (or "proj" ↪
    ↪                                     ↪
    ↪                                     ↪
    ↪                                     (_ t))))
    (setf org-link-elisp-confirm-function
      (lambda (prompt)
        (if (and (buffer-file-name) (string-match "vendor" (buffer-file-name)))
            (y-or-n-p prompt)
            t)))
      (require 'ob-ledger)
      (org-babel-do-load-languages
        'org-babel-load-languages
        '((dot . t)
          (ditaa . t)
              (pikchr . t)
          (forth . t)
              (gnuplot . t)
              (mermaid . t)
          (emacs-lisp . t)
          (plantuml . t)
          (lilypond . t)
          (python . t)
          (ruby . t)
          (shell . t)
          (calc . t)
          (js . t)
          (sqlite . t)
          (http . t)
              (org . t)
          (ledger . t)
          (shell . t)
```

```
(R . t)))
(setq org-babel-python-command "python3")
(setq python-shell-interpreter "python3")
  (add-to-list 'org-src-lang-modes '("html" . web))
(add-to-list 'org-src-lang-modes '("dot" . graphviz-dot)))
```

8.36.1 Treemap visualization

Treemap visualization of an Org Mode file

```
(defvar sach-org-treemap-temp-file "~/Downloads/treemap.html") ; Firefox inside Snap can't ↵
↵ access /tmp

(defvar sach-org-treemap-command "treemap" "Executable to generate a treemap.")

;;;###autoload
(defun sach-org-treemap-include-p (node)
  (not (or (eq (org-element-property :todo-type node) 'done)
            (member "notree" (org-element-property :tags node))
            (org-element-property-inherited :archivedp node ↵
↵ 'with-self))))

;;;###autoload
(defun sach-org-treemap-data (node &optional path)
  "Output the size of headings underneath this one."
  (let ((sub
        (apply
         'append
         (org-element-map
          (org-element-contents node)
          '(headline)
          (lambda (child)
            (if (sach-org-treemap-include-p child)
                (sach-org-treemap-data
                 child
                 (append path ↵
↵ (list
↵ (org-no-properties
↵ (org-element-property :raw-value node))))))
            (list
             (list
              (-
               (org-element-end child)
               (org-element-begin ↵
↵ child))
              (string-join
               (cdr
                (append path ↵
↵ (list
↵ (org-no-properties
↵ (org-element-property :raw-value node))
↵ (org-no-properties
```

```

                                (org-element-property :raw-value child))))
                                ("/")
                                nil))))
                                nil nil 'headline))))

(append
 (list
  (list
   (-
    (org-element-end node)
    (org-element-begin node)
    (apply '+ (mapcar 'car sub))
    )
   (string-join
    (cdr
     (append path
              (list
               (org-no-properties
                (org-element-property :raw-value node))))
              ("/")
              (sacha-org-treemap-include-p node)))
    sub)))

;;###autoload
(defun sacha-org-treemap ()
  "Generate a treemap."
  (interactive)
  (save-excursion
    (goto-char (point-min))
    (let ((file (expand-file-name (expand-file-name sacha-org-treemap-temp-file)))
          (data (cdr (sacha-org-treemap-data
                     (org-element-parse-buffer))))
          (with-temp-file file
            (call-process-region
             (mapconcat
              (lambda (entry)
                (if (elt entry 2)
                    (format "%d %s\n" (car entry)
                                (replace-regexp-in-string org-link-bracket-re "\\2" (cadr entry))
                                ""))
              data
              ""))
            nil
            sacha-org-treemap-command nil t t))
      (browse-url (concat "file://" (expand-file-name
                                sacha-org-treemap-temp-file))))))

```

8.37 Org Babel, Mermaid JS, and fixing "Failed to launch the browser process" on Ubuntu 24 org

[Mermaid](#) makes pretty diagrams from text. It's Javascript-based, so the command-line tool (mmdc) uses Puppeteer to get the results of evaluating the diagram in the browser. I was running into some errors trying to get it to work from Org Mode over [ob-mermaid](#) on Ubuntu 24, since apparently [AppArmor](#) restricts Puppeteer. (Error: Failed to launch the browser process! ð Issue #730 ð [mermaid-js/mermaid-cli](#)).

I put together a [pull request to modify ob-mermaid-cli-path](#) so that it doesn't get quoted and can therefore have the `aa-exec` command needed to work around that. With that modified `org-babel-execute:mermaid`, I can then configure `ob-mermaid` like this:

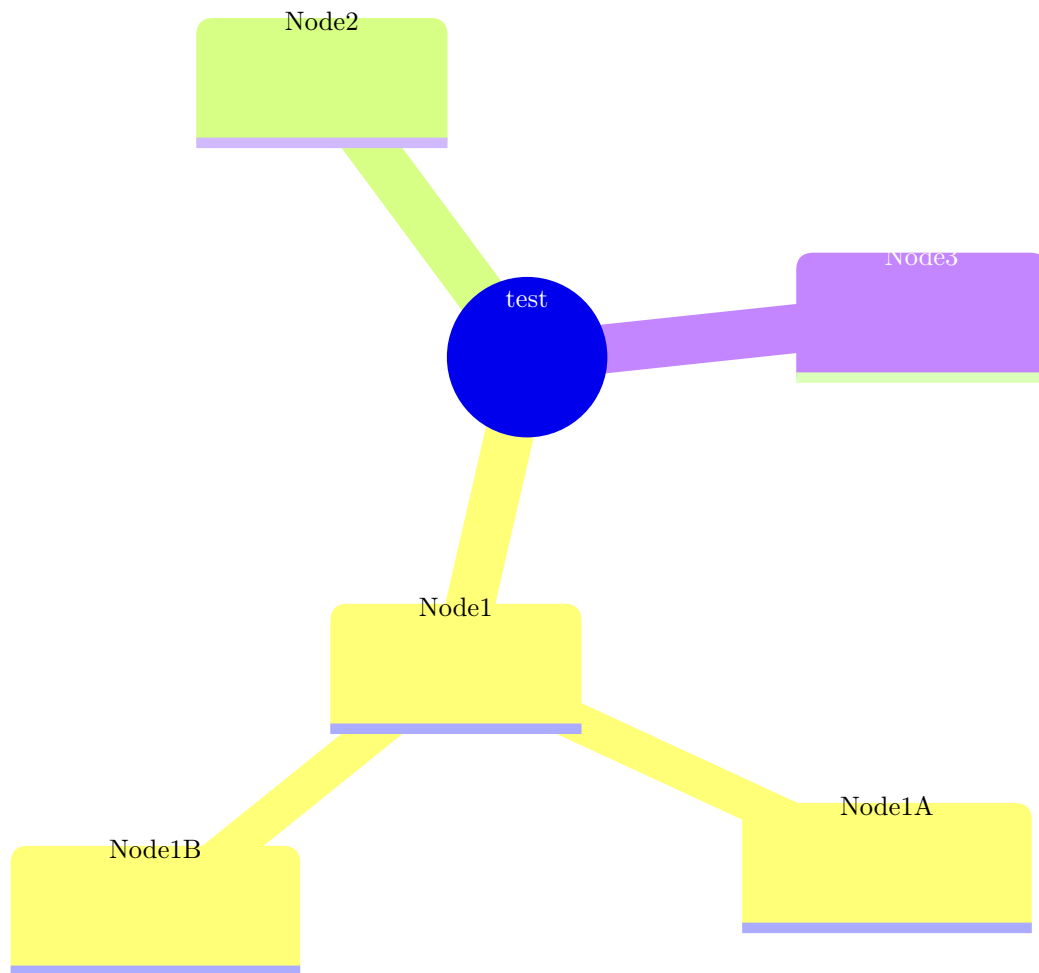
```
(use-package ob-mermaid
  :load-path "~/vendor/ob-mermaid")
;; I need to override this so that the executable isn't quoted
(setq ob-mermaid-cli-path "aa-exec --profile chrome mmdc -c ~/.config/mermaid/config.json")
```

I also ran into a problem where the library that Emacs uses to display SVGs could not handle the `foreignObject` elements used for the labels. [mermaid missing text in svg](#) & [Issue #112](#) & [mermaid-js/mermaid-cli](#). Using the following `~/.config/mermaid/config.json` fixed it, and I put the option in the `ob-mermaid-cli-path` above so that it always gets loaded.

```
{
  "flowchart": {
    "useMaxWidth": false,
    "htmlLabels": false
  }
}
```

Here's sample Mermaid markup and the file it creates:

```
mindmap
  root((test))
    Node 1
      Node 1A
      Node 1B
    Node 2
    Node 3
```



Now I can see the labeled diagrams inside Emacs, too.

8.38 Counting

Good way to remind myself that I have lots of STARTED tasks.

```

;;;###autoload
(defun sacha-org-summarize-task-status ()
  "Count number of tasks by status.
  Probably should make this a dblock someday."
  (interactive)
  (let (result)
    (org-map-entries
     (lambda ()
       (let ((todo (elt (org-heading-components) 2)))
         (if todo
             (if (assoc todo result)
                 (setcdr (assoc todo result)
                         (1+ (cdr (assoc todo result))))
                 (setq result (cons (cons todo 1) result))))))
      (message "%s" (mapconcat (lambda (x) (format "%s: %d" (car x) (cdr x)))
                              result "\n")))))

```

8.39 Spreadsheets

```
;;;###autoload
(defun sacha-org-days-between (start end)
  "Number of days between START and END (exclusive).
   This includes START but not END."
  (- (calendar-absolute-from-gregorian (org-date-to-gregorian end))
     (calendar-absolute-from-gregorian (org-date-to-gregorian start))))
```

8.40 Literate programming

8.40.1 Editing source code

I don't want to get distracted by the same code in the other window, so I want org src to use the current window.

```
(setq org-src-window-setup 'current-window)
```

8.40.2 Copying and sharing code

```
;;;###autoload
(defun sacha-copy-code-as-org-block-and-gist (beg end)
  (interactive "r")
  (let ((filename (or (file-name-base) ""))
        (mode (symbol-name major-mode))
        (contents
         (if (use-region-p) (buffer-substring beg end) (buffer-string)))
        (gist (if (use-region-p) (gist-region beg end) (gist-buffer))))
    (kill-new
     (format "\n%s\n#+begin_src %s\n%s\n#+end_src\n"
             (org-link-make-string (oref (oref gist :data) :html-url) filename)
             (replace-regexp-in-string "-mode$" "" mode)
             contents))))
```

8.40.3 Tables

Requires dash.

```
;;;###autoload
(defun sacha-org-table-as-alist (table)
  "Convert TABLE to an alist. Remember to set :colnames no."
  (let ((headers (seq-map 'intern (car table))))
    (cl-loop for x in (cdr table) collect (-zip headers x))))
```

8.41 Invoices

```
(setq calendar-week-start-day 6) ;; My weeks start on Saturday
```

```
;;;###autoload
(defun sacha-org-get-invoice-range-based-on-date (date)
  (let* ((invoice-date (org-date-to-gregorian date))
```

```

        (start (list (1- (car invoice-date)) 1 (elt invoice-date 2)))
        (end (list (car invoice-date) 1 (elt invoice-date 2))))
    (mapcar (lambda (date)
              (format-time-string "%F %H:%M" (encode-time 0 0 0 1 (elt date 0) (elt date 2))))
            (list start end))))

;; ###autoload
(defun sach-org-quantified-get-hours-based-on-range (category start end)
  "Return the number of hours for the specified category."
  (/ (assoc-default category
                    (quantified-summarize-time start end)) 3600.0))

;; TODO: paginate
;; ###autoload
(defun sach-org-quantified-get-detailed-hours-based-on-range (category start end)
  "Return a list of (date week-ending-date dow seconds) for CATEGORY from START to END."
  (let ((entries
        (quantified-parse-json
         (quantified-request (format ~<
~> "records.json?start=%s&end=%s&filter_string=%s&per_page=1000&split=split" start end ~<
~> (url-encode-url category))
                             nil "GET"))))
    (mapcar
     (lambda (entry)
       (let ((time (date-to-time (assoc-default 'timestamp entry))))
         (list
          (format-time-string "%F" time)
          (format-time-string "%F" (sach-org-get-week-end-for-time time))
          (format-time-string "%a" time)
          (assoc-default 'duration entry))))
      entries)))

;; ###autoload
(defun sach-org-get-week-end-for-time (time &optional week-ends-on-day)
  "WEEK-ENDS-ON-DAY: 0 is Sunday"
  (let* ((decoded (decode-time time))
         (dow (elt decoded 6))
         (end-week (or week-ends-on-day (% (+ 6 calendar-week-start-day) 7))))
    (encode-time
     (elt decoded 0)
     (elt decoded 1)
     (elt decoded 2)
     (+ (elt decoded 3)
        (% (+ 7 (- end-week dow)) 7))
     (elt decoded 4)
     (elt decoded 5))))

(ert-deftest sach-org-get-week-ending-date ()
  (let ((calendar-week-start-day 6)
        (tests '(
                  ("2015-09-03" . "2015-09-04")
                  ("2015-12-01" . "2015-12-04")
                  ("2015-12-03" . "2015-12-04")
                  ("2015-12-04" . "2015-12-04")
                  ("2015-12-05" . "2015-12-11")))))
    (dolist (test tests)
      (should (string=
                    (format-time-string
                     "%F"

```



```

        (sacha-get-week-end-for-time (org-time-string-to-time (car test))))
        (cdr test)))
(should (string=
  (format-time-string
    "%F"
    (sacha-get-week-end-for-time (org-time-string-to-time (car test)) 5))
  (cdr test))))))

;;###autoload
(defun sacha-org-quantified-format-detailed-hours-as-table (list)
  "Return a table with rows for LIST.
  | Week ending ____ | Sat | Sun | Mon | Tue | Wed | Thu | Fri | Total |
  LIST elements should be in the form (date week-end-date dow seconds).
  See `sacha-org-quantified-get-detailed-hours-based-on-range'."
  ;; Group by week ending date
  (let ((days '("Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri"))))
    (append
      (list (append '("Week ending") days '("Total"))))
      (mapcar
        (lambda (row)
          (let ((day-values (-group-by (lambda (x) (elt x 2)) (cdr row)))
                (week-total 0))
            (append
              (list (format "Week ending %s" (format-time-string "%b %-e" ↵
↵ (org-time-string-to-time (car row))))))
              (mapcar (lambda (day)
                        (if (assoc-default day day-values)
                            (format "%.1f"
                                (apply '+
                                    (mapcar
                                      (lambda (day-val) (/ (elt day-val 3) 3600.0))
                                      (assoc-default day day-values))))
                            ""))
                        days)
              (list (format "%.1f"
                            (apply '+ (mapcar (lambda (day-val) (/ (elt day-val 3) 3600.0)) (cdr ↵
↵ row))))))
                ))
          )
        (-sort (lambda (a b) (string< (car a) (car b))) (-group-by (lambda (x) (elt x 1)) ↵
↵ list))))))

;;###autoload
(defun sacha-org-quantified-hours-table ()
  (sacha-org-quantified-format-detailed-hours-as-table
    (apply 'sacha-org-quantified-get-detailed-hours-based-on-range
      (org-entry-get-with-inheritance "QUANTIFIED_CATEGORY")
      (sacha-org-get-invoice-range-based-on-date (org-entry-get-with-inheritance ↵
↵ "INVOICE_DATE")))))

(ert-deftest sacha-org-get-invoice-range-based-on-date ()
  "Check if invoice range is sane."
  (should (equal (sacha-org-get-invoice-range-based-on-date "2015-12-05")
    ("2015-11-01 00:00" "2015-12-01 00:00"))))

```

8.42 Presentations

```
(use-package org-re-reveal
  :config
  (setq org-re-reveal-revealjs-version "4")
  (setq org-re-reveal-history t)
  :defer t)
(use-package oer-reveal
  :defer t
  :config
  (setq oer-reveal-plugin-4-config
        "audioslideshow RevealAudioSlideshow ↵
↵ plugin/audio-slideshow/plugin.js
anything RevealAnything ↵
↵ https://cdn.jsdelivr.net/npm/reveal.js-plugins@latest/anything/plugin.js"))
```

8.42.1 Counting words

```
(defvar sach-org-note-words-target (* 140 20))
;;###autoload
(defun sach-org-collect-notes (&optional block-name)
  (let (results)
    (org-block-map
     (lambda ()
       (unless (org-in-commented-heading-p)
         (let ((elem (org-element-at-point)))
           (when (string= (downcase (org-element-property :type) ↵
↵ elem)))
↵
↵ (or block-name "notes"))
↵ (push (string-trim
↵
↵ (buffer-substring-no-properties
↵
↵ (org-element-property :contents-begin elem)
↵
↵ (org-element-property :contents-end elem)))
↵ results))))))
    (reverse results)))

;;###autoload
(defun sach-org-count-words-in-notes (&optional target block-name)
  "Count words in #+begin_notes blocks.
If TARGET or `sach-org-note-words-target' is specified, calculate percentage and words left.
If BLOCK-NAME is specified, use that block type instead."
  (interactive)
  (let ((notes (sach-org-collect-notes)))
    (with-temp-buffer
      (insert (string-join notes "\n"))
      (let ((num (count-words-region (point-min) (point-max))))
        (if (or target sach-org-note-words-target)
            (message "%d words (%.f%% of %d, %d to go)"
                      num
                      (/ (* 100.0 ↵
↵ num) sach-org-note-words-target)
↵
↵ sach-org-note-words-target
```

```
(- ↩)
↪ sachal-org-note-words-target num))
      (message "%d words" num)))))))

;;###autoload
(defun sachal-org-create-notes-buffer ()
  (interactive)
  (let ((notes (sachal-org-collect-notes)))
    (with-current-buffer (get-buffer-create "*Notes*")
      (insert (string-join notes "\n\n"))
      (switch-to-buffer (current-buffer)))))
```

8.43 Allow dashes in tags

```
;;###autoload
(defun sach-org-add-dashes-to-tag-regexps ()
  (setq org-complex-heading-regexp
    (concat "^\\((\\|*+\\|)\"
      "\\(?: + org-todo-regexp \"\\|)\"?\"
      "\\(?: +\\((\\|#[#\\.\\|\\|\\|\\|)\\|)\"?\"
      "\\(?: +\\((\\.\\|*?\\|\\|\\|)?? \"
      "\\(?:[ \\t]+\\|\\(:[-[:alnum:]]_@#%:[+:(\\|\\|)\\|)\"?\"
      "[ \\t]*$)\"
    org-complex-heading-regexp-format
    (concat "^\\((\\|*+\\|)\"
      "\\(?: + org-todo-regexp \"\\|)\"?\"
      "\\(?: +\\((\\|#[#\\.\\|\\|\\|\\|)\\|)\"?\"
      "\\(?: + \"
      ;; Stats cookies can be stuck to body.
      "\\(?::\\|\\[[0-9%]/\\|+\\|) *\\|)\"*\"
      "\\(?:%$\\|)\"
      "\\(?: + *\\|\\[[0-9%]/\\|+\\|\\|)\\|)\"*\"
      "\\|)\"
      "\\(?::[ \\t]+\\|\\(:[-[:alnum:]]_@#%:[+:(\\|\\|)\\|)\"?\"
      "[ \\t]*$)\"
    org-todo-line-tags-regexp
    (concat "^\\((\\|*+\\|)\"
      "\\(?: + org-todo-regexp \"\\|)\"?\"
      "\\(?: +\\((\\.\\|*?\\|\\|\\|)\\|)?? \"
      "\\(?::[ \\t]+\\|\\(:[-[:alnum:]]_@#%:[+:(\\|\\|)\\|)\"?\"
      "[ \\t]*$)\"))
```

```
(use-package org :hook (org-mode . sachas-org-add-dashes-to-tag-regexps))
```

8.44 Convert from Markdown

ChatGPT likes to output Markdown. I like to think in Org Mode.

```
;;###autoload
(defun sach-org-convert-region-from-markdown (beg end)
  (interactive "r")
  (shell-command-on-region beg end "pandoc -t org" nil t))
```

8.45 Copying information from my phone

I have a tiny Tasker script that makes it easy to log timestamped entries as files in a directory that I synchronize with Dropbox. This code pulls that information into my ~/Dropbox/tasker/

```
;;###autoload
(defun sacha-read-phone-entries ()
  "Copy phone data to a summary Org file."
  (interactive)
  (mapc
   (lambda (filename)
     (let ((base (file-name-base filename)) contents timestamp category encoded-time date)
       (when (string-match "^[^ ]+ [^ ]+ \\([^\ ]+\\) - \\(.*\\)" base)
         (setq time (seconds-to-time (/ (string-to-number (match-string 1 base)) 1000))
               encoded-time (decode-time time)
               date (list (elt encoded-time 4) (elt encoded-time 3) (elt encoded-time 5))
               category (match-string 2 base))
         (with-temp-buffer
           (insert-file-contents filename)
           (setq contents (s-trim (buffer-string))))
         (with-current-buffer
          (find-file "~/dropbox/tasker/summary.txt")
          (org-datetree-find-date-create date)
          (unless (save-excursion (re-search-forward (regexp-quote base) nil t))
            (goto-char (line-end-position))
            (insert "\n")
            (insert "**** " contents " :" category ":\n" base "\n")
            (insert (format-time-string "[%Y-%m-%d %a %H:%M]\n" time)))

          (if (member category '("Think" "Do"))
              (save-excursion
                (org-back-to-heading t)
                (if (looking-at org-outline-regexp) (goto-char (1- (match-end 0))))
                (unless (looking-at org-todo-regexp)
                  (org-todo "TODO"))))
              (if (string-match "^Energy \\([0-9]\\)" contents)
                  (org-set-property "ENERGY" (match-string 1 contents))))
          (delete-file filename))))
     (directory-files "~/dropbox/tasker/data" t "\\..txt$")))
```

8.46 Emacs packages, other settings for easy Emacs News generation

8.46.1 ASCII export

This setting puts Org ASCII export links right after the text instead of in a separate section:

```
(setq org-ascii-links-to-notes nil)
```

8.46.2 Reddit

This one exports links from my secret `sacha-reddit-upvoted-json`. You can get your Reddit upvoted JSON URL at <https://www.reddit.com/prefs/feeds/>.

```
;;###autoload
(defun sacha-reddit-list-upvoted (date)
  (interactive (list (org-read-date))))
```

```

(let ((threshold (org-read-date nil t (concat (substring date 0 (min (length date) 10)) " ←
↳ 0:00"))))
  (url sacharreddit-upvoted-json)
  results)
(while url
  (with-current-buffer (url-retrieve-synchronously url)
    (goto-char (point-min))
    (re-search-forward "$")
    (let* ((data (json-read))
           (items (assoc-default 'children (assoc-default 'data data)))
           (after (assoc-default 'after (assoc-default 'data data)))
           (result
            (mapconcat
             (lambda (item)
               (let* ((o (assoc-default 'data item))
                      (title (assoc-default 'title o))
                      (url (helm-html-decode-entities-string (assoc-default 'url o)))
                      (date (seconds-to-time (assoc-default 'created_utc o)))
                      (permalink (concat "https://reddit.com" (assoc-default 'permalink o)))
                      (num-comments (assoc-default 'num_comments o 'eq 0)))
                 (when (time-less-p threshold date)
                   (if (and (> num-comments 0) (not (string-match "reddit\\.com" url)))
                     (format "- %s (%s)\n"
                              (org-link-make-string (url-unhex-string url) title)
                              (org-link-make-string (url-unhex-string permalink) "Reddit"))
                     (format "- %s\n" (org-link-make-string (url-unhex-string url) ←
↳ title))))))
              items "")))
    (setq results (concat result "\n" results))
    (setq url
      (if (and after (> (length result) 0))
        (concat sacharreddit-upvoted-json "&after=" after)
        nil))))
results))
;; (sacharreddit-list-upvoted "-mon")

```

Trying out reddigg:

```
(use-package reddigg :vc (:url "https://github.com/thanhvg/emacs-reddigg") :commands reddigg)
```

8.46.3 Sorting Org Mode lists using a sequence of regular expressions

emacs:org

I manually categorize Emacs News links into an Org unordered list, and then I reorganize the list by using M-S-up (org-shiftmetaup) and M-S-down (org-shiftmetadown). I decide to combine or split categories depending on the number of links. I have a pretty consistent order. John Wiegley suggested promoting Emacs Lisp and Emacs development links at the top of the list. I like to sort the rest of the list roughly by interest: general links first, then Org, then coding, then other links at the bottom.

Here's some code that sorts Org lists in a custom sequence, with unknown items at the bottom for easy re-ordering. It will take a list like:

- Other:
 - Link A
 - Link B
- Emacs development:

- Link A
- Link B
- Emacs Lisp:
 - Link A
 - Link B

and turn it into:

- Emacs Lisp:
 - Link A
 - Link B
- Emacs development:
 - Link A
 - Link B
- Other:
 - Link A
 - Link B

```
;;###autoload
(defun sach-org-sort-list-in-custom-order (order)
  "Sort the current Org list so that items are in the specified order.
  ORDER is a list of regexps."
  (org-sort-list
   nil ?f
   (lambda ()
     (let ((case-fold-search t)
           (item
            (when (looking-at "[ \t]*[-+*0-9.])+\\([ \t]+\\([- X]\\)\\)\\)?[ \t]+")
              (org-sort-remove-invisible (buffer-substring (match-end 0) (point-at-eol))))))
      (or (cl-position item order :test (lambda (a b) (string-match b a))) (1+ (length order))))
     order))))
  '<))
```

8.46.4 Package links

```
;;###autoload
(defun sach-org-package-open (package-name)
  (interactive "MPackage name: ")
  (describe-package (intern package-name)))

(ert-deftest sach-org-package-export ()
  (should
   (string=
    (sach-org-package-export "transcribe" "transcribe" 'html)
    "<a target=\"_blank\" href=\"https://elpa.gnu.org/packages/transcribe.html\">transcribe</a>")
   ))
  (should
   (string=
    (sach-org-package-export "fireplace" "fireplace" 'html)
    "<a target=\"_blank\" href=\"http://melpa.org/#/fireplace\">fireplace</a>")
   ))
  )

;;###autoload
(defun sach-org-package-export (link description format &optional arg)
  (let* ((package-info (car (assoc-default (intern link) package-archive-contents)))
         (package-source (and package-info (package-desc-archive package-info)))
         (path (format
                 (cond
```

```

((null package-source) link)
((string= package-source "gnu") "https://elpa.gnu.org/packages/%s.html")
((string= package-source "melpa") "https://melpa.org/#/%s")
((string= package-source "nongnu") "https://elpa.nongnu.org/nongnu/%s.html")
(t (error 'unknown-source)))
link))
(desc (or description link)))
(if package-source
  (cond
    ((eq format 'lily) (format "<a target=\"_blank\" ↵
↵ href=\"%s\">%s</a>" path desc))
    ((eq format 'html) (format "<a target=\"_blank\" ↵
↵ href=\"%s\">%s</a>" path desc))
    ((eq format 'wp) (format "<a target=\"_blank\" ↵
↵ href=\"%s\">%s</a>" path desc))
    ((eq format 'latex) (format "\\href{%s}{%s}" path desc))
    ((eq format 'texinfo) (format "@uref{%s,%s}" path desc))
    ((eq format 'ascii) (format "%s <%s>" desc path))
    ((eq format 'org) (org-link-make-string (concat "package:" ↵
↵ link) description)))
  (t path))
  desc)))
;;;###autoload
(defun sacha-org-package-complete ()
  (require 'finder-inf nil t)
  (unless package--initialized
    (package-initialize t))
  (concat
    "package:"
    ;; Load the package list if necessary (but don't activate them).
    (let ((packages (mapcar #'symbol-name (mapcar #'car package-archive-contents)))))
      (completing-read "Package: "
        packages nil t nil nil))))
;;;###autoload
(defun sacha-org-package-link-description (link description)
  (unless description
    (when (string-match "package:\\(\\.+\\.\\)" link)
      (match-string 1 link))))

```

```

(with-eval-after-load 'org
  (org-link-set-parameters
    "package"
    :follow 'sacha-org-package-open :export 'sacha-org-package-export :complete ↵
    ↵ 'sacha-org-package-complete
    :insert-description #'sacha-org-package-link-description))

```

8.47 Save when Emacs loses focus

```

;;;###autoload
(defun sacha-org-save-all-org-buffers ()
  (unless sacha-unfocusing
    (let ((sacha-unfocusing t))
      (sacha-org-debounce-idle-timer 10
        sacha-org-save-all-org-buffers-timer
        'org-save-all-org-buffers))))

```

```
(use-package org
  :config
  (add-function :after after-focus-change-function 'sacha-org-save-all-org-buffers))
```

8.48 Clipboard

```
;;;###autoload
(defun sacha-org-insert-clipboard ()
  "Convert clipboard contents from HTML to Org and then paste (yank).\"
  (interactive)
  (insert (shell-command-to-string "xclip -o -selection clipboard -t text/html | pandoc -f ↵
    ↵ html -t json | pandoc -f json -t org"))))

;;;###autoload
(defun sacha-org-insert-clipboard-without-data-images ()
  "Convert clipboard contents from HTML to Org and then insert, but replace images with ↵
    ↵ placeholders\"
  (interactive)
  (insert
    (with-temp-buffer
      (shell-command "xclip -o -selection clipboard -t text/html" (current-buffer))
      (goto-char (point-min))
      (while (re-search-forward "<img src='data:image[^>]*?'" nil t)
        (replace-match "{{{ image }}}"))
      (buffer-string))))

;;;###autoload
(defun sacha-org-convert-clipboard-to-org-without-data-images ()
  "Convert clipboard contents from HTML to Org and then insert, but replace images with ↵
    ↵ placeholders\"
  (interactive)
  (kill-new
    (with-temp-buffer
      (shell-command "xclip -o -selection clipboard -t text/html" (current-buffer))
      (goto-char (point-min))
      (while (re-search-forward "<img src='data:image[^>]*?'" nil t)
        (replace-match "{{{ image }}}"))
      (goto-char (point-min))
      (while (re-search-forward "<\\(tr\\)>" nil t)
        (replace-match "{{{ image }}}"))
      (buffer-string))))
```

8.49 Setting properties

```
;;;###autoload
(defun sacha-org-set-property (property value)
  "In the current entry, set PROPERTY to VALUE.
Use the region if active.\"
  (interactive
    (list
      (org-read-property-name)
      (when (region-active-p)
        (replace-regexp-in-string
          "[ \\n\\t]+" " "
          (buffer-substring (point) (mark)))))))
```



```
(org-set-property property value))
```

```
(use-package org
  :bind (:map org-mode-map
    ("C-c C-x p" . sach-org-set-property)))
```

8.50 Linking to and exporting function definitions in Org Mode

emacs:org

- [2024-01-11 Thu]: Added ?link=1 to copy the context link
- 2023-09-12: added a way to force the defun to start open with ?open=1
- 2023-09-05: fixed the completion to include **defun**:

I'd like to write more blog posts about little Emacs hacks, and I'd like to do it with less effort. Including source code is handy even when it's missing some context from other functions defined in the same file, since sometimes people pick up ideas and having the source code right there means less flipping between links. When I'm working inside my config file or other literate programming documents, I can just write my blog post around the function definitions. When I'm talking about Emacs Lisp functions defined elsewhere, though, it's a little more annoying to copy the function definition and put it in a source block, especially if there are updates.

The following code creates a **defun** link type that exports the function definition. It works for functions that can be located with find-function, so only functions loaded from .el files, but that does what I need for now. Probably once I post this, someone will mention a much more elegant way to do things. Anyway, it makes it easier to use **org-store-link** to capture a link to the function, insert it into a blog post, navigate back to the function, and export HTML.

```
;;;###autoload
(defun sach-org-defun-complete ()
  "Return function definitions."
  (concat "defun:"
          (completing-read
            "Function: "
            #'help--symbol-completion-table
            #'fboundp
            'confirm
            nil nil))) ; (and fn (symbol-name fn)) ?

;;;###autoload
(defun sach-org-defun-link-description (link description)
  "Add documentation string as part of the description"
  (unless description
    (when (string-match "defun:\\\(.+\\)" link)
      (let ((symbol (intern (match-string 1 link))))
        (when (documentation symbol)
          (concat (symbol-name symbol) ": "
                  (car (split-string ↵
                                ↵ (documentation symbol) "\n"))))))))

;;;###autoload
(defun sach-org-defun-open-complete ()
  "Return function definitions."
  (concat "defun-open:"
          (completing-read
            "Function: "
```

```

                                #'help--symbol-completion-table
                                #'fboundp
                                'confirm
                                nil nil)))

;;;###autoload
(defun sacha-org-defun-open-export (link description format _)
  (sacha-org-defun-export (concat link (if (string-match "\\?" link) "&open=1" ←
    ↪ "?open=1")) description format _))

;;;###autoload
(defun sacha-org-defun-export (link description format _)
  "Export the function."
  (let (symbol params path-and-query
        sacha-org-defun-open-special)
    (if (string-match "\\?" link)
        (setq path-and-query (url-path-and-query ←
    ↪ (url-generic-parse-url link))
                                symbol (car path-and-query)
                                params (url-parse-query-string (cdr ←
    ↪ path-and-query)))
        (setq symbol link))
    (save-window-excursion
      (sacha-org-defun-open symbol t)
      (let ((function-body
    ↪ sexp at point
        (if (derived-mode-p 'org-mode)
            (nth 2 (org-src--contents-area ←
    ↪ (org-element-at-point)))
            (thing-at-point 'defun)))
          body)
        (pcase format
          ((or '11ty 'html)
           (setq body
    ↪ (if (assoc-default "bare" ←
    ↪ (format "<div ←
    ↪ class=\"org-src-container\"><pre class=\"src src-emacs-lisp\">%s</pre></div>"
    ↪
    ↪ (org-html-do-format-code function-body "emacs-lisp" nil nil ←
    ↪ nil nil))
           (format ←
    ↪ "<details%s><summary>%s</summary><div class=\"org-src-container\"><pre class=\"src ←
    ↪ src-emacs-lisp\">%s</pre></div></details>"
    ↪
    ↪ (if (assoc-default "open" params 'string=) " open"
    ↪
    ↪ "")
    ↪
    ↪ (or description
    ↪
    ↪ (and (documentation (intern symbol))
    ↪
    ↪ (concat
    ↪
    ↪ symbol
    ↪
    ↪ ": "

```

```

↪                                     (car (split-string ↪
↪ (documentation (intern symbol)) "\n"))))
↪
↪                                     symbol)
↪
↪                                     (org-html-do-format-code function-body "emacs-lisp" nil nil nil nil)))
↪                                     (when (assoc-default "link" params)
↪                                     (setq body (format "%s<div><a ↪
↪ href=\"%s\">Context</a></div>" body (sacha-copy-link))))
↪                                     body)
↪                                     ('latex
↪                                     (org-latex-src-block `(test
↪
↪                                     (:language "emacs-lisp" :value ↪
↪ ,function-body))
↪
↪                                     function-body nil))
↪                                     ('ascii function-body)
↪                                     (_ function-body))))))

;;;###autoload
(defun sacha-org-defun-store ()
  "Store a link to the function."
  (when (derived-mode-p 'emacs-lisp-mode)
    (org-link-store-props :type "defun"

↪                                     :link (concat "defun:" (lisp-current-defun-name)))))

(defvar sacha-org-defun-open-special t)

;;;###autoload
(defun sacha-org-defun-open (symbol &rest _)
  "Jump to the function definition.
If it's from a tangled file, follow the link."
  (interactive (list (symbol-at-point)))
  (when (symbolp symbol) (setq symbol (symbol-name symbol)))
  (cond
   ((fboundp (intern symbol)) (find-function (intern symbol)))
   ((boundp (intern symbol)) (find-variable (intern symbol)))
   (when (re-search-backward "^;; \\[\\[file:" nil t)
     (goto-char (match-end 0))
     (org-open-at-point-global)
     (when (re-search-forward (concat "( *\\(cl-\\)?defun +" (regexp-quote ↪
↪ (replace-regexp-in-string "\\?.*$" "" symbol)))
↪
↪                                     nil t)
↪                                     (goto-char (match-beginning 0))
    (when (and sacha-org-defun-open-special
              (org-in-src-block-p))
      (org-edit-special)
      (goto-char (point-min))
      (when (re-search-forward (concat "( *\\(cl-\\)?defun +" (regexp-quote ↪
↪ (replace-regexp-in-string "\\?.*$" "" symbol)))
↪
↪                                     nil t)
↪                                     (goto-char (match-beginning 0))))))

(org-link-set-parameters "defun" :follow #'sacha-org-defun-open

```

```

↪      :export #'sacha-org-defun-export
↪      :complete #'sacha-org-defun-complete
↪      :insert-description #'sacha-org-defun-link-description
↪      :store #'sacha-org-def-store)
(org-link-set-parameters "defun-open" :follow #'sacha-org-defun-open
↪
↪      :export #'sacha-org-defun-open-export
↪      :complete #'sacha-org-defun-open-complete
↪      :insert-description #'sacha-org-defun-link-description)

```

sacha-copy-link is at [web-link](#) (p. 334).

8.50.1 Still allow linking to the file

Sometimes I want to link to a defun and sometimes I want to link to the file itself. Maybe I can have a file link with the same kind of scoping so that it kicks in only when `defun:` would also kick in.

```

;;;###autoload
(defun sacha-org-defun-store-file-link ()
  "Store a link to the file itself."
  (when (derived-mode-p 'emacs-lisp-mode)
    (org-link-store-props :type "file"
↪      :link (concat "file:" (buffer-file-name)))))

```

```

(with-eval-after-load 'org
  (org-link-set-parameters "_file" :store #'sacha-org-defun-store-file-link))

```

8.51 Including variables

[2024-01-20 Sat]: Fixed org-def-store thanks to oantolin's comment.

```

;;;###autoload
(defun sacha-org-defvar-complete ()
  "Return variable definitions."
  (concat "defvar:"
↪      (completing-read
        "Variable: "
        #'help--symbol-completion-table
        #'indirect-variable
        'confirm
        nil nil))) ; (and fn (symbol-name fn)) ?

;;;###autoload
(defun sacha-org-defvar-link-description (link description)
  "Add documentation string as part of the description"
  (unless description

```



```

                                (`ascii body)
                                (_ body))))))

;;;###autoload
(defun sacha-org-def-store ()
  "Store a link to the function."
  (when (derived-mode-p 'emacs-lisp-mode)
    (save-excursion
      (or (eobp) (forward-char 1))
      (condition-case nil
        (progn
          (beginning-of-defun)
          (let ((data (read (current-buffer))))
            (cond
              ((not (listp data)) nil)
              ((eq (car data) 'defun)
               (org-link-store-props :type "defun"
                                     :link (concat "defun:" ↵
                                     ↵ (lisp-current-defun-name))))
              ((member (car data) '(defvar defcustom))
               (org-link-store-props :type "defvar"
                                     :link (format "defvar:%s" (cadr data))))))
          (error nil))))))

;;;###autoload
(defun sacha-org-defvar-open (symbol _)
  "Jump to the function definition."
  (find-variable (intern (replace-regexp-in-string "\\?.*$" "" symbol))))

(org-link-set-parameters "defvar" :follow #'sacha-org-defvar-open
  ↵ :export #'sacha-org-def-export ↵
  ↵ :complete #'sacha-org-defvar-complete ↵
  ↵ :insert-description #'sacha-org-defvar-link-description ↵
  ↵ ; :store #'sacha-org-def-store ; already added by defun link ↵
  ↵ )

```

8.52 Org - send things to the bottom of the list

Handy for collecting items together.

```

;;;###autoload
(defun sacha-org-send-to-bottom-of-list ()
  "Send the current line to the bottom of the list."
  (interactive)
  (beginning-of-line)
  (let ((kill-whole-line t))
    (save-excursion
      (kill-line 1)
      (org-end-of-item-list)
      (yank))))

```

8.53 Org Mode: Format Libby book highlights exported as JSON

The [Toronto Public Library](#) (and many other libraries) offers e-book access through Overdrive, which I can read through the Libby app on my phone. It turns out that I can select passages to highlight. It also turns out that I can use the Reading Journey view to [export the highlights as JSON](#), even for books I've returned. This is what the JSON looks like.

```
{
  "version": 1,
  "readingJourney": {
    "cover": {
      "contentType": "image/jpeg",
      "url": ↵
      ↵ "https://img1.od-cdn.com/ImageType-100/7635-1/{B41A3269-BC2A-4497-8C71-0A3F1FA3C694}Img100.jpg",
      "title": "How to Take Smart Notes",
      "color": "#D9D9D9",
      "format": "ebook"
    },
    "title": {
      "text": "How to Take Smart Notes",
      "url": "https://share.libbyapp.com/title/5796521",
      "titleId": "5796521"
    },
    "author": "Sönke Ahrens",
    "publisher": "Sönke Ahrens",
    "isbn": "9781393776819",
    "percent": 0.313229455252918
  },
  "highlights": [
    {
      "timestamp": 1729898852000,
      "chapter": "13      Share Your Insight",
      "percent": 0.824912451361868,
      "color": "#FFB",
      "quote": "For every document I write, I have another called “xy-rest.doc”, and every ↵
      ↵ single time I cut something, I copy it into the other document, convincing myself that ↵
      ↵ I will later look through it and add it back where it might fit. Of"
    },
    {
      "timestamp": 1729898760000,
      "chapter": "13      Share Your Insight",
      "percent": 0.801566108949416,
      "color": "#FFB",
      "quote": "I always work on different manuscripts at the same time. With this method, to ↵
      ↵ work on different things simultaneously, I never encounter any mental blockages"
    },
    ...
  ]
}
```

I want to save those highlights in my `books.org` file for easy searching, grouping the highlights by chapter. The following code helps with that:

```
;; ##autoload
(defun sachsa-org-insert-book-highlights-from-libby (url)
  (interactive "MURL: ")
  (let-alist (plz 'get url :as #'json-read)
    (insert
```

```

    "*"
    .readingJourney.title.text
    "- "
    .readingJourney.author
    "\n")
(org-set-property "ISBN" .readingJourney.isbn)
(org-set-property "COVER" .readingJourney.cover.url)
(org-set-property "TITLE" .readingJourney.title.text)
(org-set-property "AUTHOR" .readingJourney.author)
(insert (org-link-make-string .readingJourney.title.url ↵
↵ .readingJourney.cover.url)
        "\n")
;; sort the highlights by chapter
(insert
  (mapconcat
    (lambda (row)
      (concat "** " (replace-regexp-in-string " + " " (car row)) "\n"
        (mapconcat (lambda (quote)
          ↵
          (concat "#+begin_quote\n"
            ↵
            (alist-get 'quote quote)
            ↵
            "\n#+end_quote\n\n"))
          ↵
          (cdr row)
          ↵
          ""))
        "\n\n"))
    (seq-group-by
      (lambda (o) (alist-get 'chapter o))
      (sort .highlights
        :key (lambda (o) (alist-get 'percent o))))))

```

This is what the resulting document looks like:

```

* How to Take Smart Notes - Sönke Ahrens
:PROPERTIES:
:ISBN:      9781393776819
:COVER:     ↵
            ↵ https://img1.od-cdn.com/ImageType-100/7635-1/{B41A3269-BC2A-4497-8C71-0A3F1FA3C694}Img100.jpg
:TITLE:     How to Take Smart Notes
:AUTHOR:    Sönke Ahrens
:END:
[[https://share.libbyapp.com/title/5796521][https://img1.od-cdn.com/ImageType-100/7635-1/{B41A3269-BC2A-4497-8C71-
** 1 Everything You Need to Know
#+begin_quote
  never force myself to do anything I 'dont feel like. Whenever I am stuck, I do something else".
#+end_quote

#+begin_quote'
Luhmanns only real help was a housekeeper who cooked for him and his children during the week, ↵
  ↵ not that extraordinary considering he had to raise three children on his own after his ↵
  ↵ wife died early.
#+end_quote

...

```


8.54 STARTED Validation

```
(defvar sach-org-validate-functions
  '(sach-org-validate-no-blank-titles
    sach-org-validate-unique-outline-paths
    sach-org-validate-no-syncthing-conflicts))
;;;###autoload
(defun sach-org-validate ()
  (interactive)
  (unless (string-match "_archive\\'" (buffer-file-name))
    (run-hooks 'sach-org-validate-functions)))
```

8.54.1 Keep only unique headings

```
;;;###autoload
(defun sach-compore-org-headings (file)
  (interactive "FOther file: ")
  (let ((current (org-map-entries (lambda () (org-entry-get (point) "ITEM")) "LEVEL=1" ↵
    ↵ 'file)))
    (find-file file)
    (goto-char
     (catch 'done
      (org-map-entries
       (lambda ()
         (when (member (org-entry-get (point) "ITEM") current)
           (throw 'done (point))))
       "LEVEL=1" 'file)))))
```

8.54.2 TODO No blank titles, no duplicate paths

```
;;;###autoload
(defun sach-org-validate-no-blank-titles ()
  (interactive)
  (let ((point (point)))
    (goto-char (point-min))
    (while (re-search-forward org-heading-regexp nil t)
      (unless (match-string 2)
        (error "Empty title")))
    (goto-char point)))

;;;###autoload
(defun sach-org-validate-unique-outline-paths ()
  (interactive)
  (let ((point (point)))
    (goto-char (point-min))
    (setq org-outline-path-cache nil)
    (let* ((paths (make-hash-table :test 'equal))
           (found (catch 'found
                     (org-map-entries
                      (lambda ()
                        (let ↵
                          ↵
                            (path (string-join (org-get-outline-path t) "/"))
                            ↵
                            (if (gethash path paths)
```

```

↩→                                     ↩→
      (throw 'found (cons (point) path))
↩→                                     ↩→
      (puthash path (point) paths))))
                                     nil)))
      (if found
          (progn
              (goto-char (car found))
              (error "Duplicate found: %s - previous %d" ↩→
                  (cdr found) (gethash (cdr found) paths))
              found)
          (goto-char point)
          (when (called-interactively-p 'any) (message "No ↩→
      ↩→ duplicates"))))))

;;;###autoload
(defun sacha-org-delete-duplicate-outline-paths-interactively ()
  (interactive)
  (let ((point (point))
        previous)
    (goto-char (point-min))
    (org-map-entries
     (lambda ()
       (let ((path (string-join (org-get-outline-path t t) "/")))
         (when (assoc-default path previous #'string=)
           (when (y-or-n-p "Delete this possible duplicate? ")
             (org-cut-subtree)))
         (push (cons path (point)) previous))))))

;;;###autoload
(defun sacha-org-validate-no-syncthing-conflicts ()
  (when (directory-files default-directory nil "sync-conflict.*\\.org")
    (message "Syncthing conflicts exist.")))

```

8.55 Sorting completion candidates, such as sorting Org headings by level emacs.org

[2026-02-26 Thu]: Made the code even neater with `:key`, included the old code as well

At this week's [Emacs Berlin meetup](#), someone wanted to know how to change the order of completion candidates. Specifically, they wanted to list the top level Org Mode headings before the second level headings and so on. They were using `org-ql` to navigate Org headings, but since `org-ql` sorts its candidates by the number of matches according to the code in the `org-ql-completing-read` function, I wasn't quite sure how to get it to do what they wanted. (And I realized my `org-ql` setup was broken, so I couldn't fiddle with it live. Edit: Turns out I needed to update the `peg` package) Instead, I showed folks `consult-org-heading` which is part of the [Consult](#) package, which I like to use to jump around the headings in a single Org file. It's a short function that's easy to use as a starting point for something custom.

Here's some code that allows you to use `consult-org-heading` to jump to an Org heading in the current file with completions sorted by level.

```

(with-eval-after-load 'consult-org
  (advice-add
   #'consult-org--headings

```

```

:filter-return
(lambda (candidates)
  (sort candidates
    :key (lambda (o) (car (get-text-property 0 'consult-org--heading o))))))

```

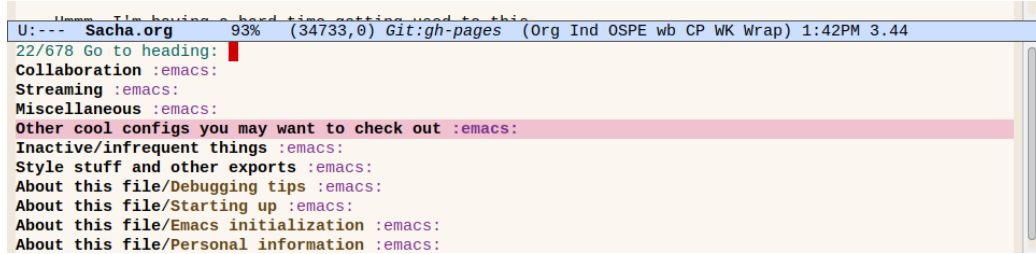


Figure 13: Screenshot showing where the candidates transition from top-level headings to second-level headings

My previous approach defined a different function based on `consult-org-heading`, but using the advice feels a little cleaner because it will also make it work for any other function that uses `consult-org--headings`. I've included the old code in case you're curious. Here, we don't modify the function's behaviour using advice, we just make a new function (`sacha-consult-org-heading`) that calls another function that processes the results a little (`sacha-consult-org--headings`).

```

(defun sacha-consult-org--headings (prefix match scope &rest skip)
  (let ((candidates (consult-org--headings prefix match scope)))
    (sort candidates
      :lessp
      (lambda (a b)
        (let ((level-a (car (get-text-property 0 'consult-org--heading a)))
              (level-b (car (get-text-property 0 'consult-org--heading b))))
          (cond
            ((< level-a level-b) t)
            ((< level-b level-a) nil)
            ((string< a b) t)
            ((string< b a) nil)))))))

(defun sacha-consult-org-heading (&optional match scope)
  "Jump to an Org heading.

MATCH and SCOPE are as in `org-map-entries' and determine which
entries are offered. By default, all entries of the current
buffer are offered."
  (interactive (unless (derived-mode-p #'org-mode)
    (user-error "Must be called from an Org buffer")))
  (let ((prefix (not (memq scope '(nil tree region start-level file)))))
    (consult--read
      (consult--slow-operation "Collecting headings ... "
        (or (sacha-consult-org--headings prefix match scope)
          (user-error "No headings"))))
    :prompt "Go to heading: "
    :category 'org-heading
    :sort nil
    :require-match t
    :history '(:input consult-org--history)
    :narrow (consult-org--narrow)
    :state (consult--jump-state)
    :annotate #'consult-org--annotate)

```

```
:group (and prefix #'consult-org--group)
:lookup (apply-partially #'consult--lookup-prop 'org-marker))))
```

I also wanted to get this to work for `C-u org-refile`, which uses `org-refile-get-location`. This is a little trickier because the table of completion candidates is a list of cons cells that don't store the level, and it doesn't pass the metadata to `completing-read` to tell it not to re-sort the results. We'll just fake it by counting the number of `/`, which is the path separator used if `org-outline-path-complete-in-steps` is set to `nil`.

```
(with-eval-after-load 'org
  (advice-add
    'org-refile-get-location
    :around
    (lambda (fn &rest args)
      (let ((completion-extra-properties
            '(:display-sort-function
              (lambda (candidates)
                (sort candidates
                  :key (lambda (s) (length (split-string s "/"))))))))
        (apply fn args)))))
```

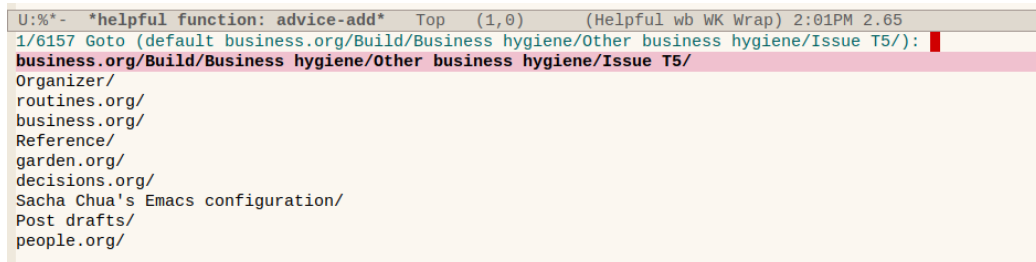


Figure 14: Screenshot of sorted refile entries

In general, if you would like completion candidates to be in a certain order, you can specify `display-sort-function` either by calling `completing-read` with a collection that's a lambda function instead of a table of completion candidates, or by overriding it with `completion-category-overrides` if there's a category you can use or `completion-extra-properties` if not.

Here's a short example of passing a lambda to a completion function ([thanks to Manuel Uberti](#)):

```
(defun mu-date-at-point (date)
  "Insert current DATE at point via `completing-read'."
  (interactive
    (let* ((formats '("%Y%m%d" "%F" "%Y%m%d%H%M" "%Y-%m-%dT%T"))
          (vals (mapcar #'format-time-string formats))
          (opts
            (lambda (string pred action)
              (if (eq action 'metadata)
                  '(metadata (display-sort-function . identity))
                  (complete-with-action action vals string pred)))))
      (list (completing-read "Insert date: " opts nil t))))
  (insert date))
```

If you use `consult--read` from the [Consult](#) completion framework, there is a `:sort` property that you can set to either `nil` or your own function.

This entry is part of the [Emacs Carnival for Feb 2026: Completion](#).

8.56 Vector search

<https://github.com/jkitchin/org-db-v3>

John Kitchin just posted a different project - org-db-v3. Let's see if I can figure that out... [Fulltext, semantic text and image search in Emacs - YouTube](#)

```
(use-package org-db-v3
  :load-path "~/vendor/org-db-v3/elisp"
  :init
  (setq org-db-v3-auto-enable nil))
```

```
;; ##autoload
(defun sacha-org-db-v3-to-emacs-rag-search (query &optional limit filename-pattern)
  "Search org-db-v3 and transform the data to look like emacs-rag-search's output."
  (org-db-v3-ensure-server)
  (setq limit (or limit 100))
  (mapcar (lambda (o)
    `((source_path . ,(assoc-default 'filename o))
      (line_number . ,(assoc-default 'begin_line o))
      ,@o))
    (sort
     (assoc-default 'results
                    (plz 'post (concat (org-db-v3-server-url) "/api/search/semantic")
                          :headers '(("Content-Type" . "application/json"))
                          :body (json-encode `((query . ,query)
                                                (limit . ,limit)
                                                (filename_pattern . ,filename-pattern)))
                          :as #'json-read))
     (lambda (o) (alist-get 'similarity_score o))
     :reverse t)))
```

8.56.1 Indexing-related code

Sample code for indexing my main files:

```
(dolist (file '("~/sync/orgzly/organizer.org"
  "~/sync/orgzly/posts.org"
  "~/sync/orgzly/reading.org"
  "~/sync/orgzly/resources.org"))
  (org-db-v3-index-file-async file))
```

Sample code for indexing the Org copies of my blog posts:

```
(mapc #'org-db-v3-index-file-async
      (sacha-blog-org-files-except-reviews))
```

Fixing the index:

```
;; Based on org-db-v3-reindex-database
;; ##autoload
(defun sacha-org-db-v3-remove-missing-files ()
  "Remove missing files currently in the database.
Fetches the list of files from the server and reindexes each one.
Also removes files that no longer exist from the database.
Uses non-blocking queue processing to keep Emacs responsive."
```

```

Skips remote Tramp files."
(interactive)
(org-db-v3-ensure-server)
(plz 'get (concat (org-db-v3-server-url) "/api/files")
:as #'json-read
:then (lambda (response)
  (let* ((files (alist-get 'files response))
        (count (length files))
        (missing-files nil)
        (existing-files nil)
        (remote-files nil))

    ;; Classify files as existing, missing, or remote
    (dotimes (i count)
      (let ((filename (alist-get 'filename (elt files i))))
        (cond
         ((file-remote-p filename)
          (push filename remote-files))
         ((file-exists-p filename)
          (push filename existing-files))
         (t
          (push filename missing-files))))))
    (if (zerop count)
        (message "No files found in database")
        (when missing-files
          (message "Removing %d missing file%s from database ..."
                   (length missing-files)
                   (if (= (length missing-files) 1) "" "s"))
          (dolist (filename missing-files)
            (org-db-v3-delete-file-async filename))))
    )))
:else (lambda (error)
  (message "Error fetching file list: %s" (plz-error-message error))))

```

8.56.2 Consult-based interface for searching blog posts

```

;;;###autoload
(defun sacha-blog-similar-link (link)
  "Vector-search blog posts using `emacs-rag-search' and insert a link.
If called with \\[universal-argument\\], use the current post's text.
If a region is selected, use that as the default QUERY.
HIDE-INITIAL means hide the initial query, which is handy if the query is very long."
  (interactive (list
    (if embark--command
        (read-string "Link: ")
        (sacha-blog-similar
         (cond
          (current-prefix-arg (sacha-11ty-post-text))
          ((region-active-p)
           (buffer-substring (region-beginning)
                             (region-end))))
         current-prefix-arg))))
  (sacha-embark-blog-insert-link link))

;;;###autoload
(defun sacha-embark-blog--inject-target-url (&rest args)
  "Replace the completion text with the URL."

```

```

(delete-minibuffer-contents)
(insert (sacha-blog-url (get-text-property 0 'consult--candidate (plist-get args :target))))))

;;;###autoload
(defun sacha-11ty-interactive-context (use-post)
  "Returns (query hide-initial) for use in interactive arguments.
If USE-POST is non-nil, query is the current post text and hide-initial is t.
If the region is active, returns that as the query."
  (list (cond
        (embark--command (read-string "Input: "))
        (use-post (sacha-11ty-post-text))
        ((region-active-p)
         (buffer-substring (region-beginning)
                           (region-end))))
        use-post))

;;;###autoload
(defun sacha-blog-similar (&optional query hide-initial)
  "Vector-search blog posts using org-db-v3 and present results via Consult.
If called with \\[universal-argument\\], use the current post's text.
If a region is selected, use that as the default QUERY.
HIDE-INITIAL means hide the initial query, which is handy if the query is very long."
  (interactive (sacha-11ty-interactive-context current-prefix-arg))
  (consult--read
   (if hide-initial
       (sacha-org-db-v3-blog-post--collection query)
       (consult--dynamic-collection
        #'sacha-org-db-v3-blog-post--collection
        :min-input 3 :debounce 1))
   :lookup #'consult--lookup-cdr
   :prompt "Search blog posts (approx): "
   :category 'sacha-blog
   :sort nil
   :require-match t
   :state (sacha-blog-post--state)
   :initial (unless hide-initial query)))

(defvar sacha-blog-semantic-search-source 'org-db-v3)
;;;###autoload
(defun sacha-org-db-v3-blog-post--collection (input)
  "Perform the RAG search and format the results for Consult.
Returns a list of cons cells (DISPLAY-STRING . PLIST)."
  (let ((posts (sacha-blog-posts)))
    (mapcar (lambda (o)
              (sacha-blog-format-for-completion
               (append o
                       (sacha-blog-post-info-for-url (alist-get 'source_path o)
                                                           posts))))
            (seq-uniq
             (sacha-org-db-v3-to-emacs-rag-search input 100 "%static-blog%")
             (lambda (a b) (string= (alist-get 'source_path a)
                                     (alist-get 'source_path b)))))))

```

```

(with-eval-after-load 'embark
  (add-to-list 'embark-target-injection-hooks '(sacha-blog-similar-link ↔
                                                ↪ sacha-embark-blog--inject-target-url)))

```

Index automatically smaller topic pages, but not the big one? exported blog posts

Handle sketches too

```
;; ##autoload
(defun sacha-org-db-v3-index-recent-sketches (after)
  (interactive (list
    (when current-prefix-arg
      (org-read-date nil nil nil "After: " nil "-2w")))))
  (setq after (or after (org-read-date nil nil "-2w")))
  (mapcar #'org-db-v3-index-file-async
    (seq-remove
      (lambda (o) (string> after (file-name-base o)))
      (directory-files "~/sync/sketches" t "\\..txt$"))))
```

```
;; ##autoload
(defun sacha-org-db-v3-sketch--collection (input)
  "Perform the RAG search and format the results for Consult.
Returns a list of cons cells (DISPLAY-STRING . PLIST)."
  (mapcar
    (lambda (o)
      (cons (file-name-base (alist-get 'source_path o)) o))
    (seq-uniq
      (sacha-org-db-v3-to-emacs-rag-search input 100 "%sync/sketches%")
      (lambda (a b) (string= (alist-get 'source_path a)
                              (alist-get 'source_path b))))))

;; ##autoload
(defun sacha-sketch-similar (&optional query hide-initial)
  "Vector-search blog posts using `emacs-rag-search' and present results via Consult.
If called with \\[universal-argument\\], use the current post's text.
If a region is selected, use that as the default QUERY.
HIDE-INITIAL means hide the initial query, which is handy if the query is very long."
  (interactive (sacha-11ty-interactive-context current-prefix-arg))
  (consult--read
    (if hide-initial
      (sacha-org-db-v3-sketch--collection query)
      (consult--dynamic-collection
        #'sacha-org-db-v3-sketch--collection
        :min-input 3 :debounce 1))
    :lookup #'consult--lookup-cdr
    :prompt "Search sketches (approx): "
    :category 'sketch
    :sort nil
    :require-match t
    :state (sacha-image--state)
    :initial (unless hide-initial query)))

;; ##autoload
(defun sacha-sketch-similar-insert (link)
  "Vector-search sketches and insert a link.
If called with \\[universal-argument\\], use the current post's text.
If a region is selected, use that as the default QUERY.
HIDE-INITIAL means hide the initial query, which is handy if the query is very long."
  (interactive (list
    (if embark--command
      (read-string "Sketch: ")
      (apply #'sacha-sketch-similar
        (sacha-11ty-interactive-context current-prefix-arg)))))
```



```

(sacha-insert-sketch-and-text link))

;;;###autoload
(defun sacha-sketch-similar-link (link)
  "Vector-search sketches and insert a link.
If called with \\[universal-argument\\], use the current post's text.
If a region is selected, use that as the default QUERY.
HIDE-INITIAL means hide the initial query, which is handy if the query is very long."
  (interactive (list
    (if embark--command
      (read-string "Sketch: ")
      (apply #'sacha-sketch-similar
        (sacha-11ty-interactive-context current-prefix-arg))))))
  (when (and (listp link) (alist-get 'source_path link))
    (setq link (sacha-image-filename (file-name-base link))))
  (insert (org-link-make-string (concat "sketchLink:" link) (file-name-base link))))

```

Multiple sources

```

(with-eval-after-load 'consult
  (defvar sacha-consult-source-similar-blog-posts
    (list :name "Blog posts"
      :narrow ?b
      :category 'sacha-blog
      :state #'sacha-blog-post--state
      :async (consult--dynamic-collection
        (lambda (input)
          (seq-take
            (sacha-org-db-v3-blog-post--collection input)
            5)))
      :action #'sacha-embark-blog-insert-link))

```

```

(defvar sacha-consult-source-similar-sketches
  (list :name "Sketches"
    :narrow ?s
    :category 'sketch
    :async (consult--dynamic-collection
      (lambda (input)
        (seq-take (sacha-org-db-v3-sketch--collection input) 5)))
    :state #'sacha-image--state
    :action #'sacha-insert-sketch-and-text)))

;;;###autoload
(defun sacha-consult-similar (query hide-initial)
  (interactive (sacha-11ty-interactive-context current-prefix-arg))
  (require 'consult)
  (if hide-initial
    (let ((new-sources
      (list
        (append
          (copy-sequence sacha-consult-source-similar-blog-posts)
          (list :items (seq-take (sacha-org-db-v3-blog-post--collection query) 5)))
        (append
          (copy-sequence sacha-consult-source-similar-sketches)
          (list :items (seq-take (sacha-org-db-v3-sketch--collection query) 5))))))
      (dolist (source new-sources)
        (cl-remf source :async))
      (consult--multi new-sources))

```

```

    (consult--multi '(sacha-consult-source-similar-blog-posts
                      sacha-consult-source-similar-sketches)
                    :initial query)))

;;;###autoload
(defun sacha-org-db-v3-index-recent-public (after)
  (interactive (list
                (when current-prefix-arg
                  (org-read-date nil nil nil "After: " nil "-2w")))))
  (setq after (or after (org-read-date nil nil "-2w")))
  (mapc #'org-db-v3-index-file-async
        (sacha-blog-org-files-except-reviews after))
  (sacha-org-db-v3-index-recent-sketches after))

```

8.56.3 emacs-rag-search?

Emacs RAG with LibSQL - Enabling semantic search of org-mode headings with Claude Code - YouTube.

```

(use-package emacs-rag
  :load-path "~/vendor/emacs-rag-libsql/emacs-rag"
  :commands (emacs-rag-menu emacs-rag--request)
  :config
  (setq emacs-rag-server-working-directory "~/vendor/emacs-rag-libsql/emacs-rag-server")
  (setq emacs-rag-server-command '("~/local/bin/uv" "run" "emacs-rag-server" "serve"))
  (setq emacs-rag-indexed-extensions '("org" "txt" "md"))
  (setq emacs-rag-auto-index-on-save nil))

```

```

;;;###autoload
(defun sacha-emacs-rag-search (query limit)
  (assoc-default 'results
    (emacs-rag--request "GET" "/search/vector" nil
      `((query . ,query)
        (limit . ,limit)
        (rerank . ,emacs-rag-search-enable-rerank)))))

;;;###autoload
(defun sacha-emacs-rag-search-blog-posts-hybrid (query &optional vector-weight limit)
  "Use hybrid search to search for QUERY.
Tune with VECTOR-WEIGHT
Start with LIMIT results and keep only unique blog posts."
  (interactive (list (if (region-active-p)
                        (buffer-substring (region-beginning)
                                           (region-end))
                        (read-string "Search: "))))
  (setq limit (or limit 100))
  (setq vector-weight (or vector-weight 0.5))
  (let* ((blog-posts (sacha-blog-posts))
         (results
          (seq-uniq
           (assoc-default 'results
                         (emacs-rag--request "GET" "/search/hybrid" nil
                           `((query . ,query)
                             (limit . ,limit)
                             (vector_weight . ,vector-weight)
                             (rerank . ,emacs-rag-search-enable-rerank)))))
         (lambda (a b)

```

```

      (string= (assoc-default 'source_path a nil "")
                (assoc-default 'source_path b nil ""))))))
    (seq-keep
      (lambda (o)
        (when (string-match "/blog/" (alist-get 'source_path o))
          (append
            o
            (sacha-blog-post-info-for-url (alist-get 'source_path o) blog-posts)
            nil)))
      results)))

```

9 Multimedia

```

(setq visible-bell t)
(use-package epwgraph :load-path "~/proj/epwgraph")

```

9.1 Emacs.tv

```

(use-package emacs.tv
  :load-path "~/proj/emacs.tv.github.io")

```

9.2 Timestamps

```

;;;###autoload
(defun sacha-filename-timestamp (file)
  (setq file (replace-regexp-in-string "^screen-" "" (file-name-base file)))
  (cond
    ((string-match
      ↵
      ↵ "^[0-9][0-9][0-9][0-9]\\_?[0-9][0-9]\\_?[0-9][0-9]\\_?[0-9][0-9]\\_?[0-9][0-9]\\_?[0-9][0-9]"
      file)
      (date-to-time (format "%s-%s-%s %s:%s:%s"
        ↵
        ↵ (match-string 1 file)
        ↵
        ↵ (match-string 2 file)
        ↵
        ↵ (match-string 3 file)
        ↵
        ↵ (match-string 4 file)
        ↵
        ↵ (match-string 5 file)
        ↵
        ↵ (match-string 6 file))))
      (t
        (time-add (date-to-time (format "%s %s" (substring file 0 10) (substring file ↵
        ↵ 11 19)))
          (float-time (/ (string-to-number ↵
        ↵ (substring file 20 23)) 1000.0))))))

(cl-assert
  (string= (format-time-string "test-%F-%T-%3N" (sacha-filename-timestamp ↵
    ↵ "screen-2024-09-20-13:18:08-024.png"))

```

```

                                "test-2024-09-20-13:18:08-024")
(string= (format-time-string "test-%F-%T-%3N" (sacha-filename-timestamp ←
↪ "screen-2024-09-20-13_1808-024.png")))
                                "test-2024-09-20-13:18:08-024"))

```

9.3 Images

:CUSTOM_ID: drawing

My sketch- and image-related code is scattered throughout my config. Here are some other headings:

- [images/Sacha.org](#)
- [images/Sacha.org](#)
- [images/Sacha.org](#)
- [images/Sacha.org](#)
- [images/Sacha.org](#)
- [images/Sacha.org](#)
- [images/Sacha.org](#)
- [images/Sacha.org](#)
- [images/Sacha.org](#)
- [images/Sacha.org](#)

9.3.1 Save edited text for sketch post

images:dotemacs

```

;;;###autoload
(defun sacha-org-sketch-open-text-file (sketch)
  (interactive (list (sacha-complete-sketch-filename)))
  (find-file (concat (file-name-sans-extension sketch) ".txt"))
  (with-current-buffer (find-file-noselect sketch)
    (display-buffer-in-side-window
     (current-buffer)
     '((window-width . 0.5)
       (side . right)))))

```

9.3.2 TODO Make header for Mastodon

drawing

9.3.3 Imagemagick

<https://xenodium.com/emacs-viewing-webp-images/>

```
(setq image-use-external-converter t)
```

Rotate clockwise or counterclockwise

```

;;;###autoload
(defun sacha-image-rotate-counterclockwise (image)
  (interactive "FImage: ")
  (call-process "mogrify" nil nil nil "-rotate" "270" image))
;;;###autoload
(defun sacha-image-rotate-clockwise (image)
  (interactive "FImage: ")

```

```
(call-process "mogrify" nil nil nil "-rotate" "90" image))
```

TOBLOG Emacs: Extract part of an image to another file It turns out that image-mode allows you to open an image and then crop it with `i c` (**i**mage-**c**rop) by drawing (or moving) a rectangle with the mouse, all within Emacs.

Instead of cropping the original image, I want to select a region and then write it to a different file. I think the ability to select a portion of an image by drawing/moving a rectangle is generally useful, so let's start by defining a function for that. The heavy lifting is done by `image-crop--crop-image-1`, which tracks the mouse and listens for events.

```
;; Based on image-crop.
;;;###autoload
(defun sach-image-select-rect (op)
  "Select a region of the current buffer's image.
OP should be a string describing the operation (ex: \"cut\").

`q': Exit without changing anything.
`RET': Select this region.
`m': Make mouse movements move the rectangle instead of altering the
rectangle shape.
`s': Same as `m', but make the rectangle into a square first."
  (unless (image-type-available-p 'svg)
    (error "SVG support is needed to crop and cut images"))
  (let ((image (image--get-image)))
    (unless (imagep image)
      (user-error "No image under point"))
    (when (overlays-at (point))
      (user-error "Can't edit images that have overlays"))
    ;; We replace the image under point with an SVG image that looks
    ;; just like that image. That allows us to draw lines over it.
    ;; At the end, we replace that SVG with a cropped version of the
    ;; original image.
    (let* ((data (cl-getf (cdr image) :data))
            (type (cond
                    ((cl-getf (cdr image) <-
                              :format)
                     (format "%s" (cl-getf <-
                                             (cdr image) :format)))
                    (data)
                     <-
                    (image-crop--content-type data))))
            (image-scaling-factor 1)
            (orig-point (point))
            (size (image-size image t))
            (svg (svg-create (car size) (cdr size)
                              <-
                              :xmlns:xlink "http://www.w3.org/1999/xlink"
                              <-
                              :stroke-width 5)))
      ;; We want to get the original text that's covered by the
      ;; image so that we can restore it.
      (image-start
       (save-excursion
        (let ((match (text-property-search-backward 'display image)))
          (if match
              (prop-match-end match)
```

```

        (point-min))))))
      (image-end
        (save-excursion
          (let ((match (text-property-search-forward 'display image)))
            (if match
              (prop-match-beginning match)
              (point-max))))))
        (text (buffer-substring image-start image-end))
        (inhibit-read-only t)
      orig-data svg-end)
    (with-temp-buffer
      (set-buffer-multibyte nil)
      (if (null data)
        (insert-file-contents-literally (cl-getf (cdr ←
↪ image) :file))
        (insert data))
      (let ((image-crop-exif-rotate nil))
        (image-crop--possibly-rotate-buffer image))
      (setq orig-data (buffer-string))
      (setq type (image-crop--content-type orig-data))
      (image-crop--process image-crop-resize-command
        `((?w . 600)
          (?f . ,(cadr (split-string type "/"))))))
      (setq data (buffer-string)))
    (svg-embed svg data type t
      :width (car size)
      :height (cdr size))
    (with-temp-buffer
      (svg-insert-image svg)
      (switch-to-buffer (current-buffer))
      (setq svg-end (point))
      ;; Area
      (let ((area
        (condition-case _
          (save-excursion
            (forward-line 1)
            ←
↪ (image-crop--crop-image-1
          (quit nil))))
          (when area
            ;; scale to original
            (let* ((image-scaling-factor 1)
              (osize (image-size ←
↪ (create-image orig-data nil t) t))
              (factor (/ (float ←
↪ (car osize)) (car size)))
              ;; width x height + ←
↪ left + top
              (width (abs (truncate ←
↪ (* factor (- (cl-getf area :right)
↪
↪
↪ (cl-getf area :left))))))
              (height (abs ←
↪ (truncate (* factor (- (cl-getf area :bottom)
↪
↪
↪ (cl-getf area :top))))))

```

```
(left (truncate (* ←
↪
↪
↪          (cl-getf area :right))))))
(top (truncate (* ←
↪
↪
↪          (cl-getf area :bottom))))))
(list :left left :top top
      :width width ←
↪
      :right (+ left ←
↪
              :bottom (+ top ←
↪
                        height))))))
```

Then we can use it to select part of an image, and then use ImageMagick to extract that part of the image:

```

;;###autoload
(defun sacha-image-write-region ()
  "Copy a section of the image under point to a different file.
This command presents the image with a rectangular area superimposed
on it, and allows moving and resizing the area to define which
part of it to crop.

While moving/resizing the cropping area, the following key bindings
are available:

`q': Exit without changing anything.
`RET': Save the image.
`m': Make mouse movements move the rectangle instead of altering the
      rectangle shape.
`s': Same as `m', but make the rectangle into a square first."
  (interactive)
  (goto-char (point-min))
  (when-let* ((orig-data (buffer-string))
              (area (sacha-image-select-rect "write"))
              (inhibit-read-only t)
              (type (image-crop--content-type ←
                    (left (plist-get area :left))
                    (top (plist-get area :top))
                    (width (plist-get area :width))
                    (height (plist-get area :height)))
                  (with-temp-file (read-file-name "File: ")
                    (set-buffer-multibyte nil)
                    (insert orig-data)
                    (image-crop--process image-crop-crop-command
                                          ←
                                          `((?l . ,left)
                                             (?t . ,top)
                                             (?w . ,width)
                                             (?h . ,height)
                                             (?f . ,(cadr (split-string type "/"))))))))))
    (← orig-data)))

```

`i w` seems like a sensible shortcut for writing a region of an image.

```
(with-eval-after-load 'image
  (keymap-set image-map "i w" #'sacha-image-write-region))
```

TODO Make an image square

```
;;;###autoload
(defun sacha-image-square (filename &optional output-filename)
  (interactive)
  (let* ((size (image-size (create-image filename) t))
        (args
         (delq nil (list
                    "-background"
                    "white"
                    "-gravity"
                    "center"
                    "-resize"
                    (format "%sx%s"
                           (max (car size) (cdr size))
                           (max (car size) (cdr size)))
                    "-extent"
                    (format "%sx%s"
                           (max (car size) (cdr size))
                           (max (car size) (cdr size)))))))
    (apply 'call-process (if output-filename "convert" "mogrify") nil nil nil
           (if output-filename
               (append
                (list filename)
                args
                (list output-filename))
               (append args (list filename)))
           args)))
```

TODO Animate highlighting part of an image

```
;;;###autoload
(defun sacha-image-get-coordinates ()
  (interactive)
  (when-let*
    ((area (sacha-image-select-rect "select"))
     (x1y1x2y2
      (format "%d,%d,%d,%d"
              (plist-get area :left)
              (plist-get area :top)
              (+ (plist-get area :left)
                 (plist-get area :width))
              (+ (plist-get area :top)
                 (plist-get area :height)))))
    (when (called-interactively-p 'any)
      (kill-new x1y1x2y2))
    x1y1x2y2))
```

Ex: 3,991,146,1041


```
ffmpeg -y -loop 1 -i /home/sacha/recordings/2025-01-08_10-50-40.png -t 2 -filter_complex ↵
    ↵ "split[original][copy];[copy]crop=x=0:y=0:w='100*N/2':h=100[cropped];[original]colorlevels=romax=0.2:gomax=
    ↵ \
output.webm && mpv output.webm
```

```
ffmpeg -loop 1 -i /home/sacha/recordings/2025-01-08_10-50-40.png -t 2 -vf "split[original][copy]; \
\ [copy]crop=w='iw-(iw-146)*(t/2)':h='ih-(ih-50)*(t/2)':x='0':y='991',pad=w=iw:h=ih[cropped]; \
\ [original]colorlevels=rimin=0.5:gimin=0.5:bimin=0.5[dimmed]; \ [dimmed][cropped]overlay" \
output.mp4
```

```
ffmpeg -loop 1 -i /home/sacha/recordings/2025-01-08_10-50-40.png -t 2 -vf "split[original][copy]; \
[copy]crop=w='w-(w-146)*(t/2)':h='h-(h-50)*(t/2)':x='0+(iw-146)*(t/4)':y='991+(ih-1041)*(t/4)',pad=w=iw:h=ih \
\ [original]colorlevels=rimin=0.5:gimin=0.5:bimin=0.5[dimmed]; \ [dimmed][cropped]overlay" \
output.mp4
```

```
ffmpeg -loop 1 -i /home/sacha/recordings/2025-01-08_10-50-40.png -t 2 -vf "split[original][copy]; \
[copy]crop=w='w-(w-146)*t/2':h='h-(h-50)*t/2':x='0+(iw-146)*t/4':y='991+(ih-1041)*t/4',pad=w=iw:h=ih[cropp
\ [original]colorlevels=rimin=0.5:gimin=0.5:bimin=0.5[dimmed]; \ [dimmed][cropped]overlay" \
output.webm
```

9.3.4 Artrage

```
;; ##autoload
(defun sacha-artrage-export-png (directory &optional prefix)
  "Change an Artrage script file (arscript) to export images to DIRECTORY.
  If PREFIX is specified, use that instead of image-."
  (interactive "MPath: ")
  (unless (file-directory-p directory)
    (make-directory directory t))
  (while (re-search-forward "[0-9\\.]+s" nil t)
    (replace-match "0.000s"))
  (goto-char (point-min))
  (while (search-forward "<StrokeEvent>" nil t)
    (replace-match (concat
                    "EvType: Command    CommandID: ExportLayer    Idx: -1    Channels: NO    ↵
    ↵ Path: \"
                    directory
                    \"/\" (or prefix "image-")
                    ".png\"
                    "<StrokeEvent>") t t)))
```

9.3.5 SVG

image

```
(auto-image-file-mode -1)
```

Animating SVGs **ANIMATION:VIDEO** Detailed notes are at [Animating SVG topic maps with Inkscape, Emacs, FFmpeg, and Reveal.js.](#)

1. Breaking up a PDF from Supernote

```
(defvar sacha-debug-buffer (get-buffer-create "*temp*"))
;; ##autoload
(defun sacha-sketch-convert-pdf (pdf-file)
  "Returns the SVG filename."
```

```

(interactive "FPDF: ")
(if-let ((links (and (file-exists-p (concat (file-name-sans-extension pdf-file)
↳ ".svg"))
↳
↳
↳ (dom-by-tag
↳
↳ (car (xml-parse-file (concat (file-name-sans-extension pdf-file) ".svg"))))
↳
↳ 'a))))
    ;; copy links over
    (let ((temp-file (concat (make-temp-name "svg-conversion") ".svg"))
          new-file)
      (unwind-protect
        (progn
          (call-process "pdftocairo" nil
↳
↳ sach-debug-buffer nil "-svg" (expand-file-name pdf-file)
↳
↳ temp-file)
          (setq new-file (car
↳
↳ (xml-parse-file temp-file)))
          (dolist (link links)
            (dom-append-child
↳
↳ new-file link))
          (with-temp-file (file-exists-p
↳
↳ (concat (file-name-sans-extension pdf-file) ".svg"))
            (svg-print new-file)))
        (error
          (delete-file temp-file))))
      (delete-file (concat (file-name-sans-extension pdf-file) ".svg"))
      (call-process "pdftocairo" nil sach-debug-buffer nil "-svg"
↳
↳ (expand-file-name pdf-file)
      (expand-file-name (concat (file-name-sans-extension pdf-file) ".svg"))
      (concat (file-name-sans-extension pdf-file) ".svg"))

;;;###autoload
(defun sach-sketch-change-fill-to-style (dom)
  "Inkscape handles these better when we split paths."
  (dolist (path (dom-by-tag dom 'path))
    (when (dom-attr path 'fill)
      (dom-set-attribute
        path 'style
        (if (dom-attr path 'style)
          (concat (dom-attr path 'style) ";fill:"
↳
↳ (dom-attr path 'fill))
          (concat "fill:" (dom-attr path 'fill))))
      (dom-remove-attribute path 'fill)))
    dom)

;;;###autoload
(defun sach-sketch-recolor (dom color-map &optional selector)
  "Colors are specified as ((\"#input\" . \"#output\") ...)."
  (if (symbolp color-map)
    (setq color-map
      (assoc-default color-map
↳
↳ sach-sketch-color-map)))
    (let ((map-re (regex-opt (mapcar 'car color-map))))
      (dolist (path (if selector (dom-search dom selector)

```

```

    ↪ (dom-by-tag dom 'path)))
        (dolist (attr '(style fill))
          (when (and (dom-attr path attr)
                     (string-match ↪

    ↪ map-re (dom-attr path attr)))
              (dom-set-attribute
                path attr
                (replace-regexp-in-string
                  map-re
                  (lambda (match)
                    (assoc-default match color-map))
                  (or (dom-attr path attr) ""))))))

    dom)

;;;###autoload
(defun sacha-sketch-add-bg (dom)
  ;; add background rectangle
  (unless (dom-search dom (lambda (elem) (and (dom-attr elem 'class) (string-match ↪
    ↪ "\\<background\\>" (dom-attr elem 'class)))))
    (let* ((view-box (mapcar 'string-to-number (split-string (dom-attr dom ↪
    ↪ 'viewBox))))
           (bg-node (dom-node 'rect `((x . 0)
                                       ↪
                                       (y . 0)
                                       ↪
                                       (class . ↪
                                       (width . ,(elt ↪
                                       (height . ,(elt ↪
                                       (fill . ↪
    ↪ "#ffffff")))))
           (if (dom-by-id dom "surface1")
               (push bg-node (cddr (car (dom-by-id dom ↪
    ↪ "surface1"))))
               (push bg-node (cddr (car dom))))))
    dom)

;;;###autoload
(defun sacha-sketch-clean (dom)
  "Remove USE and IMAGE tags."
  (dolist (use (dom-by-tag dom 'use))
    (dom-remove-node dom use))
  (dolist (use (dom-by-tag dom 'image))
    (dom-remove-node dom use))
  dom)

;;;###autoload
(defun sacha-sketch-rotate (dom)
  (let* ((old-width (dom-attr dom 'width))
         (old-height (dom-attr dom 'height))
         (view-box (mapcar 'string-to-number (split-string ↪
    ↪ (dom-attr dom 'viewBox))))
         (rotate (format "rotate(90) translate(0 %s)" (- (elt ↪
    ↪ view-box 3)))))

```

```

        (dom-set-attribute dom 'width old-height)
        (dom-set-attribute dom 'height old-width)
        (dom-set-attribute dom 'viewBox (format "0 0 %d %d" (elt view-box 3) ←
→ (elt view-box 2)))
        (dolist (g (dom-by-tag dom 'g))
          (dom-set-attribute g 'transform rotate)))
      dom)

;; ##autoload
(defun sacha-sketch-mix-blend-mode-darken (dom &optional selector)
  (dolist (p (if (functionp selector) (dom-search dom selector) (or selector ←
→ (dom-by-tag dom 'path))))
    (when (and (dom-attr p 'style)
                (not (string-match ←
→ "mix-blend-mode" (dom-attr p 'style))))
      (dom-set-attribute
        p 'style
        (replace-regexp-in-string ";;\\|\\|^;" ""
→ (concat
→ (or (dom-attr p 'style) ""
→ ";mix-blend-mode:darken")))))
      dom)

;; ##autoload
(defun sacha-sketch-color-to-hex (dom &optional selector)
  (dolist (p (if (functionp selector) (dom-search dom selector) (or selector (dom-search dom ←
→ (lambda ←
→ (p) (or (dom-attr p 'style)
→ (dom-attr p 'fill))))))
    (dolist (attr '(style fill))
      (when (dom-attr p attr)
        (dom-set-attribute
          p attr
          (replace-regexp-in-string
            "rgb(\\([0-9\\.]+\\))%, *\\([0-9\\.%]+\\))%, ←
→ *\\([0-9\\.]+\\))%)"
            (lambda (s)
              (color-rgb-to-hex
                (* 0.01 (string-to-number (match-string ←
→ 1 s)))
                (* 0.01 (string-to-number (match-string ←
→ 2 s)))
                (* 0.01 (string-to-number (match-string ←
→ 3 s)))
                2))
              (dom-attr p attr))))))
      dom)

;; default for now, but will support more colour schemes someday
(defvar sacha-sketch-color-map
  '((blue
    ("#9d9d9d" . "#2b64a9")

```

```

        ("#9c9c9c" . "#2b64a9")
        ("#c9c9c9" . "#b3e3f1")
        ("#c8c8c8" . "#b3e3f1")
        ("#cacaca" . "#b3e3f1")
        ("#a6d2ff" . "#ffffff"))
      (t
        ("#9d9d9d" . "#888888")
        ("#9c9c9c" . "#888888")
        ("#cacaca" . "#f6f396")
        ("#c8c8c8" . "#f6f396")
        ("#a6d2ff" . "#ffffff")
        ("#c9c9c9" . "#f6f396"))))

(cl-defun sacha-sketch-svg-prepare (file &key color-map color-scheme new-file)
  "Clean up SVG for publishing."
  (when (string= (file-name-extension file) "pdf")
    (setq file (sacha-sketch-convert-pdf file)))
  (let ((dom (xml-parse-file file)))
    (setq dom (sacha-sketch-clean dom))
    (setq dom (sacha-sketch-color-to-hex dom))
    (setq dom (sacha-sketch-add-bg dom))
    (setq dom (sacha-sketch-change-fill-to-style dom))
    (setq dom (sacha-sketch-recolor dom
      ↪                                     (or color-map
      ↪
      ↪                                     color-scheme
      ↪
      ↪                                     t))))
    (with-temp-file (or new-file file) (svg-print (car dom)))
    (or new-file file)))

```

```

;; ##autoload
(defun sacha-sketch-regroup (dom groups)
  "Move matching paths to their own group.
GROUPS is specified as ((id . (lambda (elem) ..))."
  (dolist (group groups)
    (when-let* ((matches (dom-search dom
      ↪                                     (lambda (elem)
      ↪                                     (funcall (cdr
      ↪ group) elem))))
      ↪ . ,(car group))))))
    (dolist (p matches)
      (dom-remove-node dom p)
      (dom-append-child node p))
    (dom-append-child dom node)))
  dom)
;; ##autoload
(defun sacha-sketch-break-apart (dom selector)
  "Break paths apart.
SELECTOR can be a function that takes the node as an argument and returns non-nil,
or a list of nodes."
  (dolist (path (if (functionp selector) (dom-search dom selector) selector))
    (let ((parent (dom-parent dom path)))
      ;; break apart

```

```

      (when (dom-attr path 'd)
        (dolist (part (split-string (dom-attr path 'd) "M " t " +"))
          (dom-add-child-before
            parent
            (dom-node 'path `((style . ,(or (dom-attr path 'style) 'style) " ")))
            (fill . ,(or (dom-attr path 'fill) "fill" " ")))
          (d . ,(concat "M " part)))
          path))
        (dom-remove-node dom path))))
      dom)

;;;###autoload
(cl-defun sacha-sketch-convert-pdf-and-break-up-paths (pdf-file &key rotate color-map
  color-scheme selector)
  "Convert PDF to SVG and break up paths."
  (interactive (list (read-file-name
    (format "PDF (%s): "
      (sacha-latest-file "~/Dropbox/Supernote/EXPORT/" "pdf"))
    "~/Dropbox/Supernote/EXPORT/"
    (sacha-latest-file "~/Dropbox/Supernote/EXPORT/" "pdf")
    t nil)))
    (lambda (s) (string-match "pdf" s))))
  (let (dom
    (new-file (expand-file-name (concat (file-name-sans-extension pdf-file) "-split.svg"))
      (sacha-sketch-svg-prepare
        file :color-map color-map :color-scheme color-scheme :new-file new-file)
      (setq dom (xml-parse-file new-file))
      (when rotate (setq dom (sacha-sketch-rotate dom)))
      (setq dom (sacha-sketch-break-apart dom (or selector
        (dom-by-tag dom 'path))))
      (with-temp-file new-file
        (svg-print (car dom)))
        new-file))
    (cancel-timer sacha-svg-auto-resize-timer))

```

:CUSTOM_ID: svg-breaking-up-supernote

2. Identifying paths

```

(defvar sacha-svg-auto-resize-timer nil)
;; based on image-mode
;;;###autoload
(defun sacha-svg-resize-with-window (window)
  (when (numberp image-auto-resize-on-window-resize)
    (when sacha-svg-auto-resize-timer
      (cancel-timer sacha-svg-auto-resize-timer))

```

```
(setq sachasvg-auto-resize-timer
  (run-with-idle-timer 1 nil
    #'sachasvg-fit-to-window window))))
;;;###autoload
(defun sachasvg-fit-to-window (window)
  (when (window-live-p window)
    (with-current-buffer (window-buffer window)
      (let ((spec (get-text-property (point-min) 'display)))
        (when (eq (car-safe spec) 'image)
          (let* ((image-width (plist-get (cdr spec) :max-width))
                 (image-height (plist-get (cdr spec) :max-height))
                 (edges (window-inside-pixel-edges window))
                 (window-width (- (nth 2 edges) (nth 0 edges)))
                 (window-height (- (nth 3 edges) (nth 1 edges))))
            ;; If the size has been changed manually (with `+`/`-`),
            ;; then :max-width/:max-height is nil. In that case, do
            ;; no automatic resizing.
            (when (and image-width image-height
                        ;; Don't do resizing if we have a manual
                        ;; rotation (from the `r' command), either.
                        (not (plist-get (cdr spec) :rotation))
                        (or (not (= image-width window-width))
                            (not (= image-height window-height)))))
              (unless image-fit-to-window-lock
                (unwind-protect
                 (progn
                  (setq-local image-fit-to-window-lock t)
                  (ignore-error remote-file-error
                                (setcdr spec
                                          (plist-put
                                           (plist-put (cdr spec) :max-width
                                                         window-width)
                                                         :max-height
                                                         window-height))
                                          (put-text-property (point-min) (1+ (point-min))
                                                             'display spec)))
                  (setq image-fit-to-window-lock nil))))))))))
;;;###autoload
(defun sachasvg-bounding-box (node)
  "Return (x1 y1 x2 y2) for NODE."
  (require 's)
  (pcase (dom-tag node)
    ('rect
     (list
      (string-to-number (dom-attr node 'x))
      (string-to-number (dom-attr node 'y))
      (+ (string-to-number (dom-attr node 'x))
         (string-to-number (dom-attr node 'width)))
      (+ (string-to-number (dom-attr node 'y))
         (string-to-number (dom-attr node 'height)))))))
```

```

(+ (string-to-number (dom-attr node 'y)) (string-to-number ←
↪ (dom-attr node 'height))))
  ('path
    (let ((path (dom-attr node 'path))
          (x1 most-positive-fixnum)
          (y1 most-positive-fixnum)
          (x2 most-negative-fixnum)
          (y2 most-negative-fixnum)
          (x 0)
          (y 0)
          (i 0))
      (dolist (seg (s-slice-at " *[Mmc] *" path))
        (unless (string= (string-trim seg) "")
          (setq seg (split-string seg "[,]" i 0))
          (let ((points (mapcar 'string-to-number (cdr ←
↪ seg))))
            (pcase (car seg)
              ((or "m" "M")
               (if (or (eq (car seg) ←
↪ "M") (= i 0))
                   ;; ←
                   (setq x ←
↪ (car points)
↪ y (cadr points))
                   ;; m, so ←
                   (setq x (+ x ←
↪ (car points)
↪ y (+ y (cadr points)))))
              (when (< x x1) (setq x1 x))
              (when (< y y1) (setq y1 y))
              (when (> x x2) (setq x2 x))
              (when (> y y2) (setq y2 y)))
            ("c"
             (let ((old-x x) (old-y y))
               (dolist (set ←
↪ (seq-partition points 6))
                 ;; ←
                 (setq x ←
↪ (+ x (elt set 4))
↪ y (+ y (elt set 5)))
                 (when (< ←
↪ x x1) (setq x1 x))
                 (when (< ←
↪ y y1) (setq y1 y))
                 (when (> ←
↪ x x2) (setq x2 x))
                 (when (> ←
↪ y y2) (setq y2 y))))
              )
            ("C"
             (dolist (set ←
↪ (seq-partition points 2))
               (setq x (elt set 0))

```



```

                                (setq y (elt set 1))
                                (when (and x y)
                                  (when (< ←
→ x x1) (setq x1 x))
                                  (when (< ←
→ y y1) (setq y1 y))
                                  (when (> ←
→ x x2) (setq x2 x))
                                  (when (> ←
→ y y2) (setq y2 y))))))
                                (cl-incf i))))
                                (list x1 y1 x2 y2)))
                                )
                                )

```

```

(ert-deftest sachasvg-bounding-box ()
  (should (equal (sachasvg-bounding-box "M 15.838959,27.678234 C ←
→ 15.838959,27.678234 50.667557,45.01362 62.948412,30.731177 75.229269,16.448732 ←
→ 98.309577,20.617771 102.23147,26.236269"))))
  (should
    (equal (sachasvg-bounding-box "M 1025.609375 852.070312 C 1025.660156 ←
→ 853.179688 1026.097656 854.332031 1026.914062 854.871094 C 1028.179688 855.707031 ←
→ 1033.238281 855.589844 1033.761719 854.746094 C 1034.320312 853.839844 ←
→ 1032.726562 851.054688 1031.199219 850.105469 C 1030.3125 849.554688 1029.003906 ←
→ 849.210938 1027.953125 849.207031 C 1027.144531 849.207031 1026.625 849.296875 ←
→ 1026.109375 849.976562 C 1025.710938 850.496094 1025.574219 851.332031 ←
→ 1025.609375 852.070312"))
      '(1025.609375 849.207031 1033.761719 854.871094)))
  (should
    (equal (sachasvg-bounding-box "m 1160.0156,382.75391 c 0.3867,4.04296 ←
→ 1.2696,9.02343 1.1719,12.88281 -1.6953,1.875 -5.8711,0.25781 -8.3906,1.05469 ←
→ -0.6055,0.26171 -0.9063,0.65234 -0.9063,1.28906 0,0.64844 0.2969,0.98047 ←
→ 0.8907,1.21094 2.5664,0.20703 5.1289,0.41406 7.6953,0.62109 1.3672,1 ←
→ 0.9218,4.21484 3.4453,4.29297 0.7344,0.0273 1.0742,-0.29688 1.2109,-0.88281 ←
→ 0.035,-1.375 -0.625,-2.5 0.457,-3.56641 2.9375,-1.20313 5.8711,-2.41016 ←
→ 8.8086,-3.61328 0.9727,-0.47656 1.793,-1.08203 1.7539,-2.0625 -0.035,-0.99219 ←
→ -0.8789,-1.27344 -1.871,-1.17969 -2.9336,0.74219 -5.8672,1.48047 -8.7969,2.22266 ←
→ -1.8281,-2.50782 -1.6758,-7.36328 -2.1953,-11.23828 -0.2813,-0.95704 ←
→ -1.1446,-1.80469 -2.1875,-1.86719 -0.7305,-0.043 -0.9922,0.26953 -1.086,0.83594 m ←
→ 11.9219,24.23828 c 0.7188,2.97656 1.4375,5.94922 2.1563,8.92187 -0.027,1.29297 ←
→ -1.125,3.60156 -2.3438,4.05078 -1.1836,0.44141 -3.1602,-0.78515 -4.4961,-1.76172 ←
→ -1.5625,-1.13671 -2.7851,-2.75781 -4.0351,-4.40234 -2.625,-2.01953 ←
→ 0.1328,-5.14844 -1.3594,-6.60156 -0.9766,-0.60938 -2.9571,0.32812 -3.1133,1.64844 ←
→ -0.5391,1.83984 -0.3594,4.5 0.7695,6.35546 1.9532,2.94532 5.1953,6.72266 ←
→ 8.3203,7.9336 1.6993,0.57422 4.7149,0.65625 6.3125,0.19531 1.0039,-0.28906 ←
→ 1.4297,-0.96094 1.8633,-2.05078 0.8008,-1.99609 1.5196,-4.24609 1.375,-6.26953 ←
→ -0.8554,-2.90625 -0.9883,-6.82031 -3.4179,-8.94922 -1.0157,-0.50781 ←
→ -1.875,-0.0508 -2.0313,0.92969 m -13.3789,26.9375 c -0.078,1.33593 ←
→ -0.1328,2.92187 0.293,4.17968 0.9453,1.51172 1.8867,3.02344 2.8281,4.53907 ←
→ -0.6524,0.73828 -1.3086,1.47656 -1.9609,2.21484 -0.7305,2.76172 -0.875,9.38672 ←
→ 0.1484,12.29297 1.0859,2.86719 4.3516,4.23047 7.0312,5.91016 1.9375,0.79296 ←
→ 4.3946,0.40234 6.3516,-0.21485 0.6641,-0.21094 1.2969,-0.46875 1.6484,-0.96484 ←
→ 0.5274,-0.7461 0.5274,-2.09766 -0.027,-2.64844 -1.9102,0.008 -3.8203,0.0156 ←
→ -5.7305,0.0273 -1.7773,-0.49218 -4.207,-1.9414 -5.6484,-3.60156 -1.8672,-2.39453 ←
→ -0.8125,-5.0625 -0.9766,-7.5625 0.1758,-1.18359 0.8164,-2.70703 1.8867,-3.11328 ←
→ 2.5977,0.14844 5.1915,0.29688 7.7891,0.44531 1.0625,-0.0664 1.918,-0.27734 ←
→ 2.8945,-1.19531 1.2657,-1.19531 2.086,-2.81641 2.3008,-4.16406 0.3164,-2 ←
→ 0.1094,-4.34375 -0.5312,-6.33203 -0.2149,-0.66016 -0.4805,-1.29297 ←

```

```

→ -1.0157,-1.63282 -0.4882,-0.30859 -1.1914,-0.30078 -1.6093,0.0156 -1.4844,1.51562 ←
→ 0.1953,4.54687 -0.2383,6.68359 -0.2969,0.9375 -1.3047,1.9961 -2.2344,2.72266 ←
→ -0.9765,0.76562 -1.7734,1.05469 -2.7187,0.95703 -1.461,-0.14844 -3.1953,-1.41797 ←
→ -4.5274,-2.86328 -1.2578,-1.37109 -2.5078,-3.19922 -2.7187,-4.59375 ←
→ -0.1289,-0.86719 0.2734,-1.10938 1.1289,-0.38672 1.3867,1.78125 2.7695,3.55859 ←
→ 4.1562,5.33594 0.586,0.28515 1.2813,0.2539 1.7071,-0.125 0.6796,-0.60547 ←
→ 0.6523,-1.85156 0.25,-2.94922 -0.6368,-1.73828 -2.043,-3.77734 -3.1602,-5.26953 ←
→ -0.7656,-1.01953 -1.668,-1.77344 -2.8086,-1.94922 -0.6992,-0.10938 -1.5234,0.004 ←
→ -2.2461,0.37891 -1.6445,0.85937 -2.1758,2.46093 -2.2617,3.86328 m ←
→ -44.8516,12.89843 c -0.1562,7.03125 -0.1875,14.48047 0.1016,21.36719 ←
→ 0.2305,0.60938 0.5703,0.91016 1.1914,0.91406 0.625,0 0.9648,-0.30078 ←
→ 1.1953,-0.89843 0.6914,-3.53125 -0.582,-10 0.8906,-11.95313 4.9375,6.73438 ←
→ 15.668,16.79688 20.3321,24.84766 -1.0469,9.58203 -3.8399,19.17187 -6.2578,28.75 ←
→ -1.8321,3.38672 -3.668,6.77344 -5.5039,10.16015 -0.1485,1.13672 0.3281,2.05469 ←
→ 1.3789,2.11329 1.0625,0.0586 2.0625,-0.78516 2.8046,-1.76954 1.8125,-2.41406 ←
→ 3.2461,-5.60937 4.129,-8.1914 2.9101,-11.14063 5.621,-21.85156 7.3515,-33.25781 ←
→ -3.9726,-6.83594 -13.1719,-14.88672 -17.6406,-20.35938 -1.8203,-2.29297 ←
→ -6.4102,-8.75 -6.3594,-9.76953 0.035,-0.78906 2.4805,-1.89844 3.8164,-2.04688 ←
→ 1.668,0.19141 3.3321,0.38672 5.057813 0.875,-0.26563 1.3047,-1.26953 ←
→ 0.7383,-2.34766 -0.3984,-0.7539 -1.0117,-1.07031 -1.7031,-1.26562 ←
→ -2.0547,-0.57031 -5.2188,-0.38281 -7.2813,-0.0703 -1.6797,0.16015 -3.9687,1.58203 ←
→ -4.1836,3.19921 m 35.4766,21.35547 c -0.668,0.67188 -0.7461,2.96485 0.039,3.65625 ←
→ 0.6523,0.56641 1.9531,0.3086 2.9531,-0.67578 0.9961,-0.98437 1.2695,-2.28515 ←
→ 0.6836,-2.9414 -0.7071,-0.79297 -3.0117,-0.70313 -3.6758,-0.0391 m ←
→ 25.8633,-0.39062 c -2.7031,1.03906 -5.4024,2.07812 -8.1055,3.11719 ←
→ -1.3398,-0.0742 -2.6836,-0.14844 -4.0234,-0.22266 -0.9102,0.23047 -1.3477,1.27734 ←
→ -0.7813,2.34766 0.3946,0.75 1.0274,1.08203 1.7227,1.26953 1.3515,0.36328 ←
→ 2.9023,0.0469 4.2109,-0.27344 2.4883,-0.60547 6.1172,-1.4375 8.1797,-2.63281 ←
→ 0.7969,-0.46094 1.2578,-1.35938 1,-2.41016 -0.2578,-1.05469 -1.0547,-1.3125 ←
→ -2.2031,-1.19531 m 0.2304,28.30078 c 0.4258,1.11719 -0.2382,2.55078 ←
→ -1.375,2.75781 -1.871,-0.043 -4.7148,-3.05078 -6.0546,-5.01562 -0.4727,-0.92188 ←
→ -0.4532,-1.77344 -0.012,-2.64063 0.4454,-0.87109 1.3633,-1.84765 2.0664,-1.92187 ←
→ 1.8711,0.53906 4.0547,4.24218 5.375,6.82031 m 3.0899,-2.16406 c -1.0859,-2.19141 ←
→ -2.168,-4.38282 -3.25,-6.57422 1.2812,-0.79688 2.5586,-1.59375 3.8398,-2.39063 ←
→ 0.6172,-0.96093 0.6602,-3.09765 -0.1601,-3.80468 -2.2735,-1.32813 -4.2344,3.59765 ←
→ -6.8633,3.10546 -3.6523,-0.54296 -7.3047,-1.08203 -10.957,-1.625 -2.8828,0.15625 ←
→ -6.6953,-0.55468 -8.8477,0.5586 -0.6953,0.88281 -0.4726,2.82031 0.6484,3.00781 ←
→ 3.2657,0.89844 7.7657,1.15234 10.7071,1.50391 0.6289,0.41797 0.2226,6.12109 ←
→ 1.4258,8.48437 1.0195,1.99219 2.8632,3.76563 4.8945,5.17969 1.4844,1.03516 ←
→ 2.7617,1.15234 4.2695,1.03516 1.3711,-0.10547 3.086,-0.37891 3.8164,-1.3711 ←
→ 0.9766,-1.32812 0.7188,-5.28125 0.4766,-7.10937 M 1167,513.47266 1167.5273,514 ←
→ 1167,514.52734 1166.4727,514 1167,513.47266 m 10.8203,-7.64844 c 0.3906,2.33594 ←
→ 0.7774,4.66797 1.1641,7.0039 -0.4024,1.29297 -2.8242,3.76172 -4.0078,4.0625 ←
→ -0.8868,0.22657 -1.586,-0.41796 -2.3125,-1.30468 -1.5469,-2.1836 -3.0938,-4.3711 ←
→ -4.6407,-6.55469 -0.875,-0.5 -2.0898,-0.54297 -3.1992,0.0352 -1.1719,0.60937 ←
→ -1.8789,1.70703 -2.1406,2.83203 -0.8633,2.57812 1.2852,4.94922 2.1484,7.125 ←
→ -0.4062,1.29687 -0.8086,2.59375 -1.2148,3.89062 -0.3281,2.24219 -0.2422,4.94922 ←
→ 0.3203,7.21875 0.4297,1.72656 1.2578,3.50391 2.5195,5.2461 0.7696,1.0625 ←
→ 1.4141,1.71875 2.4258,1.92187 2.5938,0.52344 7.75,-0.74609 10.3945,-1.55078 ←
→ 1.0547,-0.32422 1.7735,-0.68359 1.9766,-1.78516 0.1992,-1.08984 -0.2422,-1.89843 ←
→ -1.0703,-2.01953 -2.9961,0.375 -5.9961,0.75391 -8.9961,1.12891 -2.207,-1.27735 ←
→ -4.4453,-4.15235 -4.6523,-6.15235 -0.086,-1.98828 0.4921,-4.85937 1.9531,-5.94531 ←
→ 2.5547,0.0547 5.1133,0.10938 7.6719,0.16406 1.5898,-0.55468 3.7968,-2.25 ←
→ 4.9414,-3.92187 1.125,-1.64063 1.375,-3.51953 1.2812,-5.1875 -0.3476,-2.22266 ←
→ -0.8398,-5.41016 -2.5117,-6.94922 -0.9102,-0.53125 -1.8203,-0.11328 ←
→ -2.0508,0.74219")

```

nil

←

→

; don't know what

)))

```

;;;###autoload
(defun sacha-svg-display (buffer-name svg &optional highlight-id full-window)
  "HIGHLIGHT-ID is a string ID or a node."
  (with-current-buffer (get-buffer-create buffer-name)
    (when highlight-id
      ;; make a copy
      (setq svg (with-temp-buffer (svg-print svg) (car ←
→ (xml-parse-region (point-min) (point-max))))))
      (if-let* ((path (if (stringp highlight-id) (dom-by-id svg ←
→ highlight-id) highlight-id))
                (view-box (split-string ←
→ (dom-attr svg 'viewBox)))
                (box ←
→ (sacha-svg-bounding-box path))
                (parent (car path)))
        (progn
          ;; find parents for possible rotation
          (while (and parent (not (dom-attr parent ←
→ 'transform))))
            (setq parent (dom-parent svg ←
→ parent)))
          (dom-set-attribute path 'style
            (concat (dom-attr path 'style) "; stroke: 1px ←
→ red; fill: #ff0000 !important")))
          ;; add a crosshair
          (dom-append-child
            (or parent svg)
            (dom-node 'path
              ←
              → `((d .
              →
              → ,(format "M %f,0 V %s M %f,0 V %s M 0,%f H %s M 0,%f H %s"
              →
              → (elt box 0)
              →
              → (elt view-box 3)
              →
              → (elt box 2)
              →
              → (elt view-box 3)
              →
              → (elt box 1)
              →
              → (elt view-box 2)
              →
              → (elt box 3)
              →
              → (elt view-box 2)))
              (stroke-dasharray . "5,5")
              (style . "fill:none;stroke:gray;stroke-width:3px")))))
            (error "Could not find %s" highlight-id)))
      (let* ((inhibit-read-only t)
              (image (svg-image svg)))

```

```

                                (edges (window-inside-pixel-edges ←
↪ (get-buffer-window))))
                                (erase-buffer)
                                (if full-window
                                    (progn
                                        (delete-other-windows)
                                        (switch-to-buffer (current-buffer)))
                                    (display-buffer (current-buffer)))
                                (insert-image (append image
                                                        ↪
                                                        (list :max-width
                                                            ↪
                                                            (floor (* 0.8 (- (nth 2 edges) ↪
↪ (nth 0 edges))))
                                                            :max-height
                                                            ↪
                                                            (floor (* 0.8 (- (nth 3 edges) ↪
↪ (nth 1 edges)))) )))
                                    ;; (sacha-svg-resize-with-window (selected-window))
                                    ;; (add-hook 'window-state-change-functions ↪
↪ #'sacha-svg-resize-with-window t)
                                    (current-buffer))))

;;###autoload
(cl-defun sacha-svg-identify-paths (filename &key selector node-func dom)
  "Prompt for IDs for each path in FILENAME."
  (interactive (list (read-file-name "SVG: " nil nil
↪
↪
↪ (lambda (f)
↪
↪ (or ↪
↪ (string-match "\\\\.svg$" f)
↪
↪
↪ (file-directory-p f))))))
  (let* ((dom (or dom (car (xml-parse-file filename))))
        (paths (if (functionp selector)
↪
↪ (dom-search dom selector)
↪ (or selector
↪
↪ (dom-by-tag dom 'path)))
        (vertico-count 3)
        (ids (seq-keep (lambda (path)
↪
↪ (and (dom-attr path 'id)
↪
↪ (unless (string-match "\\(path\\|rect\\)[0-9]+"
↪
↪
↪ (or (dom-attr path 'id) "path0"))
↪
↪ (dom-attr path 'id))))
        paths))
        (edges (window-inside-pixel-edges (get-buffer-window)))
        id)
    (sacha-svg-display "*image*" dom nil t)

```

```

(dolist (path paths)
  ;; display the image with an outline
  (unwind-protect
    (progn
      (sacha-svg-display "*image*" dom
        (if (functionp node-func)
            (funcall node-func path dom)
            (setq id (completing-read
                      (format "ID (%s): " (dom-attr path 'id))
                      ids))
              ;; already exists, merge with
              (if-let* ((old (dom-by-id dom id)))
                (progn
                  (dom-set-attribute
                    (dom-attr (dom-by-id dom id) 'd)
                    (concat
                      (dom-attr (dom-by-id dom id) 'd)
                      (replace-regexp-in-string "^m" "M"
                        (dom-attr path 'd))))
                  (dom-remove-node dom path)
                  (setq id
                    (dom-set-attribute path
                      (add-to-list 'ids id))))
                ;; save the image just in case we get interrupted halfway through
                (with-temp-file filename
                  (svg-print dom))))))
    existing element)

;;###autoload
(defun sacha-svg-identify-rects (filename)
  (interactive (list (read-file-name "SVG: " nil nil
    (lambda (f)
      (string-match "\\..svg$" f)
      (file-directory-p f))))))
  (sacha-svg-identify-paths
   filename
   :selector
   (lambda (elem)
     (and (eq (dom-tag elem) 'rect)

```

[illegible]

```

      :node-func
      (lambda (elem dom)
        (let* ((current-link-node (sacha-dom-closest dom elem 'a))
                (current-title-node (or (dom-by-tag elem 'title)
                                         (dom-by-tag current-link-node 'title))))
          (title (string-trim
                  (completing-read
                   "Title: "
                   (mapcar 'car links-from-text)
                   nil nil
                   (dom-text current-title-node))))
                (link (string-trim
                      (read-string
                       "URL: "
                       (or (dom-attr current-link-node 'href)
                           (assoc-default title links-from-text 'string=)))
                      (cond
                       ((and current-link-node (not (string= link "")))
                        (dom-set-attribute elem
                                             'style
                                             "stroke: blue; stroke-dasharray: 4; fill: #006fff; fill-opacity: 0.25")
                        (dom-set-attribute current-link-node 'href link))
                       ((and current-link-node (string= link ""))
                        (dom-add-child-before
                         (dom-parent dom current-link-node)
                         elem)
                        (dom-remove-node current-link-node))
                       ((and (null current-link-node) (not (string= link "")))
                        (setq current-link-node (dom-node
                                                  'a
                                                  `((href . ,link)
                                                    (class . "link")))))
                        (dom-add-child-before (dom-parent dom elem)
                                              (dom-remove-node dom elem)
                                              (dom-append-child current-link-node elem)
                                              (dom-remove-attribute elem 'fill)
                                              (dom-set-attribute elem

```

```

↪          'style
↪
↪          "stroke: blue; stroke-dasharray: 4; fill: ↪
↪          #006fff; fill-opacity: 0.25")
↪          (dom-set-attribute
↪            elem
↪            'class
↪            (if (dom-attr elem 'class)
↪                (concat (dom-attr elem ↪
↪          'class) " link-rect")
↪                "link-rect"))))
↪          (cond
↪            ((and (string= title "") current-title-node)
↪              (dom-remove-node current-title-node))
↪            ((and (not (string= title "")) (not ↪
↪          current-title-node))
↪              (dom-append-child current-link-node (dom-node ↪
↪          'title nil title)))
↪            ((and (not (string= title "")) current-title-node)
↪              (setf (car (dom-children current-title-node))
↪                    title))))))
;;###autoload
(defun sacha-svg-update-links-from-text (filename)
  (interactive (list (read-file-name
↪
↪          "SVG: " nil
↪
↪          (if (file-exists-p (concat (file-name-sans-extension (buffer-file-name)) ".svg"))
↪
↪          (concat (file-name-sans-extension (buffer-file-name)) ".svg")
↪          (cdr (sacha-embark-image)))
↪          (lambda (f)
↪            (or (string-match "\\\\.svg$" f)
↪              (file-directory-p f))))))
↪          (let ((dom (car (xml-parse-file filename)))
↪                (links-from-text (sacha-org-links-from-file (concat ↪
↪          (file-name-sans-extension filename) ".txt"))))
↪            (dolist (link (dom-by-tag dom 'a))
↪              (when (and
↪                (assoc-default (dom-text (dom-by-tag ↪
↪          link 'title))
↪                links-from-text)
↪                (not (string=
↪                (dom-attr link ↪
↪          'href)
↪                (assoc-default ↪
↪          (dom-text (dom-by-tag link 'title))
↪                links-from-text))))
↪              (dom-set-attribute
↪                link
↪                'href
↪                (assoc-default (dom-text (dom-by-tag link 'title))

```



```

↩      links-from-text))))
      (with-temp-file filename
        (svg-print dom))))

```

3. Linking paths I may want to add hyperlinks around any path that has a title.

```

;; ##autoload
(defun sacha-dom-closest (dom node tag)
  (let ((current node))
    (while (and current (not (eq (dom-tag current) tag)))
      (setq current (dom-parent dom current)))
    current))

;; ##autoload
(defun sacha-svg-save-links (widget &rest ignore)
  (let ((inputs (widget-get widget :inputs))
        (dom (widget-get widget :dom)))
    (dolist (input inputs)
      (let ((link (xml-escape-string (string-trim (widget-value (cdr ↩
↩ input))))))
        (node (widget-get (cdr input) :node))
        (link-node (widget-get (cdr input) ↩
↩ :link-node))
        (parent)
        (cond
         ;; remove link if linked
         ((string= link "")
          (when link-node
            (setq parent (dom-parent dom link-node))
            (dolist (child (dom-children link-node))
              ;; move all the parent's ↩
↩ children to the grandparent
              (dom-add-child-before parent ↩
↩ child link-node))
            (dom-remove parent link-node)
            (widget-put (cdr input) :link-node nil)))
         ;; update link
         (t
          (link-node
           (dom-set-attribute link-node 'href link))
          ;; add a link
          (let ((new-link (dom-node 'a `((href . ,link)))))
            (setq parent (dom-parent dom node))
            (dom-add-child-before parent new-link node)
            (dom-remove-node parent node)
            (dom-append-child new-link node)
            (widget-put (cdr input) :link-node ↩
↩ new-link))))))
      (with-temp-file (widget-get widget :file)
        (svg-print dom))
      (find-file (widget-get widget :file))))

(defvar-local original-styles nil)

;; (defun
;;       sacha-svg-highlight-from-widget (widget)
;;       (let ((new-dom (copy-tree (widget-get widget :dom)))

```

```

;;          (image (widget-get widget :image)))
;;          (dom-by-id new-dom (dom-attr (widget-get widget :node) 'id))

;;          (put-text-property 0 (length image) 'display (svg-image new-dom))

;;          ;; highlight the widget's node
;;          ;; update the image being displayed

;;      )
;;  )

;;;###autoload
(defun sachasvg-link-rects (filename)
  "Add links to rects in FILENAME.
Exclude the background rect."
  (interactive (list (read-file-name "SVG: " nil nil (lambda (f)

    ↵
    ↵
    ↵          (or (string-match "\\\\.svg$" f)
    ↵
    ↵
    ↵          ↵
    ↵ (file-directory-p f))))))
    (let* ((dom (car (xml-parse-file filename)))
           (rects (seq-remove (lambda (elem)
    ↵
    ↵          (and (dom-attr elem 'class)
    ↵
    ↵          (string-match "\\<background\\>" ↵
    ↵ (dom-attr elem 'class))))
    ↵
    ↵          (dom-by-tag dom 'rect)))
           image
           inputs)
    (with-current-buffer (get-buffer-create "*svg*")
      (erase-buffer)
      (setq image
    ↵          (propertize "x" 'display (svg-image dom)))
      (widget-insert image "\n")
      (seq-map-indexed
        (lambda (node i)
          ;; save the original style for including later
          (push (cons (dom-attr node 'id) (dom-attr node 'style))
    ↵          original-styles)
          (let ((link (sachasvg-dom-closest dom node 'a)))
            (widget-insert (format "Rect %d: " ) i)
            (push (list
    ↵          title
    ↵          (widget-create ↵
    ↵ 'editable-field
    ↵
    ↵          :image
    ↵
    ↵          image
    ↵
    ↵          :dom
    ↵
    ↵          dom

```

```

↪                                     :node ↵
↪                                     node ↵
↪                                     :link-node ↵
↪                                     link ↵
↪                                     :help-echo ↵
↪                                     #'sacha-svg-highlight-from-widget ↵
↪                                     :format ↵
↪                                     "Link: %v" ↵
↪                                     :value ↵
↪                                     (and link (dom-attr link 'href))) ↵
↪                                     (widget-create ↵
↪ 'editable-field                                     ↵
↪                                     :image ↵
↪                                     image ↵
↪                                     :dom ↵
↪                                     dom ↵
↪                                     :node ↵
↪                                     node ↵
↪                                     :help-echo ↵
↪                                     #'sacha-svg-highlight-from-widget ↵
↪                                     :link-node ↵
↪                                     link ↵
↪                                     :format ↵
↪                                     "Title: %v" ↵
↪                                     :value ↵
↪                                     (dom-text (or (dom-by-tag node 'title) ↵
↪                                     ↵
↪                                     (and link (dom-by-tag link 'title)))))) ↵
↪                                     inputs))) ↵
↪                                     rects) ↵
↪ (widget-create 'push-button ↵
↪ filename                                     :file ↵
↪                                     :inputs ↵

```



```

↪ filename
↪ inputs
↪ #'sacha-svg-save-links
                                (widget-setup)
                                (widget-minor-mode)
                                (goto-char (point-min))
                                (switch-to-buffer (current-buffer))))
;; (sacha-svg-link-paths (sacha-latest-file "~/sync/sketches"))
                                :inputs ↪
                                :dom dom
                                :notify ↪
                                "Save")

```

4. Sorting paths

```

;;;###autoload
(defun sacha-svg-reorder-paths (filename &optional ids output-filename)
  "Sort paths in FILENAME."
  (interactive (list (read-file-name "SVG: " nil nil (lambda (f) (string-match ↪
↪ "\\\\.svg$" f)))
                                nil ↪
↪ (read-file-name "Output: ")
  (let* ((dom (car (xml-parse-file filename)))
         (paths (dom-by-tag dom 'path))
         (parent (dom-parent dom (car paths)))
         (ids-left
          (nreverse (seq-keep (lambda (path)
↪
↪ (dom-attr path 'id) "path0"))
                                ↪
↪ (dom-attr path 'id)))
                                ↪
↪ paths)))
    (list)
    (when (called-interactively-p)
      (while ids-left
        (sacha-svg-display "*image*" dom (car ids-left))
        (let ((current (completing-read
↪
↪ (format "ID (%s): "
                                ↪
↪ (car ids-left))
                                ↪
↪ ids-left nil nil nil nil (car ids-left)))
              (node
               (add-to-list 'ids current)
               (setq ids-left (seq-remove (lambda (o) (string= ↪
↪ o current)) ids-left))))
          (if ids ;; reorganize under the first path's parent
              (progn
                (dolist (id ids)
                  (if-let ((node (car (dom-by-id dom id))))
                      (progn
                        (dom-remove-node ↪
↪ dom node)
                        ↪
↪ (dom-append-child parent node))

```

```

                                (message "Could not find %s" id)))
                                (with-temp-file (or output-filename filename)
                                (svg-print dom))))
(nreverse (seq-keep (lambda (path)
↳
                                (unless (string-match "path[0-9]+" (or (dom-attr path 'id) "path0"))
↳
                                (dom-attr path 'id)))
↳
↳
                                (dom-by-tag dom 'path))))))

```

```
(sacha-svg-reorder-paths "~/proj/2023-12-audio-workflow/map.svg")
```

```
(sacha-svg-reorder-paths "~/proj/2023-12-audio-workflow/map.svg"
```

```

↳ '("t-start" "h-audio" "h-capture" "t-but" "t-mic" "h-mic"
↳
↳ "t-reviewing" "h-reviewing"
↳
↳ "t-words" "h-words" "t-workflow" "h-workflow"
↳
↳ "t-lapel"
↳
↳ "h-lapel"
↳
↳ "mic-recorder"
↳
↳ "t-recorder"
↳
↳ "h-recorder"
↳
↳ "t-syncthing" "h-sync"
↳
↳ "t-keywords" "h-keywords" "t-keyword-types"
↳
↳ "t-lines"
↳
↳ "h-lines"
↳
↳ "t-align"
↳
↳ "h-align"
↳
↳ "arrow"
↳
↳ "t-org" "h-org" "t-todo" "h-todo" "h-linked"
↳
↳ "t-jump" "h-jump"
↳
↳ "t-waveform" "h-waveform"
↳
↳ "t-someday"
↳
↳ "h-sections"
↳
↳ "t-speech-recognition" "h-speech-recognition"

```

```

↳      "t-ai"
↳      "h-ai"
↳      "t-summary"
↳      "extra")
↳ "~/proj/2023-12-audio-workflow/map-output.svg"
)

```

5. Animating paths in order

```

;;;###autoload
(defun sacha-animate-svg-paths (filename output-dir)
  "Add one path at a time. Save the resulting SVGs to OUTPUT-DIR."
  (unless (file-directory-p output-dir)
    (make-directory output-dir t))
  (let* ((dom (xml-parse-file filename))
         (paths (seq-filter (lambda (e) (dom-attr e 'style))
                             (dom-by-tag dom 'path)))
         (total (length paths))
         (frame-num (length paths))
         result)
    (dolist (elem paths)
      (dom-set-attribute elem 'style
                          (concat
                           (dom-attr elem 'style)
                           ";mix-blend-mode:darken")))
      (with-temp-file (expand-file-name (format "frame-%03d.svg" (1+
↳ frame-num)) output-dir)
        (xml-print dom))
        (dolist (elem paths)
          (dom-set-attribute elem 'style
                              (concat
                               (dom-attr elem 'style)
                               ";fill-opacity:0")))
          (with-temp-file (expand-file-name
↳ (format "frame-%03d.svg"
↳ (- total frame-num))
↳ output-dir)
            (message "%03d" frame-num)
            (dom-set-attribute elem 'style
                                (concat (dom-attr elem 'style)
↳ ";fill-opacity:1")))

```

```

                                (push (list (format "frame-%03d.svg"
↪                                (1+ (- total frame-num)))
↪                                (dom-attr elem 'id))
                                result)
                                (setq frame-num (1- frame-num))
                                (xml-print dom)))
                                (reverse result)))

```

Example:

```

(sacha-animate-svg-paths "~/proj/2023-12-audio-workflow/map-output.svg"
↪
↪ "~/proj/2023-12-audio-workflow/frames/")

```

for FILE in *.svg; do inkscape -export-type=png -export-dpi=96 -export-background-opacity=1 \$FILE; done

One image per second

- `ffmpeg -i frame-%03d.svg.png -vf palettegen palette.png`
- `ffmpeg -f image2 -framerate 1 -i frame-%03d.svg.png -loop -1 animation.gif`
- `ffmpeg -framerate 1 -i frame-%03d.svg.png -i palette.png -lavfi "paletteuse" -loop -1 animation.gif`

```

;; ##autoload
(defun sacha-ffmpeg-animate-images (files output-file &optional framerate)
  "Make an animated GIF or WEBM out of FILES.
  Save it to OUTPUT-FILE.
  If FRAMERATE is specified, use that instead of 30."
  (setq framerate (or framerate 30))
  (if (string-match "\\..webm$" output-file)
      (let ((compile-media-ffmpeg-arguments
              (append compile-media-ffmpeg-arguments
                        (list "-r"
↪                        (number-to-string framerate))))
              (compile-media `((video ,@ (mapcar (lambda (o) (list ↪
↪ :source o :duration-ms (/ 1000.0 framerate)
↪
↪
↪ :before-input
↪
↪
↪ (list ↪
↪ "-width" compile-media-output-video-width))))
↪
↪
↪ files)))
      (output-file))
      (with-current-buffer (get-buffer-create "*gif*")
        (erase-buffer)
        (let ((frame-input (seq-mapcat (lambda (o) (list "-i" o)) files))

```



```

        (palette (make-temp-file "palette" nil ←
        ↪ ".png"))))
        (insert "ffmpeg "
        (string-join (append ←
        ↪ frame-input (list "-vf" "palettegen" "-y" palette)) " ")
        "\n")
        (apply #'call-process "ffmpeg" nil t t
        (append frame-input (list "-vf" ←
        ↪ "palettegen" "-y" palette)))
        (insert "ffmpeg "
        (string-join (append ←
        ↪ (list "-i" palette "-lavfi" "paletteuse")
        ↪ (list "-framerate" ←
        ↪ (number-to-string framerate))
        ↪ frame-input
        ↪ (list "-loop" "-1" "-y" ←
        ↪ output-file)) " ")
        "\n")
        (apply #'call-process "ffmpeg" nil t t
        (append (list "-i" palette ←
        ↪ "-lavfi" "paletteuse")
        ↪ (list "-framerate" (number-to-string framerate))
        ↪ frame-input
        ↪ (list "-loop" "-1" "-y" output-file)))
        (delete-file palette))
        (display-buffer (current-buffer)))
        output-file)

```

6. RevealJS CSS animation of sketches

```

;; ##autoload
(defun sacha-reveal-svg-animation (slide)
  (string-join
    (seq-map-indexed
      (lambda (step-ids i)
        (format "%s { fill: #f6f396; transition: fill %ds; ↪
        ↪ transition-delay: %ds }"
        (mapconcat
          (lambda (id) (format ↪
          (split-string step-ids ",")
          ", ")
          highlight-duration
          (* i highlight-duration)))
        (split-string (elt slide 1) ";")
        "\n"))
    ;; ##autoload
    (defun sacha-reveal-svg-highlight-different-colors (slide)
      (let* ((colors '("#f6f396" "#c6c6c6")) ; reverse
        (steps (split-string (elt slide 1) ";"))
        (step-length 0.5))

```

```

        (string-join
          (seq-map-indexed
            (lambda (step-ids i)
              (format "%s { fill: %s; opacity: 1 !important; ↵
↵ transition: fill %.1fs; transition-delay: %.1fs }"
              (mapconcat
                (lambda (id) (format ↵
                  (split-string step-ids ",")
                  ", ")
                (elt colors (- (length ↵
                  step-length
                  (* i 0.5))))
              steps))))

;;###autoload
(defun sachareveal-svg-progression-css (map-progression &optional highlight-duration)
  "Make the CSS.
map-progression should be a list of lists with the following format:
((\"slide-id\" \"prev1,prev2;curl\" \"id-to-add1,id-to-add2\") ... )."
  (setq highlight-duration (or highlight-duration 2))
  (let (full)
    (format
      "<style>%s</style>"
      (mapconcat
        (lambda (slide)
          (setq full (append (split-string (elt slide 2) ",") full))
          (format "#slide-%s.present path { opacity: 0.2 }
%s { opacity: 1 !important }
%s"
              (car slide)
              (mapconcat (lambda (id) ↵
                ↵
                ↵
                ↵
              (sachareveal-svg-highlight-different-colors slide)))
              map-progression
              "\n")))))

```

9.3.6 Finding sketches

image

```

(defvar sachasketch-directories
  '("~/sync/sketches"
    "~/sync/private-sketches"))

;;###autoload
(defun sachaget-sketch-filenames-between-dates (start end &optional filter)
  "Returns index card filenames between START and END."
  (setq start (replace-regexp-in-string "[^0-9]" "" start))
  (setq end (replace-regexp-in-string "[^0-9]" "" end))
  (sachaget-sketch-filenames
    (lambda (filename)
      (let ((f (replace-regexp-in-string "[^0-9]" "" (file-name-nondirectory filename))))

```

```

    (and (string> f start)
          (string> end f)
          (or (not filter) (string-match filter filename))))))

;;;###autoload
(defun sacha-get-sketch-filenames (base &optional as-regex)
  (sacha-get-image-filenames base as-regex sacha-sketch-directories))
;;;###autoload
(defun sacha-get-image-filenames (base &optional as-regex directories)
  "Check several directories for files matching BASE.
  Return the matching filenames, if any.
  If AS-REGEXP is non-nil, treat BASE as a regular expression.
  If BASE is a function, use that to filter."
  (when (and (stringp base) (string-match <←
    → "^([0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]" base))
    (setq base (match-string 0 base)))
  (let ((base-regex (unless (functionp base)
    (concat
      "\\("
      (if as-regex base (regexp-quote base))
      "\\)"
      ".*\\(\\.\\(png\\|psd\\|tiff\\|jpe?g\\|svg\\)\\)\\)$"))))
    (-filter
      (lambda (o) (not (string-match "\\\\.xmp" o)))
      (sort (-flatten
        (delq nil
          (mapcar
            (lambda (dir)
              (and (file-directory-p dir)
                    (if (functionp base)
                      (-filter base (directory-files dir t <←
    → ".*\\.\\(png\\|psd\\|tiff\\|jpe?g\\|svg\\)?\\)$"))
                    (directory-files
                      dir t
                      base-regex))))
              (or directories sacha-image-directories))))
          'string<))))))

;;;###autoload
(defun sacha-image-filename (base &optional as-regex directories)
  "Check several directories for files matching BASE.
  Return the first matching filename, if any.
  If AS-REGEXP is non-nil, treat BASE as a regular expression."
  (when (and (listp base) (alist-get 'source_path base))
    (setq base (file-name-base (alist-get 'source_path base))))
  (if (file-exists-p base)
      base
      (car (sacha-get-image-filenames base as-regex directories))))
;;;###autoload
(defun sacha-sketch-filename (base &optional as-regex)
  (sacha-image-filename base as-regex sacha-sketch-directories))
(defalias 'sacha-get-image-filename 'sacha-image-filename)
(defalias 'sacha-get-sketch-filename 'sacha-sketch-filename)

;;;###autoload
(defun sacha-list-sketches (regexp &optional full-filename directories)
  "Return a list of sketch filenames matching REGEXP."
  (interactive (list (read-string "Filter: ")))
  (let ((sacha-sketch-directories (or directories sacha-sketch-directories)))

```

```
(funcall (if (called-interactively-p 'interactive)
  (lambda (x) (insert (mapconcat (lambda (y) (concat "- " (org-link-make-string
↳ (concat "sketchLink:" y)))) x "\n")))) 'identity)
  (sort (-uniq
    (mapcar (if full-filename 'identity
      'file-name-nondirectory)
      (sacha-get-sketch-filenames regexp t)))
    'string>))))
```

9.3.7 Renaming and recoloring sketches

image

[2024-02-07 Wed]

```
;;###autoload
(defun sacha-sketch-rename (file)
  (interactive "FFile: ")
  (sacha-image-rename-set
   file
   (sacha-image-recognize-get-new-filename file)))

;;###autoload
(defun sacha-sketch-recolor-png (file &optional color-scheme)
  (interactive (list (read-file-name "File: ")
↳ (completing-read "Scheme: " (mapcar (lambda (o) (symbol-name (car o)))
↳
↳
↳ sacha-sketch-color-map))))
  (setq color-scheme (or color-scheme 't))
  (call-process "/home/sacha/bin/recolor.py" nil nil nil
    "--colors"
    (mapconcat
      (lambda (row)
        (concat (car row) ","
↳
↳ (cdr row)))
    (assoc-default (if (stringp
↳
↳ color-scheme)
    (intern color-scheme)
    color-scheme)
    sacha-sketch-color-map)
    ",")
    (expand-file-name file))
  file)
(defalias 'sacha-image-recolor 'sacha-sketch-recolor-png)
```

9.3.8 TODO Interactively recolor a sketch

I wanted to be able to change the colours used in a sketch, all from Emacs. For this, I can reuse my [Python script for analyzing colours and changing them](#) and just add some Emacs Lisp to pick colours from Emacs.

```
(defvar sacha-recolor-command "/home/sacha/bin/recolor.py")
```

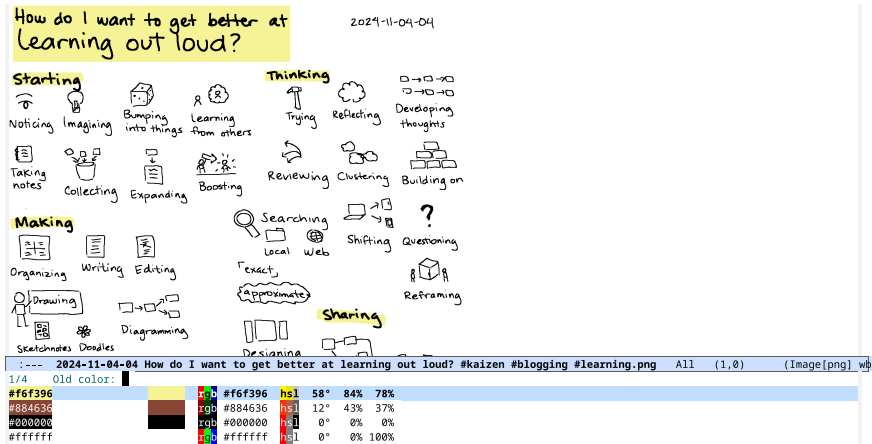


Figure 15: Selecting the colour to replace

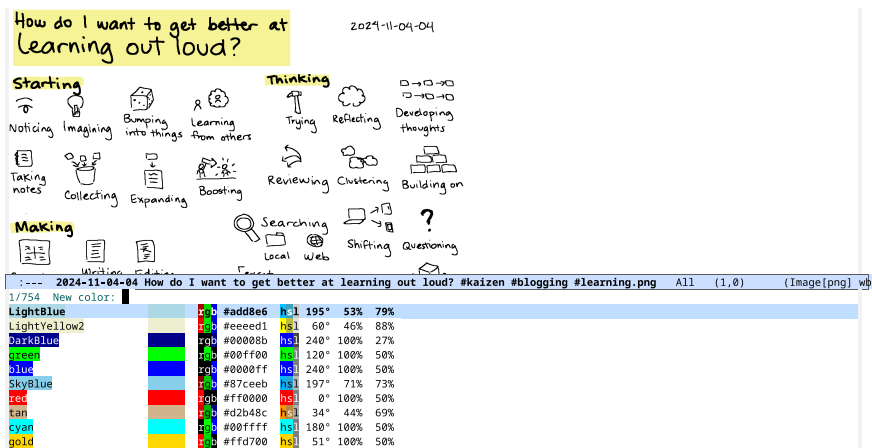


Figure 16: Selecting the new colour

```

;;;###autoload
(defun sach-image-colors-by-frequency (file)
  "Return the colors in FILE."
  (with-temp-buffer
    (call-process sach-recolor-command nil t nil (expand-file-name file))
    (goto-char (point-min))
    (delete-region (point-min) (1+ (line-end-position)))
    (mapcar (lambda (o) (concat "#" (car (split-string o "[ \t]"))))
      (split-string (string-trim (buffer-string)) ←
        ↪ "\n"))))

;;;###autoload
(defun sach-completing-read-color (prompt list)
  "Display PROMPT and select a color from LIST."
  (completing-read
    (or prompt "Color: ")
    (mapcar (lambda (o)
              (faces--string-with-color o o))
      list)))

;;;###autoload
(defun sach-image-recolor-interactively (file)
  (interactive (list (read-file-name "File: " (concat sach-sketches-directory "/") nil t
    ↪
    ↪ nil
    ↪
    ↪ (lambda (file) (string-match ↪
    ↪ "\\\\.png\\\\" file)))))
  (save-window-excursion
    (find-file file)

    ;; Identify the colors by frequency
    (let (choice done)
      (while (not done)
        (let* ((by-freq (sach-image-colors-by-frequency file))
              (old-color ↪
    ↪ (sach-completing-read-color "Old color: " by-freq))
              (new-color (read-color "New color: " t))
              (temp-file (make-temp-file "recolor" ↪
    ↪ nil (concat "." (file-name-extension file))))
              color-map)
          (when (string-match "#\\((..\\)\\.\\.\\((..\\)\\.\\.\\((..\\)\\.\\.)" ↪
    ↪ new-color)
              (setq new-color (concat (match-string 1 ↪
    ↪ new-color)
    ↪
    ↪ (match-string 2 new-color)
    ↪
    ↪ (match-string 3 new-color))))
            (setq color-map (replace-regexp-in-string "#" "" ↪
    ↪ (concat old-color "," new-color)))
            (call-process sach-recolor-command nil nil nil
    ↪
    ↪ (expand-file-name file)
    ↪
    ↪ "--colors"
    ↪
    ↪ color-map
    ↪

```

```

↪      "--output" temp-file)
      (find-file temp-file)
      (pcase (read-char-choice "(y)es, (m)ore, (r)edo, ↪
↪ (c)ancel: " "yrc")
          (?y
            (kill-buffer)
            (rename-file temp-file file t)
            (setq done t))
          (?m
            (kill-buffer)
            (rename-file temp-file file t))
          (?r
            (kill-buffer)
            (delete-file temp-file))
          (?c
            (kill-buffer)
            (delete-file temp-file)
            (setq done t))))))

```

It would be nice to update the preview as I selected things in the completion, which I might be able to do by making a `consult-read` command for it. It would be extra cool if I could use this [webkit-based color picker](#). Maybe someday!

9.3.9 Org Mode sketch: links

image

```

;;;###autoload
(defun sacha-open-images-in-krita (files)
  (apply 'call-process "krita" nil 0 nil "--nosplash" files))
;;;###autoload
(defun sacha-open-images-in-gwenview (files)
  (apply 'call-process "gwenview" nil 0 nil "--slideshow" files))
;;;###autoload
(defun sacha-open-images-in-feh (files)
  (apply 'call-process "feh" nil nil nil "-D" "1" "-F" files))
;;;###autoload
(defun sacha-org-image-open (id &optional arg directories)
  "Open image named ID.
  If ARG is specified, prompt for application to open it in."
  (interactive (list
    (completing-read "Sketch ID: " (sacha-list-sketches ".")
      (current-prefix-arg))))
  (let* ((files (mapcar (lambda (o) (sacha-get-image-filename o (or sacha-image-directories)))
    ↪ (if (listp id) id (list id))))
    (input (if arg (read-char "(k)rita, (g)wenview, (f)eh: " ?k)))
    (funcall
      (cond
        ((eq input ?g) 'sacha-open-images-in-gwenview)
        ((eq input ?f) 'sacha-open-images-in-feh)
        (t 'sacha-open-images-in-krita))
      files)))
  files)))
;;;###autoload
(defun sacha-org-sketch-edit (id &optional arg)
  (sacha-org-image-open id arg sacha-sketch-directories))
;;;###autoload
(defun sacha-org-sketch-open (id &optional arg)
  (delete-other-windows)
  (with-selected-window (split-window-right)
    (find-file (sacha-get-image-filename

```

```

id sachachua-directories))))
;; ##autoload
(defun sachachua-org-image-export (link description format info)
  (let* ((path (concat "https://sketches.sachachua.com/filename/" link))
         (image (concat "https://sketches.sachachua.com/static/" link))
         (backend (org-export-backend-name (plist-get info :back-end)))
         (desc (or description link)))
    (cond
      ((eq backend '11ty) (format "{% sketchLink \"%s\", \"%s\" %}" link desc))
      ((or (eq format 'html) (eq format 'wp))
       (if description
         (format "<a target=\"_blank\" href=\"%s\">%s</a>" path desc)
         (format "<div style=\"text-align: center\"><a target=\"_blank\" href=\"%s\"><img ↵
↵ src=\"%s\" style=\"max-height: 90vw; height: auto; width: auto\"><br />%s</a></div>" ↵
↵ path image desc)))
      ((eq format 'latex) (format "\\href{%s}{%s}" path desc))
      ((eq format 'texinfo) (format "@uref{%s,%s}" path desc))
      ((eq format 'md)
       (if (file-exists-p (expand-file-name link "~/sketches"))
         (format "{{<photo src=\"%s\">}}" image)
         (format "{{<photo nas=\"1\" src=\"%s\">}}" link)))
      ((eq format 'ascii) (format "%s <%s>" desc path))
      (t path))))

;; ##autoload
(defun sachachua-org-image-export-link (link description format info)
  (let* ((backend (if (plist-get info :back-end) (org-export-backend-name (plist-get info ↵
↵ :back-end))
                     format))
         (desc (or description link)))
    (cond ((eq backend 'md)
           (format "[%s](%s)" desc link))
          ((eq backend '11ty)
           (format "{% sketchLink \"%s\", \"%s\" %}"
                   (replace-regexp-in-string "\"\" \"\\\\\"" (file-name-base link) nil t)
                   (replace-regexp-in-string "\"\" \"\\\\\"" desc nil t)))
          ((eq backend 'html)
           (format "<a href=\"https://sketches.sachachua.com/filename/%s\">%s</a>" ↵
↵ (file-name-nondirectory link) desc))
          (t (format "[[%s][%s]]" link desc)))))

;; ##autoload
(defun sachachua-org-image-export-thumb (link description format info)
  (let* ((path (concat "https://sketches.sachachua.com/filename/" link))
         (image (concat "https://sketches.sachachua.com/static/" link))
         (backend (org-export-backend-name (plist-get info :back-end)))
         (desc (replace-regexp-in-string "%23" "#" (or description link))))
    (cond
      ((eq backend '11ty) (format "{% sketchThumb \"%s\", \"%s\" %}" (file-name-base link) ↵
↵ desc))
      ((or (eq format 'html) (eq format 'wp))
       (format "<div class=\"sketch-thumbnail\"><a target=\"_blank\" href=\"%s\"><img ↵
↵ src=\"%s\"><br />%s</a></div>" path image desc))
      ((eq format 'latex) (format "\\href{%s}{%s}" path desc))
      ((eq format 'texinfo) (format "@uref{%s,%s}" path desc))
      ((eq format 'md)
       (if (file-exists-p (expand-file-name link "~/sketches"))
         (format "{{<photo src=\"%s\">}}" image)
         (format "{{<photo nas=\"1\" src=\"%s\">}}" link)))
      (t path))))

```



```

((eq format 'ascii) (format "%s <%s>" desc path))
(t path))))

;;;###autoload
(defun sach-org-image-export-full (link description format info)
  (let* ((path (concat "https://sketches.sachachua.com/filename/" link))
         (image (concat "https://sketches.sachachua.com/static/" link))
         (backend (org-export-backend-name (plist-get info :back-end)))
         (desc (or description link)))
    (cond
      ((eq backend 'lily) (format "{%% sketchFull \"%s\", \"%s\" %%}" link desc))
      ((or (eq format 'html) (eq format 'wp))
       (if description
         (format "<figure><a target=\"_blank\" href=\"%s\"><img src=\"%s\" /><br ↵
↵ /></a><figcaption>%s</figcaption></figure>" path image desc)
         (format "<figure><a target=\"_blank\" href=\"%s\"><img src=\"%s\" /><br ↵
↵ /><figcaption>%s</figcaption></a></figure>" path image desc)))
      ((eq format 'latex) (format "\\href{%s}{%s}" path desc))
      ((eq format 'texinfo) (format "@uref{%s,%s}" path desc))
      ((eq format 'md)
       (if (file-exists-p (expand-file-name link "~/sketches"))
         (format "{{<photo src=\"%s\">}}" image)
         (format "{{<photo nas=\"1\" src=\"%s\">}}" link)))
      ((eq format 'ascii) (format "%s <%s>" desc path))
      (t path))))

;;;###autoload
(defun sach-org-sketch-complete (&optional prefix)
  (concat "sketch:" (file-name-nondirectory (sacha-complete-sketch-filename))))

;;;###autoload
(defun sach-org-sketch-complete-full (&optional prefix)
  (concat "sketchFull:" (file-name-nondirectory (sacha-complete-sketch-filename))))

;;;###autoload
(defun sach-org-image-complete (&optional prefix)
  (concat "image:"
    (completing-read "Image: " (sacha-list-sketches "." nil sach-org-image-directories))))

;; Based on https://emacs.stackexchange.com/questions/38098/org-mode-custom-youtube-link-syntax
;;;###autoload
(defun sach-org-sketch-preview (start end path bracketp)
  "Include overlays for sketches."
  (when (display-graphic-p)
    (let ((filename (sacha-get-sketch-filename path))
          (refresh nil)
          (link (save-excursion
                  (goto-char start)
                  (org-element-lineage
                   (save-match-data (org-element-context))
                   '(link) t)))) ; set this someday
      (when (and (not (org-element-property :contents-begin link)) filename)
        (let ((width
              ;; Apply `org-image-actual-width' specifications.
              (cond
                ((not (image-type-available-p 'imagemagick)) nil)
                ((eq org-image-actual-width t) nil)
                ((numberp org-image-actual-width) org-image-actual-width)
                ;; Pick this up from the paragraph someday
                ))
              (old (get-char-property-and-overlay start 'org-image-overlay)))
          (if (and (car-safe old) refresh)
              (refresh)
              (set-char-property-and-overlay start (width) old))))))

```

```

      (image-refresh (overlay-get (cdr old) 'display))
      (let ((image (create-image filename
                                (and width 'imagemagick)
                                nil
                                :width width)))
        (when image
          (let* ((ov (make-overlay start end)))
            (overlay-put ov 'display image)
            (overlay-put ov 'face 'default)
            (overlay-put ov 'org-image-overlay t)
            (overlay-put ov 'evaporate t)
            (overlay-put
             ov 'modification-hooks
             (list 'org-display-inline-remove-overlay))
            (push ov org-inline-image-overlays)))))))))

```

```

(use-package org
  :config
  (setq org-image-actual-width 600)
  (org-link-set-parameters
   "sketch"
   :follow 'sacha-org-sketch-open
   :export 'sacha-org-image-export-link
   :complete 'sacha-org-sketch-complete
   :activate-func nil)
  (org-link-set-parameters
   "sketchLink"
   :follow 'sacha-org-sketch-open
   :export 'sacha-org-image-export-link
   :complete 'sacha-org-sketch-complete
   :activate-func nil)
  (org-link-set-parameters
   "sketchThumb"
   :follow 'sacha-org-sketch-open
   :export 'sacha-org-image-export-thumb
   :complete 'sacha-org-sketch-complete
   :activate-func nil)
  (org-link-set-parameters
   "sketchFull"
   :follow 'sacha-org-sketch-open
   :export 'sacha-org-image-export-full
   :complete 'sacha-org-sketch-complete-full
   :activate-func nil))
  (org-link-set-parameters
   "image"
   :follow 'sacha-org-image-open
   :export 'sacha-org-image-export
   :complete 'sacha-org-image-complete))

```

9.3.10 Org Mode custom link: copy to clipboard

emacs:org

I have a tiny corporation for my consulting. I do all of my own paperwork. I have lots of notes in Org Mode for infrequent tasks like the tax-related paperwork I do once a year. My notes include checklists, links, and Org Babel blocks for calculations. I often need to copy standard text (ex: the name of the company) or parts of the output of my Org Babel blocks (ex: tax collected) so that I can fill in web forms on the Canada Revenue Agency website.

This little snippet makes it easy to copy text for pasting. It defines a custom Org link that starts with `copy:`. When I follow the link by clicking on it or using `C-c C-o` (`org-open-at-point`), it copies the text to the kill ring (which is what Emacs calls the clipboard) so that I can paste it anywhere. For example, `[[copy:Hello world]]` becomes a link to copy "Hello world". Copying means never having to worry about typos or accidentally selecting only part of the text.

```
;;###autoload
(defun sach-org-copy-export (link desc format)
  (pcase format
    ('org (org-link-make-string (concat "copy:" link) desc))
    (_ desc)))
```

```
(use-package org
  :config
  (org-link-set-parameters
   "copy"
   :follow (lambda (link) (kill-new link))
   :export #'(sach-org-copy-export)))
```

I can use these links as part of my checklist so that I can quickly fill in things like my business name and other details. I can put sensitive information like my social insurance number in a GPG-encrypted file. (Just set up your GPG keys and end a filename with `.gpg`, and Emacs will take care of transparently encrypting and decrypting the file.)

I can also export those links as part of my Org Babel output. For example, the following code calculates the numbers I need to fill in a T5 form for the other-than-eligible dividends that I issue myself according to the [T5 instructions from the CRA](#).

```
(let* ((box-10 1234) ; fake number for demo
      (box-11 (* 1.15 box-10))
      (box-12 (* 0.090301 box-11)))
  `((box-10 ,(format "[[copy:%.2f][%.2f]]" box-10 box-10))
    (box-11 ,(format "[[copy:%.2f][%.2f]]" box-11 box-11))
    (box-12 ,(format "[[copy:%.2f][%.2f]]" box-12 box-12))))
```

```
box-10  1234.00
box-11  1419.10
box-12   128.15
```

`copy:Hello world`

Hello world On my computer, the numbers become links that I can click and copy. Another little shortcut thanks to Emacs and Org Mode!

9.3.11 Helm completion with `sacha-helm-org-sketches`

```
;;###autoload
(defun sach-helm-source-org-sketch-list ()
  (sacha-list-sketches "."))

;;###autoload
(defun sach-helm-org-insert-sketch-candidates (&optional candidates)
  (mapc (lambda (o)
    (org-insert-link nil (concat "sketch:" o))
    (insert "\n"))
    (helm-marked-candidates)))
```



```

(sacha-org-sketch-open (sacha-prepare-index-card-template nil ←
↪ (org-read-date))))
      "Date")
  (widget-create 'push-button
    :notify (lambda (&rest ignore) (shell-command "~/bin/rotate-screen")) ←
↪ "Rotate")
  (insert "\n")
  (widget-create 'push-button
    :notify (lambda (&rest ignore)
      (shell-command "~/bin/add-output-png"))
    "Add output.png")
  (widget-create 'push-button
    :notify (lambda (&rest ignore)
      (sacha-rotate-screen 0)
      (kill-buffer)
      (sacha-rename-scanned-cards))
    "Process")
  (widget-create 'push-button
    :notify (lambda (&rest ignore)
      (sacha-rotate-screen 0)
      (delete-window)
      (sacha-rename-scanned-cards))
    "Rename")
  (widget-create 'push-button
    :notify (lambda (&rest ignore)
      (sacha-rotate-screen 0)
      (delete-window)
      (sacha-convert-and-upload-cards))
    "Upload")
  (widget-create 'push-button
    :notify (lambda (&rest ignore)
      (sacha-rotate-screen 0)
      (org-clock-out)
      (kill-buffer))
    "Quit")
  (text-scale-set 10)
  (widget-setup)
  (widget-minor-mode)
  (pop-to-buffer (current-buffer))
  (goto-char (point-min))
  (current-buffer))))

;; ###autoload
(defun sacha-prepare-index-cards (n)
  (interactive (list (or current-prefix-arg 5)))
  (let ((counter 1)
        (directory "~/Dropbox/Inbox")
        (template sacha-index-card-template-file)
        (date (substring (org-read-date nil nil ".") 0 10))
        temp-file)
    (quantified-track "Drawing")
    (dotimes (i 5) (sacha-org-sketch-open (sacha-prepare-index-card-template)))
    (sacha-rotate-screen 180)
    (sacha-set-up-sketch-buffer)))

(defun sacha-index-card-file-name nil "Most recent index card file name.")
;; ###autoload
(defun sacha-rotate-screen (degrees)
  (cond

```

```

((eq system-type 'windows-nt)
 (shell-command (format "c:/sacha/Dropbox/bin/orient /rotate:%d" degrees)))
((eq system-type 'gnu/linux)
 (shell-command (format "~/bin/rotate-screen %s"
                      (cond
                        ((= degrees 0) "normal")
                        ((= degrees 180) "inverted")
                        ((= degrees 90) "left")
                        ((= degrees 270) "right"))))))

```

```

(setq sacha-sketch-executable "krita"
      sacha-sketch-inbox-directory "~/Dropbox/Inbox"
      sacha-index-card-template-file "~/Dropbox/drawings/templates/0 - index.psd"
      sacha-sketch-large-template-file "/home/sacha/Dropbox/drawings/templates/0 - base.psd")

```

9.3.13 Templates

image

```

;;;###autoload
(defun sacha-prepare-drawing-template (&optional name date template)
  "Create the image file for NAME. Return the new filename."
  (let* ((date (or date (substring (org-read-date nil nil ".") 0 10)))
         (data (sacha-journal-post (or name "sketch") :Date date)))
    (setq name (expand-file-name
                  (concat (assoc-default 'ZIDString data)
                          (if name
                              (concat " "
                                      (sacha-convert-sketch-title-to-filename (or name "")))
                              ""))
                  "." (file-name-extension template))
          "~/Dropbox/Inbox"))
    (copy-file (or template sacha-index-card-template-file) name)
    name))

;;;###autoload
(defun sacha-org-insert-new-index-card-link ()
  (interactive)
  (let ((filename
        (sacha-prepare-index-card-template)))
    (insert "[[sketch:" filename "]]\n")
    (save-window-excursion
      (sacha-rotate-screen 180)
      (shell-command
       (concat (shell-quote-argument sacha-sketch-executable)
               " " (shell-quote-argument filename) " &")))))

;;;###autoload
(defun sacha-prepare-index-card-template (&optional name date)
  "Create the image file for NAME. Return the new filename."
  (sacha-prepare-drawing-template name date sacha-index-card-template-file))

;;;###autoload
(defun sacha-prepare-large-template (&optional name date)
  "Create the image file for NAME. Return the new filename."
  (sacha-prepare-drawing-template name date sacha-sketch-large-template-file))

```



```

(dot-spacing 60)
(grid-color "#a6d2ff")
(row-size 6)
(col-size 6)

↩↪
↩↪
↩↪ (text-size 50)

(margin-top 2))

"Prepare an SVG with a dot grid within a table with solid gridlines.
Each dot is a solid circle of DOT-SIZE filled with GRID-COLOR spaced DOT-SPACING apart.
The gridlines are also GRID-COLOR. They should divide the image into ROWS and COLUMNS, which ↩↪
↩↪ are ROW-SIZE * DOT-SPACING and COL-SIZE * DOT-SPACING apart.
The table has a top margin with the dot grid, and this is MARGIN-TOP * DOT-SPACING tall.
All dots are centered on their x, y coordinates.
The rest of the image's background is white."

(let* ((width (* num-cols col-size dot-spacing))
      (height (* dot-spacing (+ margin-top (* num-rows row-size))))
      (margin-top-height (* margin-top dot-spacing))
      (svg (svg-create width height)))
  (dotimes (row (+ (* num-rows row-size) margin-top))
    (dotimes (col (1+ (* num-cols col-size)))
      (let ((x (* col dot-spacing))
            (y (* row dot-spacing)))
        (svg-circle svg x y dot-size

↩↪

↩↪ :fill-color grid-color

↩↪ :stroke-width 0))))
  (when (> text-size 0)
    (dotimes (i (* num-rows num-cols))
      (let ((x (* (% i num-cols) col-size dot-spacing))
            (y (+ margin-top-height (* (/ i num-cols) row-size dot-spacing))))
        (svg-text svg

↩↪

↩↪ (number-to-string (1+ i))

↩↪ :x x :y (+ y ↩↪

↩↪ :fill-color ↩↪

↩↪ :font-size ↩↪

↩↪ :stroke-width ↩↪

↩↪ 0))))
  (dotimes (col (1+ num-cols))
    (let ((x (* col col-size dot-spacing)))
      (svg-line svg x margin-top-height x height

↩↪ :stroke-color grid-color
↩↪ :stroke-width ↩↪

↩↪ line-width)))
  (dotimes (row (1+ num-rows))
    (let ((y (+ margin-top-height (* row row-size dot-spacing))))
      (svg-line svg 0 y width y

↩↪ :stroke-color grid-color
↩↪ :stroke-width ↩↪

↩↪ line-width)))
  svg))

```



```
(svg-print
  (sacha-dot-grid-boxes-template)))
```

```
;;###autoload
(cl-defun sacha-dot-grid-boxes-list (&key (num-rows 5)

  ↪                                     (num-cols 7) ↪
  ↪                                     (dot-spacing 60) ↪
  ↪                                     (row-size 6) ↪
  ↪                                     (col-size 6) ↪
  ↪                                     (text-bottom 1) ↪
  ↪                                     (margin-top 2) ↪
  ↪                                     filename ↪
  ↪                                     &allow-other-keys) ↪
  "Return a list of boxes."
  (let* ((margin-top-height (* margin-top dot-spacing))
         (max-image-size nil)
         (size (image-size (create-image filename nil nil :scale 1) t))
         (ratio (/ (car size) (* num-cols col-size dot-spacing 1.0)))
         results)
    (message "Expected adjusted height %f actual height %f"
             (* (+ margin-top (* num-rows row-size))
               ↪
             dot-spacing ratio)
             (cdr size))
    (dotimes (i (* num-rows num-cols))
      (let* ((r (/ i num-cols))
             (c (% i num-cols))
             (y (* (+ margin-top-height (* r col-size ↪
             dot-spacing)) ratio))
             (x (* c row-size dot-spacing ratio))
             (width (* col-size dot-spacing ratio))
             (height (* (- row-size text-bottom) ↪
             dot-spacing ratio)))
        (setq results (cons
                       ↪
                       . ,r)
                       ↪
                       (c . ,c)
                       ↪
                       (i . ,i)
                       ↪
                       (x . ,(floor x))
                       ↪
                       (y . ,(floor y))
                       ↪
                       (w . ,(floor width))
                       ↪
                       (h . ,(floor height))
                       ↪
                       (x2 . ,(floor (+ x width)))
                       ↪
```

```

→      (y2 . ,(floor (+ y height))))
→ results))))
→      (nreverse results)))

(defvar sacha-sketch-icon-directory "~/sync/sketches/icons")
;;###autoload
(cl-defun sacha-dot-grid-boxes-extract (&rest args &key filename labels
→
→
→ (output-dir sacha-sketch-icon-directory) force &allow-other-keys)
→ (let* ((list (apply #'sacha-dot-grid-boxes-list args))
→          (base (file-name-base filename))
→          (ext (concat "." (file-name-extension filename)))
→          (id
→            (if (string-match
→                "[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]" base)
→                (match-string 0 base)
→                ""))
→          results
→          args)
→ (dolist (icon list)
→ (let-alist icon
→ (let ((new-filename (expand-file-name
→ (concat (sacha-make-slug (elt labels .i)) "--"
→ id
→ (format "-%d-%d"
→ .r .c)
→ ext)
→ output-dir)))
→ (push `((term . ,(elt labels .i))
→ (icon .
→ ,(file-name-nondirectory new-filename))
→ (source .
→ ,(file-name-nondirectory filename)))
→ results)
→ (when (or force (not (file-exists-p new-filename)))
→ (setq args
→ (list
→ (expand-file-name filename)
→ "-crop"
→ (format "%dx%d+%d+%d" .w .h .x .y)
→ "+repage"
→ new-filename))
→ (message "%s" (concat "convert " (mapconcat
→ #'shell-quote-argument args " " ))))
→ (apply #'call-process "convert" nil nil nil
→ args))))))

```



```

↪                                     :num-rows (/ (length labels) 7)
↪                                     ↪
↪                                     :filename ↪
↪ (sacha-get-sketch-filename id)
↪                                     ↪
↪                                     :labels labels
↪                                     ↪
↪                                     :force force))
      (sacha-sketch-icon-update-index list)))

;;;###autoload
(defun sacha-dot-grid-boxes-process-all-icons ()
  (interactive)
  (dolist (source (sacha-sketches "icons")) (sacha-dot-grid-boxes-process source)))

```

9.3.15 Easily backfill my journal

```

;;;###autoload
(defun sacha-draw-journal-entry (date)
  "Creates a blank journal entry for DATE and brings up the log."
  (interactive (list (org-read-date)))
  ;; Open the Quantified Awesome time log for that date
  (let ((filename (sacha-get-journal-entry date))
        (day (format-time-string "%A" (org-time-string-to-time date))))
    (if filename
        (sacha-org-sketch-open filename)
        ;; (browse-url (format "http://quantifiedawesome.com/records?start=%s&end=%s"
        ;;                   date
        ;;                   (format-time-string
        ;;                     "%Y-%m-%d"
        ;;                     (seconds-to-time
        ;;                       (+ (org-time-string-to-seconds date) 86400))))))
        (setq filename
              (sacha-prepare-index-card-template (concat day " #daily #journal") date))
        (sacha-org-sketch-open filename))))

;;;###autoload
(defun sacha-get-journal-entry (date)
  "Returns the filename for the journal sketch for DATE."
  (car
   (-filter (lambda (x) (not (string-match "weekly" x)))
            (sacha-get-sketch-filenames
             (format "%s.*.*#daily" date)
             t))))

;;;###autoload
(defun sacha-get-missing-journal-dates (start-date end-date)
  "Return a list of dates missing journal entries.
Range is specified by START-DATE (inclusive) and END-DATE (exclusive)."
  (let* ((current-day (org-time-string-to-absolute end-date))
        (start-day (org-time-string-to-absolute start-date))
        current-date
        current-date-string
        missing-list)
    (while (≥ current-day start-day)
      (setq current-date (calendar-gregorian-from-absolute current-day))

```

```

    (setq current-date-string (format "%04d-%02d-%02d" (elt current-date 2) (elt ↵
    ↵ current-date 0) (elt current-date 1)))
    (unless (sacha-get-journal-entry current-date-string)
      (add-to-list 'missing-list current-date-string))
    (setq current-day (1- current-day)))
    missing-list))

;;;###autoload
(defun sacha-show-missing-journal-entries (since)
  (interactive (list (if current-prefix-arg (org-read-date) (org-read-date nil nil "-7"))))
  (let ((missing-dates (sacha-get-missing-journal-dates since (org-read-date nil nil "."))))
    (with-current-buffer (sacha-set-up-sketch-buffer)
      (mapc
        (lambda (date)
          (widget-create 'push-button
            :date date
            :notify (lambda (widget &rest ignore)
                      (sacha-draw-journal-entry (plist-get (cdr widget) :date)))
            date))
          missing-dates)
        (widget-setup)
        (widget-minor-mode))))

```

9.3.16 Rename scanned index cards

```
(use-package s)
```

```

;;;###autoload
(defun sacha-process-tiff (files)
  "Convert, display, rename, and upload FILES."
  (interactive (list (dired-get-marked-files)))
  (unless (listp files) (setq files (list files)))
  (save-window-excursion
    (apply 'call-process "mogrify" nil nil nil (append (list "-format" "png" "-quality" "1") ↵
    ↵ files))
    (delete-other-windows)
    (setq files
      (mapcar
        (lambda (filename)
          (find-file (setq filename (s-append ".png" (s-chop-suffix ".tif" filename)))))
        (let ((new-name
              (read-string "New name: "
                (concat
                  (if (string-match "/\\((\\([0-9]+-[0-9]+-[0-9]+)\\)\\)\\( ↵
    ↵ ?.*)\\)?\\)\\.png" filename)
                  (match-string 1 filename)
                  filename)
                " "))))
          (rename-file filename (concat new-name ".png"))
          (setq filename (expand-file-name (concat new-name ".png") (file-name-directory ↵
    ↵ filename))))))
    files)))
  (find-file "~/Dropbox/Public/sharing/index.org")
  (goto-char (point-min))
  (when (re-search-forward (regexp-quote "#+ORGLST: sketchinbox"))
    (forward-line 1)

```

```

(org-end-of-item-list)
(apply 'call-process "up" nil t nil files)))

;;;###autoload
(defun sacha-convert-index-card-to-png (o)
  (lambda (o)
    (call-process "krita" nil nil nil o "--export" "--export-filename"
      (concat (file-name-sans-extension o) ".png"))
    (rename-file o "~/Dropbox/Inbox/backup/" t)))

;;;###autoload
(defun sacha-convert-index-card-tiffs-to-pngs ()
  (interactive)
  (let ((pattern "^\\(IMG\\|\\[0-9]+-[0-9]+-[0-9]+\\)\\.\\.\\(tif\\|psd\\)$"))
    (when (directory-files "~/Dropbox/Inbox/" t pattern)
      ;; Convert the TIFFs first
      (mapc 'sacha-convert-index-card-to-png
        (directory-files "~/Dropbox/Inbox/" t pattern)))))

;;;###autoload
(defun sacha-convert-and-upload-cards ()
  "Trust in existing filenames, upload without modification."
  (interactive)
  (sacha-convert-index-card-tiffs-to-pngs)
  (sacha-upload-scanned-cards))

;;;###autoload
(defun sacha-rename-scanned-card (filename)
  (find-file filename)
  (delete-other-windows)
  (let ((base (file-name-sans-extension filename))
        notes)
    (when (string-match "/IMG.*\\|\\(\\[0-9]+-[0-9]+-[0-9]+\\)\\.\\.\\(.*\\)?\\)" base)
      (let ((kill-buffer-query-functions nil)
            old-name
            (new-name (read-string "New name: "
                                  (if (match-string 1 base)
                                      (concat (match-string 1 base))
                                      ""))))
        (while (and (string-match "[0-9]+-[0-9]+-[0-9]+[a-z]" new-name)
                     (setq old-name (sacha-get-sketch-filename (match-string 0 new-name)))
                     (and old-name
                          (not (string= old-name filename))
                          (not (string= (file-name-nondirectory old-name)
                                         (concat (s-trim new-name) "." (file-name-extension ↵
↵ filename))))))
          (setq new-name
                (read-string (format "Already exists (%s) - new name: " old-name)
                           new-name)))
        (when (string-match new-name "^\\(.*?\\) *| *\\(.*?\\)")
          (with-current-buffer (find-file "~/sync/orgzly/Inbox.org")
            (goto-char (point-max))
            (insert "\n* " (match-string 1 new-name) "\n" (match-string 2 new-name))
            (save-buffer))
          (setq new-name (match-string 1 new-name)))
        (when (> (length new-name) 0)
          (revert-buffer t t)
          (rename-file filename (concat (s-trim new-name) "." (file-name-extension filename)) t)
          (kill-buffer))))))

```

```

;;;###autoload
(defun sacha-rename-scanned-cards ()
  "Display and rename the scanned or saved files."
  (interactive)
  (sacha-convert-index-card-tiffs-to-pngs)
  (mapc (lambda (o)
    (when (string= (file-name-extension o) "psd")
      (sacha-convert-index-card-to-png o)
      (setq o (concat (file-name-sans-extension o) ".png"))))
    (sacha-rename-scanned-card o))
    (reverse (directory-files "~/Dropbox/Inbox/" t ↵
      ↵ "^(\\(IMG\\|\\[0-9]+-\\[0-9]+-\\[0-9]+\\)\\.\\(psd\\|png\\|jpg\\)"))))
  (sacha-upload-scanned-cards))

;;;###autoload
(defun sacha-clean-index-card-directory ()
  "Remove files marked for deletion and move private files."
  (shell-command "mv ~/Dropbox/Inbox/*delete* ~/Dropbox/Inbox/backup")
  (shell-command "mv ~/Dropbox/Inbox/*private* ~/cloud/private-sketches/"))

;;;###autoload
(defun sacha-upload-scanned-cards ()
  (interactive)
  (sacha-clean-index-card-directory)
  (with-current-buffer (get-buffer-create "*Files to be uploaded*")
    (erase-buffer)
    (insert (mapconcat 'identity (directory-files "~/Dropbox/Inbox" nil ↵
      ↵ "[0-9]+-[0-9]+-[0-9]+[^\ ]? .*\\.\\(png\\|jpg\\)") "\n"))
    (goto-char (point-min))
    (switch-to-buffer (current-buffer))
    (delete-other-windows))
  (shell-command "~/bin/copy-sketches"))

```

I might tweak the files a little more after I rename them, so I don't automatically upload them. When I'm happy with the files, I use a [Node script](#) to upload the files to Flickr, move them to my [To blog](#) directory, and copy Org-formatted text that I can paste into my learning outline.

9.3.17 Automatically resize images

The `image+` package is handy for displaying the images so that they're scaled to the window size.

```

(use-package image+
  :if sacha-laptop-p
  ;; :load-path "~/elisp/Emacs-imagex"
  :commands (imagex-global-sticky-mode imagex-auto-adjust-mode)
  :init (progn (imagex-global-sticky-mode) (imagex-auto-adjust-mode)))

```

9.3.18 Get information for sketched books

For sketchnotes of books, I set up the filename based on properties in my Org Mode tree for that book.

```

;;;###autoload
(defun sacha-prepare-sketchnote-file ()
  (interactive)
  (let* ((base-name (org-entry-get-with-inheritance "BASENAME"))))

```

```
(unless base-name (error "Missing basename property"))
(sacha-org-sketch-open (sacha-prepare-large-template base-name)))
```

By using Emacs Lisp functions to set up files that I'm going to use in an external application, I minimize fussing about with the keyboard while still being able to take advantage of structured information.

Do you work with external applications? Where does it make sense to use Emacs Lisp to make setup or processing easier?

9.3.19 Make it easy to follow up on a sketch

```
;; ###autoload
(defun sacha-follow-up-on-sketch (filename)
  "Prompt for FILENAME to follow up on.
  Create an index card with it as a layer, and add the ref to the filename."
  (interactive (list (helm-read-file-name "Image: " :initial-input "~/sketches/")))
  ;; Allow the specification of a short identifier
  (unless (file-exists-p filename)
    (setq filename (car (directory-files "~/sketches" t (concat "^" filename)))))
  (let ((async-shell-command-buffer 'new-buffer)
        (index-card (sacha-prepare-index-card-template
                      (format "-- index card ref %s"
                              (and (string-match "[^\\.]+" (file-name-nondirectory filename))
                                   (match-string 0 (file-name-nondirectory filename))))))
    (shell-command (format "convert %s %s -colorspace cmyk %s"
                          (shell-quote-argument (expand-file-name ↵
↵ sacha-index-card-template-file))
                          (shell-quote-argument (expand-file-name filename))
                          (shell-quote-argument (expand-file-name index-card))))
    (shell-command (format "%s %s &"
                          (shell-quote-argument sacha-sketch-executable)
                          (shell-quote-argument (expand-file-name index-card))))
    (sacha-rotate-screen 180)
    (sacha-set-up-sketch-buffer)))
```

9.3.20 Digital index piles with Emacs

Somewhat daunted by the prospect of categorizing more than a hundred sketches and blog posts for my monthly review, I spent some time figuring out how to create the digital equivalent of sorting index cards into various piles.

2015-02-01 [Digital piles of index cards – index card #indexing #organization #pkm](#)

In fact, wouldn't it be super-cool if the items could automatically guess which category they should probably go in, prompting me only if it wasn't clear?

I wanted to write a function that could take a list structured like this:

- Keyword A
 - Previous links
- Keyword B
 - Previous links
- Link 1 with Keyword A
- Link 2 with Keyword B

- Link 3 with Keyword A
- Link 4

It should file Link 1 and 3 under Keyword A, Link 2 under Keyword B, and prompt me for the category for Link 4. At that prompt, I should be able to select Keyword A or Keyword B, or specify a new category.

Inspired by John Kitchin's recent post on [defining a Helm source](#), I wanted to get it to work with Helm.

First step: I needed to figure out the structure of the list, maybe including a sample from the category to make it clearer what's included. `org-list.el` seemed to have useful functions for this. `org-list-struct` gave me the structure of the current list. Let's say that a category is anything whose text does not match `org-link-bracket-re`.

```
;; ##autoload
(defun sach-org-get-list-categories ()
  "Return a list of (category indent matching-regexp sample).
  List categories are items that don't contain links."
  (let ((list (org-list-struct)) last-category results)
    (save-excursion
      (mapc
        (lambda (x)
          (goto-char (car x))
          (let ((current-item
                (buffer-substring-no-properties
                 (+ (point)
                   (elt x 1)
                   (length (elt x 2)))
                 (line-end-position))))
            (if (string-match
                 org-link-bracket-re
                 (buffer-substring-no-properties
                  (point)
                  (line-end-position)))
                ;; Link - update the last category
                (when last-category
                  (if (< (elt x 1) (elt last-category 1))
                      (setq results
                           (cons (append last-category
                                           (list
                                            (match-string-no-properties
                                             3
                                             (buffer-substring-no-properties
                                              (point)
                                              (line-end-position)))))
                                  (cdr results))))
                      (setq last-category nil))
                  ;; Category
                  (setq results
                       (cons
                        (setq last-category
                            (list
                             current-item
                             (elt x 1)
                             (concat "^"
                                     (make-string (elt x 1) ?\ )
                                     (regexp-quote
                                      (concat (elt x 2)
                                               (elt x 3))))
                            nil))))
                        results))))
          (setq last-category nil))
      (setq results
           (cons
            (setq last-category
                (list
                 current-item
                 (elt x 1)
                 (concat "^"
                         (make-string (elt x 1) ?\ )
                         (regexp-quote
                          (concat (elt x 2)
                                  (elt x 3))))
                 nil))))
            results))))
```

```

                                current-item)))
                                "$"))))
                                results))))))
                                list))
(append '("x" 2 "^$" nil) results)))

```

The next step was to write a function that guessed the list category based on the item text, and moved the item there.

```

(defvar sacha-helm-org-list-candidates nil)
;;;###autoload
(defun sacha-helm-org-list-categories-init-candidates ()
  "Return a list of categories from this list in a form ready for Helm."
  (setq sacha-helm-org-list-candidates
    (mapcar (lambda (x)
      (cons (if (elt x 3)
        (format "%s - %s" (car x) (elt x 3))
        (car x))
      x))
    (sacha-org-get-list-categories))))

```

```

;;;###autoload
(defun sacha-org-move-current-item-to-category (category)
  "Move current list item under CATEGORY earlier in the list.
CATEGORY can be a string or a list of the form (text indent regexp).
Point should be on the next line to process, even if a new category
has been inserted."
  (interactive (list (completing-read "Category: " (sacha-org-get-list-categories))))
  (when category
    (let* ((col (current-column))
      (item (point-at-bol))
      (struct (org-list-struct))
      (category-text (if (stringp category) category (elt category 0)))
      (category-indent (if (stringp category) 2 (+ 2 (elt category 1))))
      (category-regexp (if (stringp category) category (elt category 2)))
      (end (elt (car (last struct)) 6))
      (pos (point))
      s)
      (setq s (org-remove-indentation (buffer-substring-no-properties item ←
← (org-list-get-item-end item struct))))
      (save-excursion
        (if (string= category-text "x")
          (org-list-send-item item 'delete struct)
          (goto-char (caar struct))
          (if (re-search-forward (concat "^ *- +" category-regexp) end t)
            (progn
              ;; needs a patch to ol.el to check if stringp
              (org-list-send-item item (point-at-bol) struct)
              (org-move-item-down)
              (org-indent-item))
            (goto-char end)
            (org-list-insert-item
              (point-at-bol)
              struct (org-list-prevs-alist struct))
            (let ((old-struct (copy-tree struct)))
              (org-list-set-ind (point-at-bol) struct 0)
              (org-list-struct-fix-bul struct (org-list-prevs-alist struct))

```

```

(org-list-struct-apply-struct struct old-struct))
(goto-char (point-at-eol))
(insert category-text)
(org-list-send-item item 'end struct)
(org-indent-item)
(org-indent-item))
(recenter))))))

;;###autoload
(defun sacha-org-guess-list-category (&optional categories)
  (interactive)
  (require 'cl-lib)
  (unless categories
    (setq categories
      (sacha-helm-org-list-categories-init-candidates)))
  (let* ((beg (line-beginning-position))
        (end (line-end-position))
        (string (buffer-substring-no-properties beg end))
        (found
          (cl-member string
                     categories
                     :test
                     (lambda (string cat-entry)
                       (unless (string= (car cat-entry) "x")
                         (string-match (regexp-quote (downcase (car cat-entry)))
                                         string))))))
        (when (car found)
          (sacha-org-move-current-item-to-category
            (cdr (car found)))
          t)))

```

After that, I wrote a function that used Helm to prompt me for a category in case it couldn't guess the category. It took me a while to figure out that I needed to use `:init` instead of `:candidates` because I wanted to read information from the buffer before Helm kicked in.

```
(defvar sach-org-browse-link-while-categorizing 'eww-readable
  "Set to nil to skip browsing.")

;;;###autoload

(defun sach-org-guess-uncategorized ()
  "Interactively move linked list items to categories from the list.
  Try to guess categories based on substring matches."
  (interactive)
  (sach-helm-org-list-categories-init-candidates)
  (let ((categories (sach-org-get-list-categories))
        category)
    (while (and (looking-at "[^+-] \\[\\[\\[([^\]]+\\)\\]\\[\\[\\([^\]]+\\)*\\]\\]")
                (not (string= "done" category)))
      (save-excursion
        (when (eq sach-org-browse-link-while-categorizing 'eww-readable)
          (save-excursion (save-match-data (sach-eww-browse-readable (match-string 1))))
          (setq category (completing-read (match-string 2) categories))
          (unless (string= category "done")
            (sach-org-move-current-item-to-category category))))))

; From https://emacs.stackexchange.com/questions/36284/how-to-open-eww-in-readable-mode/47757
;;;###autoload
(defun sach-eww-readable-nonce ())
```


I think I prefer Krita because I'm more used to cropping with it. Xournal++ uses vectors and makes it easier to insert space when drawing notebook-style stuff, so that might be interesting too. I probably want to write something that converts it to PNG for export.

9.3.22 Sketched books

Convenience functions to make my life easier when sketchnoting books.

```
(setq yas-indent-line 'fixed)
```

```
;;;###autoload
(defun sachu-convert-sketch-title-to-filename (text)
  (setq text (replace-regexp-in-string "[?!]" "" text))
  (setq text (replace-regexp-in-string "[!:] " " - " text)))
(ert-deftest sachu-convert-sketch-title-to-filename ()
  (should (string= (sachu-convert-sketch-title-to-filename "Test") "Test"))
  (should (string= (sachu-convert-sketch-title-to-filename "Another Test!") "Another Test"))
  (should (string= (sachu-convert-sketch-title-to-filename "Does this work? Yes") "Does this ↵
    ↵ work - Yes")))
  (should (string= (sachu-convert-sketch-title-to-filename "Title: Subtitle") "Title - ↵
    ↵ Subtitle")))
)

;;;###autoload
(defun sachu-convert-sketched-book-to-png ()
  "Convert TIFF to PNG."
  (interactive)
  (let ((basename (org-entry-get-with-inheritance "BASENAME")))
    (shell-command (format "convert \"c:/sachu/dropbox/inbox/%s.tif\" ↵
      ↵ \"c:/sachu/dropbox/inbox/%s.png\""
        basename
        basename))))
)

;;;###autoload
(defun sachu-index-sketched-book ()
  "Add entries to sketched books index."
  (interactive)
  (let* ((title (org-entry-get-with-inheritance "SHORT_TITLE"))
        (author (org-entry-get-with-inheritance "AUTHOR"))
        (basename (org-entry-get-with-inheritance "BASENAME"))
        (base-file (format "~/Dropbox/Inbox/%s.png" basename)))
    (when (file-exists-p base-file)
      (copy-file base-file
        (format "~/Dropbox/Packaging/sketched-books/%s.png" basename) t t))
    (find-file "~/Dropbox/Packaging/sketched-books/index.org")
    (vc-git-register (list (format "%s.png" basename)))
    (goto-char (point-min))
    (re-search-forward "<<insert-point>>")
    (insert (format "\n- [[file:%s.png]][%s - %s (sketched %s)]\n [[file:%s.png]]\n\n"
      basename
      title
      author
      (substring basename 0 10)
      basename)))
    (find-file "~/Dropbox/Packaging/sketched-books/ebook.org")
    (goto-char (point-min))
    (re-search-forward "<<insert-point>>"))
)
```

```

(insert (format "\n* %s - %s (sketched %s)\n\n[[file:%s.png]]\n\n"
              title
              author
              (substring basename 0 10)
              basename)))

;;;###autoload
(defun sacha-package-sketched-book ()
  "Add the latest sketch and package the collection."
  (interactive)
  (shell-command
   (format "plink -A vagrant@127.0.0.1 -P 2222 \"cd ~/Dropbox/Packaging/sketched-books; git ←
↪ add '%s.png'; git commit -m 'Added %s - %s' -a; git push; make all\" &"
           (org-entry-get-with-inheritance "BASENAME")
           (org-entry-get-with-inheritance "SHORT_TITLE")
           (org-entry-get-with-inheritance "AUTHOR"))))

```

9.3.23 Other sketches

Based on <http://williamedwardscoder.tumblr.com/post/84505278488/making-image-mosaics>
Aspect ratio is width / height

```

;;;###autoload
(defun sacha-get-tile-dimensions (num-items orig-width orig-height target-aspect-ratio)
  (let ((rows 1) (cols 1)
        (current-aspect (/ orig-width (float orig-height)))
        add-col-aspect
        add-row-aspect)
    (while (< (* rows cols) num-items)
      (setq add-col-aspect (/ (* (1+ cols) (float orig-width))
                              (* rows orig-height))
            add-row-aspect (/ (* cols (float orig-width))
                              (* (1+ rows) orig-height)))
      (if (< (abs (- add-col-aspect target-aspect-ratio))
            (abs (- add-row-aspect target-aspect-ratio)))
          (setq cols (1+ cols))
          (setq rows (1+ rows))))
    (cons cols rows)))
(ert-deftest sacha-get-tile-dimensions ()
  (should (equal (sacha-get-tile-dimensions 2 2 1 1) (cons 1 2)))
  (should (equal (sacha-get-tile-dimensions 4 2 1 0.5) (cons 1 4)))
  (should (equal (sacha-get-tile-dimensions 12 1 1 (/ 4.0 3.0)) (cons 4 3)))
  (should (equal (sacha-get-tile-dimensions 11 1 1 (/ 4.0 3.0)) (cons 4 3)))
  (should (equal (sacha-get-tile-dimensions 13 1 1 (/ 4.0 3.0)) (cons 4 4))))

;;;###autoload
(defun sacha-extract-image-filenames (beg end)
  "Return the filenames from the links in this region."
  (let (files)
    (save-excursion
      (goto-char (min beg end))
      (while (re-search-forward "sketch:" (max beg end) t)
        (let ((link (org-element-context)))
          (add-to-list 'files (org-element-property :path link))))
    files)))

;;;###autoload
(defun sacha-create-sketch-montage (files &optional tiles)

```

```

"Combine the sketches in the region."
(interactive
  (list
    (if (derived-mode-p 'dired-mode)
      (dired-get-marked-files)
      (mapcar 'sacha-get-sketch-filename
        (sacha-extract-image-filenames (min (point) (mark)) (max (point) (mark))))))
    (if current-prefix-arg (read-string "Tiling: ")))
  ;; Extract the links
  (let ((output-file "~/Dropbox/Inbox/output.png"))
    (unless tiles
      (setq tiles
        (format "%dx"
          (car (sacha-get-tile-dimensions (length files) 1500 900 (/ 4.0 3))))))
    (with-temp-buffer
      (cd "~/Dropbox/Inbox/To blog")
      (apply 'call-process
        "montage" nil nil nil
        (append
          files
          (list
            "-geometry" "1500x900>+0+0"
            "-tile" tiles
            (expand-file-name output-file))))
      (if (called-interactively-p 'any) (find-file output-file))))

;;;###autoload
(defun sacha-create-week-montage (beg end)
  (interactive "r")
  (let* ((date (org-read-date nil nil (unless current-prefix-arg "-fri")))
    (filename (format "Week ending %s #journal #weekly" date))
    (full-filename (sacha-get-sketch-filename filename)))
    (if full-filename
      (sacha-org-sketch-open full-filename)
      (sacha-create-index-card-montage
        (mapcar 'sacha-get-sketch-filename
          (sacha-extract-image-filenames (min (point) (mark)) (max (point) (mark))))
        "2x"
        (sacha-prepare-index-card-template filename))))

;;;###autoload
(defun sacha-create-index-card-montage (files &optional tiling filename)
  "Prepare an index card with a montage of the selected sketches as a layer."
  (interactive
    (list
      (if (derived-mode-p 'dired-mode)
        (dired-get-marked-files)
        (mapcar 'sacha-get-sketch-filename
          (sacha-extract-image-filenames (min (point) (mark)) (max (point) (mark))))))
      (let ((async-shell-command-buffer 'new-buffer)
        (index-card (or filename (sacha-prepare-index-card-template))))
        (sacha-create-sketch-montage files tiling)
        (shell-command
          (format "convert %s \\( %s -resize 1500x900 \\) -colorspace cmyk %s"
            (shell-quote-argument (expand-file-name sacha-index-card-template-file))
            (shell-quote-argument (expand-file-name "~/Dropbox/Inbox/output.png"))
            (shell-quote-argument (expand-file-name index-card)))
          (shell-command (format "%s %s &"
            (shell-quote-argument sacha-sketch-executable)

```

```

(shell-quote-argument (expand-file-name index-card))))
(sacha-rotate-screen 180)
(sacha-set-up-sketch-buffer)))

```

add-output-png is:

```

xdotool windowactivate --sync $(xdotool search --name krita | tail -1); sleep 1
xdotool key --delay 50 Alt+l n m ; sleep 3
xdotool type ~/Dropbox/Inbox/output.png ; sleep 1
xdotool key Return ; sleep 3
xdotool key Alt+l l ; sleep 1
xdotool key Tab Tab ; sleep 1
xdotool type 896 ; sleep 1
xdotool key Return

```

9.3.24 Other sketch-related functions

```

;; ##autoload
(defun sacha-show-sketches-as-slideshow (list &optional shuffle)
  "Display a quick slideshow of sketches in LIST.
   If LIST is a string, look up those sketch filenames in my Flickr copy."
  (interactive "MFilter: \nP")
  (apply 'call-process "feh" nil nil nil "-D" "1" "-F" (if shuffle "-z" "")
    (-filter (lambda (x) (string-match "photostream" x))
      (if (stringp list)
          (sacha-list-sketches list t)
          list))))

(defvar sacha-org-index-card-source nil)
;; ##autoload
(defun sacha-org-prompt-index-cards ()
  "Display a buffer for easy selection of questions to work on."
  (interactive)
  (find-file "~/personal/questions.org")
  (let ((questions
        (cl-sort (org-map-entries 'org-heading-components "TODO=\"DRAW\""
          '<:key (lambda (x) (or (elt x 3) 100))))))
    (setq sacha-org-index-card-source (current-buffer))
    (sacha-rotate-screen 180)
    (sacha-set-up-sketch-buffer)
    (mapc (lambda (q)
      (widget-create 'push-button
        :notify (lambda (widget &rest ignore)
          (sacha-org-sketch-open
            (sacha-prepare-index-card-template
              (widget-value widget))))
          (with-current-buffer sacha-org-index-card-source
            (save-excursion
              (goto-char (org-find-exact-headline-in-buffer ↵
                (widget-value widget) sacha-org-index-card-source t))
              (org-set-property "Effort" "0:15")
              (org-clock-in)
              (org-todo "LINK"))
              (widget-delete widget))
            (elt q 4))
          (insert "\n"))
        questions)

```



```

(text-scale-set 5)
(widget-setup)
(widget-minor-mode)
(goto-char (point-min))
(when (functionp 'scroll-bar-mode) (scroll-bar-mode))
(switch-to-buffer (current-buffer)))

;;;###autoload
(defun sacha-prepare-index-card-for-journal ()
  "Create an index card for my process journal."
  (interactive)
  (quantified-track "Drawing")
  (sacha-prepare-index-card "Journal"))

```

```

(with-eval-after-load 'org
  (let ((listvar (if (boundp 'org-speed-commands) 'org-speed-commands
                    'org-speed-commands-user)))
    (add-to-list listvar '("d" call-interactively 'sacha-prepare-index-card-for-subtree))))

```

9.3.25 SOMEDAY Write about half-page scans

```

;;;###autoload
(defun sacha-write-about-half-page-scan (filename)
  (interactive (list (read-file-name (format "Sketch (%s): "
                                             (file-name-base (sacha-latest-file <-
<- sacha-scan-directory)))
                                (expand-file-name sacha-scan-directory)
                                (sacha-latest-file sacha-scan-directory)
                                nil
                                (expand-file-name sacha-scan-directory)
                                (lambda (f) (string-match "\\.(jpg|png|)$" f))))))
  (let (new-name)
    (shell-command (concat "~/bin/prepare-half-page " (shell-quote-argument filename)))
    (if (string-match "[0-9]+-[0-9]+-[0-9]+\\([a-z]\\|-[0-9]+\\)?\\.?" (file-name-base filename))
        (progn
          (rename-file filename (expand-file-name (file-name-nondirectory filename) <-
<- sacha-sketches-directory) t)
          (setq new-name filename))
        (save-window-excursion
          (find-file filename)
          (setq new-name (expand-file-name (concat (read-string "New name: ") "." <-
<- (file-name-extension filename))
                                           sacha-sketches-directory))
          (rename-file filename new-name)))
    (sacha-write-about-sketch new-name)))

```

9.3.26 Doodles

```

;;;###autoload
(defun sacha-org-copy-as-doodle ()
  (interactive)
  (cond
    ((derived-mode-p 'dired-mode)
     (kill-new
      (mapconcat

```

```
(lambda (s)
  (format
    "#+begin_center-doodle\n#+ATTR_HTML: :style max-height:100px ↵
↵ :alt \n[[file:%s]]\n#+end_center-doodle"
    s))
  (dired-get-marked-files) "\n\n"))
((derived-mode-p 'image-mode)
  (kill-new
    (format
      "#+begin_center-doodle\n#+ATTR_HTML: :style max-height:100px ↵
↵ :alt \n%s\n#+end_center-doodle"
      (org-link-make-string (concat "file:" (buffer-file-name)))))))
))
```

9.3.27 Supernote

supernote

```
(defvar sacha-supernote-export-dir "~/Dropbox/Supernote/EXPORT")
(defvar sacha-dropbox-sketches-dir "~/Dropbox/sketches")
;;;###autoload
(defun sacha-dropbox-sketches-dired () (interactive) (dired sacha-dropbox-sketches-dir))
;;;###autoload
(defun sacha-latest-sketch (&optional skip-download)
  (interactive "P")
  (let ((file
        (or (condition-case nil
              (sacha-supernote-download-latest-exported-file)
              (error nil))
            (sacha-latest-file (list
                               sacha-supernote-export-dir
                               sacha-dropbox-sketches-dir
                               "png\\|svg\\|jpe?g"))))
        (when (called-interactively-p 'any)
              (find-file file))
        file))
    (when (not skip-download)
      (sacha-supernote-process-latest)))
  file)
;;;###autoload
(defun sacha-supernote-process-latest (&optional skip-download)
  (interactive "P")
  (sacha-sketch-process (sacha-latest-sketch skip-download)))
```

```
;;###autoload
(defun sachasupernote-open-latest ()
  (interactive)
  (find-file-other-window
   (or (sachasupernote-download-latest-exported-file)
       (sachasupernote-latest-file (list sachasupernote-export-dir
                                           sachasupernote-sketches-dir)
                                   "png\\|svg\\|jpe?g")))))

;;###autoload
(defun sachasupernote-export-dired ()
```

```
(interactive)
(dired sachasupernote-export-dir "-tl"))
```

```
;;###autoload
(defun sachas-image-autorotate (file)
  (let ((tags (sachas-image-tags file)))
    (cond
      ((member "ccw" tags)
       (call-process "mogrify" nil nil nil "-rotate" "270" file)
       (sachas-image-rename-set file file (delete "ccw" tags)))
      ((member "cw" tags)
       (call-process "mogrify" nil nil nil "-rotate" "90" file)
       (sachas-image-rename-set file file (delete "cw" tags)))
      (t file))))
```

```
;;###autoload
(defun sachas-sketch-process (file &optional do-crop)
  (interactive (list (read-file-name "File: ")))
  (condition-case nil
    (progn
      (sachas-image-recognize file)
      (setq file (sachas-sketch-rename file))

      (pcase (file-name-extension file)
        ((or "svg" "pdf")
         (setq file
              (sachas-image-store
               (sachas-sketch-svg-prepare file)))
              ((or "png" "jpg" "jpeg")
               (setq file
                    (sachas-image-store
                     (sachas-image-autorotate
                      (if do-crop
                        (sachas-image-autocrop file)
                        (sachas-sketch-recolor-png
                         (sachas-sketch-svg-prepare file)
                         (error nil))
                        (find-file file)
                        (find-file-other-window (concat (file-name-sans-extension file) ".txt"))
                        file)
                     file))))))
      (error nil))
    (error nil))
  (find-file file)
  (find-file-other-window (concat (file-name-sans-extension file) ".txt"))
  file)
```

```
;;###autoload
(defun sachas-open-latest-export ()
  (interactive)
  (find-file (sachas-latest-file "~/Dropbox/Supernote/EXPORT")))

;;###autoload
(defun sachas-copy-latest-export-filename ()
  (interactive)
  (kill-new (sachas-latest-file "~/Dropbox/Supernote/EXPORT")))
```

```
(defvar sacha-supernote-inbox "~/Dropbox/Supernote/INBOX")
;;###autoload
(defun sacha-save-manpage-to-supernote (path)
  (interactive (list (woman-file-name nil)))
  (unless (file-exists-p path) (setq path (woman-file-name path)))
  (let* ((base (file-name-base path))
         (temp-html (make-temp-file base nil ".html")))
    (with-temp-buffer
      (insert-file-contents path)
      (call-process-region (point-min) (point-max) "man2html" t t)
      (when (re-search-backward "Invalid Man Page" nil t)
        (delete-file temp-html)
        (error "Could not convert."))
      (write-file temp-html))
    (call-process "ebook-convert" nil (get-buffer-create "*temp*") nil temp-html
                  (expand-file-name ↵
    (concat base ".epub") sacha-supernote-inbox))
    (delete-file temp-html)))
```

```
;; ##autoload
(defun sacha-supernote-save-info (path)
  (interactive (list (read-file-name "Texi: " nil nil
    ↵
    (and Info-current-file ↵
    ↵
    (file-exists-p (concat Info-current-file ".texi")) ↵
    ↵
    (concat Info-current-file ".texi"))) ↵
    nil ↵
    (lambda (f) ↵
      (or ↵
      (string-match ↵
      ↵
      (file-directory-p f)))))) ↵
    (call-process "texi2pdf" nil "*temp*" t (expand-file-name path)
      "-o"
      (expand-file-name (concat ↵
    ↵
    (file-name-base path) ".pdf")
    ↵
    ↵
```

```
↵
↵ sachasupernoteinbox)))
```

And in general:

```
(defvar sachasupernotecss "~/proj/static-blog/assets/css/style.css")
;;###autoload
(defun sachasupernote-save ()
  (interactive)
  (cond
   ((derived-mode-p 'Man-mode) (sachasave-manpage-to-supernote Man-arguments))
   ((derived-mode-p 'Info-mode)
    (sachasupernote-save-info
     (or (and Info-current-file
              (file-exists-p (concat
                              ↵
                              (concat Info-current-file ".texi"))
              (read-file-name
               "Texi: " nil nil nil nil
               (lambda (f)
                 (or
                  (string-match "\\..texi\\'" f)
                  (file-directory-p f)))))))
     ((derived-mode-p 'org-mode)
      (org-latex-export-to-pdf)
      (copy-file (concat (file-name-base (buffer-file-name)) ".pdf")
                  (expand-file-name (concat ↵
                                      ↵
                                      sachasupernoteinbox) t))
      ((or (derived-mode-p 'html-mode)
            (derived-mode-p 'web-mode)
            (derived-mode-p 'markdown-mode))
       (call-process "pandoc" nil nil nil (buffer-file-name) "-t" "latex"
                     "-o"
                     (expand-file-name ↵
                                      ↵
                                      sachasupernoteinbox)))
      ((and (buffer-file-name) (string-match "\\..\\(pdf\\|lepub\\)" (buffer-file-name)))
       (copy-file (buffer-file-name)
                  (expand-file-name ↵
                                      ↵
                                      (file-name-nondirectory (buffer-file-name))
                                      ↵
                                      sachasupernoteinbox)
                  t))
      (t
       (let ((filename (expand-file-name
                         (concat ↵
                         ↵
                         ↵
                         (format-time-string "%Y-%m-%d-%H-%M-%S"))
                         ↵
                         ".pdf")
                         ↵
                         sachasupernoteinbox)))
        (with-current-buffer (htmlize-buffer
```

```

(call-process-region
 (point-min) (point-max) "wkhtmltopdf" nil nil nil ↵
↵ "--no-background" "-")
filename))))))

```

```

(setq htmlize-css-name-prefix "org-")
(setq htmlize-head-tags "<link rel=\"stylesheet\" ↵
↵ href=\"https://sachachua.com/assets/css/style.css\" />")

```

Using Emacs Lisp to export TXT/EPUB/PDF from Org Mode to the Supernote via Browse and Access SUPERNOTE:ORG:EMACS 1.5

I've been experimenting with the Supernote's Browse and Access feature because I want to be able to upload files quickly instead of waiting for Dropbox to synchronize. Here's how to upload:

```

(defvar sacha-supernote-ip-address)
;;;###autoload
(defun sacha-supernote-upload (filename &optional supernote-path)
  (interactive "FFile: ")
  (setq supernote-path (or supernote-path "/INBOX"))
  (let* ((boundary (mml-compute-boundary '()))
         (url-request-method "POST")
         (url-request-extra-headers
          `(("Content-Type" . ,(format "multipart/form-data; ↵
↵ boundary=%s" boundary))))
         (url-request-data
          (mm-url-encode-multipart-form-data
           `(("file" . (("name" . "file")
↵
↵ ("filename" . ,(file-name-nondirectory filename))
↵
↵ ("content-type" . "application/octet-stream")
↵
↵ ("filedata" . ,(with-temp-buffer
↵
↵ (insert-file-contents-literally filename)
↵
↵ (buffer-substring-no-properties (point-min) (point-max))))))
↵ boundary)))
         (condition-case nil
           (with-current-buffer
              (url-retrieve-synchronously
               (format "http://%s:8089%s" ↵
↵ sacha-supernote-ip-address supernote-path))
              (re-search-backward "^$")
              (prog1 (json-read)
                (kill-buffer)))
           (error
            (copy-file filename (expand-file-name (file-name-nondirectory ↵
↵ filename) sacha-supernote-inbox) t)
            (message "Copied %s to %s, please sync" (file-name-nondirectory ↵
↵ filename) sacha-supernote-inbox))))))

```

HTML isn't supported. Text works, but it doesn't support annotation. PDF or EPUB could work.

It would make sense to register this as an export backend so that I can call it as part of the usual export process.

```
;;###autoload
(defun sacha-supernote-org-upload-as-text (&optional async subtree visible-only body-only ↵
  ↵ ext-plist)
  "Export Org format, but save it with a .txt extension."
  (interactive (list nil current-prefix-arg))
  (let ((filename (org-export-output-file-name ".txt" subtree)))
    (text (org-export-as 'org subtree visible-only body-only ↵
  ↵ ext-plist)))
  ;; consider copying instead of exporting so that #+begin_export html etc. is ↵
  ↵ preserved
  (with-temp-file filename
    (insert text))
  (sacha-supernote-upload filename)))

;;###autoload
(defun sacha-supernote-org-upload-as-pdf (&optional async subtree visible-only body-only ↵
  ↵ ext-plist)
  (interactive (list nil current-prefix-arg))
  (sacha-supernote-upload (org-latex-export-to-pdf async subtree visible-only body-only ↵
  ↵ ext-plist)))

;;###autoload
(defun sacha-supernote-org-upload-as-epub (&optional async subtree visible-only body-only ↵
  ↵ ext-plist)
  (interactive (list nil current-prefix-arg))
  (sacha-supernote-upload (org-epub-export-to-epub async subtree visible-only ext-plist)))

(with-eval-after-load 'org
  (org-export-define-backend
    'supernote nil
    :menu-entry '("S" "Supernote"
                  ((?s "as ↵
    ↵ PDF" sacha-supernote-org-upload-as-pdf)
                  (?e ↵
    ↵ "as EPUB" sacha-supernote-org-upload-as-epub)
                  (?o ↵
    ↵ "as Org" sacha-supernote-org-upload-as-text))))))
```

Adding this line to my Org file allows me to use `\spacing{1.5}` for 1.5 line spacing, so I can write in more annotations..

```
#+LATEX_HEADER+: \usepackage{setspace}
```

Sometimes I use custom blocks for HTML classes. When $\text{\LaTeX}$ complains about undefined environments, I can define them like this:

```
#+LATEX_HEADER+: \newenvironment{whatever_my_custom_environment_is_called}
```

Now I can export a subtree or file to my Supernote for easy review.

I wonder if multimodal AI models can handle annotated images with editing marks...

org-attaching the latest image from my Supernote via Browse and AccessSUPERNOTE:ORG

[2024-09-29 Sun]: Use sketch links when possible. Recolor before cropping so that the grid is removed.

2024-09-26-01 Supernote A5X Browse and Access [supernote.png](#)

nil

I want to quickly get drawings from my Supernote A5X into Emacs so that I can include them in blog posts. Dropbox/Google Drive sync is slow because it synchronizes all the files. The Supernote can mirror its screen as an .mjpeg stream. I couldn't figure out how to grab a frame from that, but I did find out how to [use Puppeteer to take an screenshot of the Supernote's screen mirror](#) (p. 462). Still, the resulting image is a little pixelated. If I turn on Browse and Access, the Supernote can serve directories and files as webpages. This lets me grab the latest file and process it. I don't often have time to fill a full A5 page with thoughts, so [autocropping](#) (p. 31) the image encourages me to get stuff out there instead of holding on to things.

```
(setq sacha-supernote-ip-address "192.168.1.221")
```

```
(defvar sacha-supernote-ip-address "192.168.1.221")
;;;###autoload
(defun sacha-supernote-get-exported-files ()
  (condition-case nil
    (let ((data (plz 'get (format "http://%s:8089/EXPORT" ↵
    ↵ sacha-supernote-ip-address)))
          (list))
      (when (string-match "const json = '\\(.*\\)'" data)
        (sort
         (alist-get 'fileList (json-parse-string (match-string ↵
         ↵ 1 data) :object-type 'alist :array-type 'list))
         :key (lambda (o) (alist-get 'date o))
         :lessp 'string<
         :reverse t)))
      (error nil)))
```

```
;;;###autoload
(defun sacha-sketch-insert-latest-doodle ()
  (interactive)
  (let* ((file (sacha-latest-sketch)))
    (insert
     (format
      "#+begin_right-doodle
#+ATTR_HTML: :title
%s
#+end_right-doodle"
      (org-link-make-string (concat "file:" file))))))
```

```
;;;###autoload
(defun sacha-supernote-download-latest-exported-file ()
  "Save exported file in downloads dir."
  (interactive)
  (let* ((info (car (sacha-supernote-get-exported-files)))
         (dest-dir sacha-download-dir)
         (new-file (and info (expand-file-name (file-name-nondirectory ↵
         ↵ (alist-get 'name info)) dest-dir)))
         renamed)
    (rename-file new-file dest-dir t)
    renamed)
```



```

      (when info
        (copy-file
         (plz 'get (format "http://%s:8089%s" sachasupernote-ip-address
↵
          (alist-get 'uri info))
          :as 'file)
         new-file
         t)
        new-file)))

```

```

;;;###autoload
(defun sachasupernote-insert-latest ()
  (interactive)
  (let ((renamed (sachasupernote-latest-sketch)))
    (when (and renamed (derived-mode-p 'org-mode))
      (if (string-match ↵
↵
        "^([0-9])([0-9])([0-9])([0-9])-([0-9])([0-9])-([0-9])([0-9])-([0-9])([0-9]) "
        (file-name-base renamed))
        (org-insert-link nil (concat "sketchFull:" ↵
↵
        (file-name-base renamed)))
        ;; insert the link
        (org-insert-link nil (concat "file:" renamed)))
        (org-redisplay-inline-images))))

```

```

;;;###autoload
(defun sachasupernote-org-attach-latest-exported-file ()
  (interactive)
  ;; save the file to the screenshot directory
  (let ((info (car (sachasupernote-get-exported-files))))
    (new-file
     renamed)
    ;; delete matching files
    (setq new-file (expand-file-name
↵
      (replace-regexp-in-string " " "%20" (alist-get 'name info) (org-attach-dir))))
    (when (file-exists-p new-file)
      (delete-file new-file))
    (org-attach-attach
     (format "http://%s:8089%s" sachasupernote-ip-address
      (alist-get 'uri info))
     nil
     'url)
    (setq new-file (sachasupernote-latest-file (org-attach-dir)))
    ;; recolor
    (sachasupernote-recolor-png new-file)
    ;; autocrop that image
    (sachasupernote-image-autocrop new-file)
    ;; possibly rename
    (setq renamed (sachasupernote-image-recognize-get-new-filename new-file))
    (when renamed
      (setq renamed (expand-file-name renamed (org-attach-dir)))
      (rename-file new-file renamed t)
      (sachasupernote-image-store renamed) ; file it in my archive
      (setq new-file renamed))
    ;; use a sketch link if it has an ID
    (if (string-match "^([0-9])([0-9])([0-9])([0-9])-([0-9])([0-9])-([0-9])([0-9])-([0-9])([0-9]) "

```

```

↳ (file-name-base renamed))
                                (org-insert-link nil (concat "sketchFull:" (file-name-base ↳
↳ renamed)))
                                ;; insert the link
                                (org-insert-link nil (concat "attachment:" (replace-regexp-in-string ↳
↳ "#" "%23" (file-name-nondirectory new-file))))
                                (org-redisplay-inline-images)))

```

DONE Using Puppeteer to grab an image from the SuperNote's screen mirror **SUPERNOTE**
[2024-09-13 Fri] I added a [mogriify](#) call to automatically trim the image.

Partly inspired by John Kitchin's video showing how to [copy screenshots from his iPad and do optical character recognition](#) so he can use the images and text in Org Mode, I'd like to be able to draw quick notes while I'm thinking through a topic on my computer.

[Krita](#) might work, but it's awkward to draw on my tablet PC's screen when it's in laptop mode because of the angle. Flipping it to tablet mode is a bit disruptive.

I can draw on my Supernote, which feels a bit more natural. I have a good workflow for [recoloring and renaming exported sketches](#), but exporting via Dropbox is a little slow since it synchronizes all the folders. The SuperNote has a built-in screen mirroring mode with an MJPEG that I can open in a web browser. Saving it to an image is a little complicated, though. `ffmpeg` doesn't work with the MJPEG that it streams, and I can't figure out how to get stuff out aside from using a browser. I can work around this by using Puppeteer and getting a screenshot. Here's a NodeJS snippet that saves that screenshot to a file.

```

/* This file is tangled to ~/bin/supernote-screenshot.js from my config at ↳
↳ https://sachachua.com/dotemacs
Usage: supernote-screenshot.js [filename]
Set SUPERNOTE_URL to the URL.
*/

const process = require('process');
const puppeteer = require('puppeteer');
const url = process.env['SUPERNOTE_URL'] || 'http://192.168.1.221:8080/screencast.mjpeg';
const scale = 0.5;
const delay = 2000;

async function takeSupernoteScreenshot() {
  const browser = await puppeteer.launch({headless: 'new'});
  const page = await browser.newPage();
  await page.setViewport({width: 2808 * scale, height: 3744 * scale, deviceScaleFactor: ↳
↳ 1});
  page.goto(url);
  await new Promise((resolve, reject) => setTimeout(resolve, delay));
  let filename = process.argv[2] || 'screenshot.png';
  await page.screenshot({type: 'png', path: filename, fullPage: true});
  await browser.close();
}

takeSupernoteScreenshot();

```

Then I can call that from Emacs Lisp and run it through my usual [screenshot insertion process](#) (p. 297):

```
;; ##autoload
```

```
(defun sacha-supernote-org-insert-screenshot-from-mirror ()
  "Copy the current image from the SuperNote mirror."
  (interactive)
  (let ((filename (expand-file-name (format-time-string "%Y-%m-%d-%H-%M-%S.png") ↵
    ↵ "~/recordings"))))
    (shell-command-to-string (concat "NODE_PATH=/usr/lib/node_modules node ↵
    ↵ ~/bin/supernote-screenshot.js " (shell-quote-argument filename)))
    ;; trim it
    (call-process "mogrify" nil nil nil "-trim" "+repage" filename)
    (shell-command-to-string (concat "~/bin/recolor.py --colors c0c0c0,f6f396 " ↵
    ↵ (shell-quote-argument filename)))
    (call-interactively 'sacha-org-insert-screenshot)))
```

[2024-09-13 Fri] I already have some code elsewhere for using the Google Cloud Vision API to extract text from an image, so I should hook that up sometime. Also, OpenAI supports multimodal requests, so I've been thinking about using AI to recognize text and diagrams like in [this Supernote to Markdown example](#). Fun fun fun!

9.4 Audio

Likewise, I have a number of headings that deal with audio.

- [images/Sacha.org\(2023\)](#)
- [images/Sacha.org](#)
- [images/Sacha.org](#)
- [images/Sacha.org](#)

9.5 FFmpeg

```
;; ##autoload
(defun sacha-ffmpeg-save-last-frame-as-image (input-file output-image)
  (interactive "FInput: \nFOutput: ")
  (let ((args (list
    "-sseof" "-2"
    "-i"
    (expand-file-name input-file)
    "-update"
    "1"
    "-q:v"
    "1"
    "-y"
    (expand-file-name output-image))))
    (with-current-buffer (get-buffer-create "*ffmpeg*")
      (insert "\nffmpeg "
        (mapconcat #'shell-quote-argument args ↵
        ↵ " ") "\n")
      (apply 'call-process "ffmpeg" nil t nil args))))
```

9.6 Subtitles with Subed

audio

9.6.1 Reformat speaker in a two-speaker transcript

```

;;;###autoload
(defun sacha-subed-format-second-speaker (speaker-name)
  "Italicize and shift the subtitles for SPEAKER-NAME"
  (interactive "MSpeaker name: ")
  (goto-char (point-min))
  (let ((line-pos "80%")
        (first t))
    (while (re-search-forward
            (format "\n<v %s>\\(\\.+\\)</v>" (regexp-quote speaker-name))
            nil t)
      (replace-match
       (format " line:%s\n<v %s><i>%s</i></v>"
               line-pos
               speaker-name
               (if first
                   (concat speaker-name ": ")
                   (match-string 1))))
      (setq first nil))))

```

9.6.2 Interleave images with transcript

```

;;;###autoload
(defun sacha-subed-interleave-image-links (dir &optional offset-ms)
  (interactive (list (read-file-name "Directory: ")
                    (if ←
                        (current-prefix-arg (read-number "Offset (ms): ")
                        (setq offset-ms (or offset-ms 0)))
                        (let* ((start-of-recording (sacha-filename-timestamp (buffer-file-name)))
                              (subtitles (subed-subtitle-list))
                              (end-of-recording
                               (time-add start-of-recording
                                           (seconds-to-time (/ (elt ←
                                                                    ←
                                                                    1000.0))))))
                              (files
                               (sort
                                (seq-keep
                                 (lambda (f)
                                   (let ((time (sacha-filename-timestamp ←
                                                                    ← f)))
                                     (when (and
                                           (not ←
                                           (not ←
                                           (time-less-p time start-of-recording)
                                           (time-less-p end-of-recording time)))
                                       (cons
                                        ;; ms
                                        (* 1000 (float-time ←
                                                                    f))))
                                     (directory-files dir t ←
                                                           "20250123.*\\.(jpg\\|png\\|svg\\|webm\\)"))
                                   :key 'car))))
                              ;; Now I have the cues and the file timestamps.

```

```

;; Do I want to add the comments to the current file or go streak to breaking ↵
↵ it out?
;; Let's break it out into a different buffer.
(save-excursion
  (goto-char (point-min))
  (unless (subed-subtitle-msecs-start) (subed-forward-subtitle-time-start))
  (dolist (cue subtitles)
    (when (and files (≥ (+ offset-ms (elt cue 1)) (caar files)))
      (let ((link (org-link-make-string (concat "file:" ↵
↵ (cdar files))))))
        (setf (elt cue 4)
              (if (elt cue 4) ↵
                  link))
        (subed-set-subtitle-comment (elt cue 4)))
      (pop files))
    (subed-forward-subtitle-time-start)))
(with-current-buffer (get-buffer-create "*interleaved*")
  (erase-buffer)
  (org-mode)
  (insert (subed-subtitle-list-text subtitles t))
  (goto-char (point-min))
  (switch-to-buffer (current-buffer))))

;;;###autoload
(defun sacha-subed-interleave-calculate-offset (filename)
  (interactive "FFile: ")
  (let ((start-of-recording (sacha-filename-timestamp (buffer-file-name)))
        (file-timestamp (sacha-filename-timestamp filename)))
    (message "%d"
             (- (* 1000.0
                  (time-to-seconds ↵
↵ (time-subtract file-timestamp start-of-recording)))
              (subed-subtitle-msecs-start))))))

```

9.6.3 TOBLOG Split a transcript into phrases for subtitles

```

;;;###autoload
(defun sacha-split-at-words ()
  (interactive)
  (while (not (eobp))
    (recenter)
    (remove-overlays (point-min) (point-max) 'sacha-split t)
    (save-excursion
      (forward-word 3)
      (dotimes (n 10)
        (let* ((word-start (point))
               (word-end (progn (skip-syntax-forward ↵
↵ "^ ") (point)))
               (overlay (make-overlay word-start ↵
↵ word-end)))
          (overlay-put overlay 'sacha-split t)
          (overlay-put overlay 'split-num n)
          (overlay-put overlay 'evaporate t)
          (overlay-put overlay 'before-string (propertize ↵
↵ (format "%s" n)

```

```

↵                                     ↵
↵                                     ↵
↵ "white"                             ↵
                                       ↵
↵                                     ↵
↵                                     ↵
↵                                     ↵
                                       :background "blue"))))
                                       ↵
                                   (skip-syntax-forward " ")
                                   )))
      (let* ((input (read-char "Split: "))
              (num (unless (= input 13)                ; enter
                           (string-to-number ←
                                ↵
                                  (char-to-string input))))
              (match (when num (seq-find (lambda (ov)
                                           ↵
                                             (and ←
                                                  ↵
                                                    (= (overlay-get ov 'split-num) num)))
                                          (overlays-in ←
                                              ↵
                                                (point-min) (point-max))))))
            (if match
                  (progn
                    (goto-char (overlay-start match))
                    (skip-syntax-backward " ")
                    (delete-region (point) (overlay-end match))
                    (insert "\n")))
                  (forward-word 7)))))
;; ##autoload
(defun sacha-split-clear-overlays ()
  (interactive)
  (remove-overlays (point-min) (point-max) 'sacha-split t))
```

9.6.4 Transcript editing

```
;; ##autoload
(defun sacha-split-sentence-and-capitalize ()
  (interactive)
  (delete-char 1)
  (insert ".")
  (capitalize-word 1))
;; ##autoload
(defun sacha-split-sentence-delete-word-and-capitalize ()
  (interactive)
  (delete-char 1)
  (insert ".")
  (kill-word 1)
  (capitalize-word 1))
;; ##autoload
(defun sacha-delete-word-and-capitalize ()
  (interactive)
  (skip-syntax-backward "w")
  (kill-word 1))
```

```
(capitalize-word 1))
```

```
;;;###autoload
(defun sach-emms-player-mplayer-set-speed (speed)
  "Depends on mplayer's -slave mode"
  (interactive "MSpeed: ")
  (process-send-string emms-player-simple-process-name
    (format "speed_set %s\n" speed)))

(defvar sach-emms-player-mplayer-speed-increment 0.1)

;;;###autoload
(defun sach-emms-player-mplayer-speed-up ()
  "Depends on mplayer's -slave mode"
  (interactive)
  (process-send-string emms-player-simple-process-name
    (format "speed_incr %f\n" sach-emms-player-mplayer-speed-increment)))

;;;###autoload
(defun sach-emms-player-mplayer-slow-down ()
  "Depends on mplayer's -slave mode"
  (interactive)
  (process-send-string emms-player-simple-process-name
    (format "speed_incr %f\n" (- 0 ←
    ↪ sach-emms-player-mplayer-speed-increment)))))
```

9.6.5 Remove underlining from WhisperX VTT

```
;;;###autoload
(defun sach-subed-remove-whisperx-underlines ()
  "Remove underlines from the transcript.
If you called whisperx with --highlight_words, this function can remove the underlines."
  (interactive)
  (let (results)
    (dolist (cue (subed-subtitle-list))
      (let ((text (replace-regexp-in-string "</?u>" "" (elt cue 3))))
        (if (and results (string= text (elt (car results) 3)))
            (setf (elt (car results) 2) (elt cue 2))
            (setf (elt cue 3) text)
            (push cue results))))
      (goto-char (point-min))
      (subed-forward-subtitle-start-pos)
      (delete-region (point) (point-max))
      (subed-append-subtitle-list (reverse results)))))
```

9.6.6 Adjust subtitles

```
;;;###autoload
(defun sach-subed-move-succeeding-subtitles-based-on-mpv ()
  "Move current and succeeding subtitles so that current starts at MPV playing position."
  (interactive)
  (if subed-mpv-playback-position
      (subed-move-subtitles
        (- subed-mpv-playback-position (subed-subtitle-msecs-start))
        (point) (point-max))
      (error "Need playback position.")))
```

```

;; ##autoload
(defun sacha-subed-check-random ()
  (interactive)
  (let* ((list (subed-subtitle-list))
         (pos (random (length list))))
    (subed-jump-to-subtitle-id
     (subed-msecs-to-timestamp (elt (elt list pos) 1)))
    (subed-mpv-jump-to-current-subtitle)
    (subed-mpv-unpause)))

```

9.6.7 Extract part of a video

```

;; ##autoload
(defun sacha-subed-get-region-start-stop (beg end)
  (interactive "r")
  (cons (save-excursion
          (goto-char (min beg end))
          (subed-subtitle-msecs-start))
        (save-excursion
          (goto-char (max beg end))
          (subed-subtitle-msecs-stop))))

;; ##autoload
(defun sacha-extend-file-name (original name &optional extension)
  "Add NAME to the end of ORIGINAL, before the file extension."
  (concat (file-name-sans-extension original) " " name ".")
  (or extension (file-name-extension original)))

;; ##autoload
(defun sacha-adjust-subtitles (offset)
  "Change all of the start and end times by OFFSET."
  (interactive (list (subed--string-to-msecs (read-string "Time: "))))
  (subed-for-each-subtitle (point-min) (point-max) nil
    (subed-adjust-subtitle-time-start offset t t)
    (subed-adjust-subtitle-time-stop offset t t))
  (subed-regenerate-ids))

;; ##autoload
(defun sacha-subed-write-adjusted-subtitles (source-file start-msecs end-msecs dest-file)
  (let ((s (with-current-buffer (find-file-noselect source-file)
            (buffer-substring-no-properties
             (subed-jump-to-subtitle-id-at-msecs start-msecs)
             (progn (subed-jump-to-subtitle-id-at-msecs end-msecs) ↵
                    (subed-jump-to-subtitle-end))))))
        (offset (- start-msecs)))
    (with-current-buffer (find-file-noselect dest-file)
      (erase-buffer)
      (insert s)
      (sacha-adjust-subtitles offset)
      (save-buffer)
      (buffer-file-name))))

;; ##autoload
(defun sacha-msecs-to-timestamp (msecs)
  "Convert MSECS to string in the format HH:MM:SS.MS."
  (concat (format-seconds "%02h:%02m:%02s" (/ msecs 1000))

```



```

        "." (format "%03d" (mod msec 1000)))

;;;###autoload
(defun sacha-subed-make-animated-gif (beg end name)
  (interactive "r\nMName: ")
  (let* ((video-file (subed-guess-video-file))
         (msecs (sacha-subed-get-region-start-stop beg end))
         (new-file (sacha-extend-file-name video-file name "gif"))
         cmd)
    (when (> (length name) 0)
      (setq cmd
            (format "ffmpeg -y -i %s -ss %s -t %s -vf subtitles=%s -r 10 -c:a copy -shortest ↵
↵ -async 1 %s"
                    (shell-quote-argument video-file)
                    (sacha-msecs-to-timestamp (car msecs))
                    (sacha-msecs-to-timestamp (- (cdr msecs) (car msecs)))
                    (shell-quote-argument (sacha-subed-write-adjusted-subtitles beg end name))
                    (shell-quote-argument new-file)))
            (message "%s" cmd)
            (kill-new cmd)
            (shell-command cmd))))

;;;###autoload
(defun sacha-subed-ffmpeg-make-mute-filter (segments)
  (mapconcat
   (lambda (s)
     (format "volume=enable='between(t,%.3f,%.3f)':volume=0"
             (/ (car s) 1000.0)
             (/ (cdr s) 1000.0)))
   segments ", "))

;;;###autoload
(defun sacha-subed-cut-video (beg end name video-file caption-file &optional kill-only)
  (interactive
   (append
    (if (use-region-p)
        (list (point) (mark))
        (list (save-excursion (subed-jump-to-subtitle-id))
              (save-excursion (subed-jump-to-subtitle-end))))
    (list
     (expand-file-name (read-file-name "New video filename: "))
     (if (derived-mode-p 'subed-mode) (expand-file-name (subed-media-file))
         (read-file-name "Video: "))
     (if (derived-mode-p 'subed-mode) (expand-file-name (buffer-file-name))
         (read-file-name "Captions: ")))))
  (let*
    ((msecs (sacha-subed-get-region-start-stop beg end))
     (new-file name)
     cmd)
    (when (> (length name) 0)
      (setq cmd
            (format "ffmpeg -y -i %s -i %s -ss %s -t %s -shortest -async 1 %s"
                    (shell-quote-argument caption-file)
                    (shell-quote-argument video-file)
                    (sacha-msecs-to-timestamp
                     (car msecs))
                    (sacha-msecs-to-timestamp
                     (-
                      (cdr msecs)

```

```

        (car msec))
      (shell-quote-argument new-file)))
  (message "%s" cmd)
  (if kill-only (kill-new cmd)
    (shell-command cmd))))))

```

9.6.8 Hide IDs and times

```

(define-minor-mode sach-subed-hide-nontext-minor-mode
  "Minor mode for hiding non-text stuff.")
;;###autoload
(defun sach-subed-hide-nontext-overlay (start end)
  (let ((new-overlay (make-overlay start end)))
    (overlay-put new-overlay 'invisible t)
    (overlay-put new-overlay 'intangible t)
    (overlay-put new-overlay 'evaporate t)
    (overlay-put new-overlay 'read-only t)
    (overlay-put new-overlay 'hide-non-text t)
    (with-silent-modifications
      (add-text-properties start end '(read-only t)))
    new-overlay))

;;###autoload
(defun sach-subed-hide-nontext ()
  (interactive)
  (remove-overlays (point-min) (point-max) 'invisible t)
  (when sach-subed-hide-nontext-minor-mode
    (save-excursion
      (goto-char (point-min))
      (subed-jump-to-subtitle-id)
      (sach-subed-hide-nontext-overlay (point-min) (subed-jump-to-subtitle-text))
      (let (next)
        (while (setq next (save-excursion (subed-forward-subtitle-text)))
          (subed-jump-to-subtitle-end)
          (sach-subed-hide-nontext-overlay (1+ (point)) (1- next))
          (subed-forward-subtitle-text))))))

;;###autoload
(defun sach-subed-show-all ()
  (interactive)
  (let ((inhibit-read-only t))
    (with-silent-modifications
      (remove-text-properties (point-min) (point-max) '(read-only t))
      (remove-overlays (point-min) (point-max) 'invisible t))))

;;###autoload
(defun sach-ignore-read-only (f &rest args)
  (let ((inhibit-read-only t))
    (apply f args)
    (sach-subed-hide-nontext)))

(advice-add 'subed-split-and-merge-dwim :around #'sach-ignore-read-only)
(advice-add 'subed-split-subtitle :around #'sach-ignore-read-only)
(advice-add 'subed-merge-with-next :around #'sach-ignore-read-only)
(advice-add 'subed-merge-with-previous :around #'sach-ignore-read-only)
(advice-add 'subed-regenerate-ids :around #'sach-ignore-read-only)
(advice-add 'subed-kill-subtitle :around #'sach-ignore-read-only)

```

9.6.9 Other subtitle code

```
;;;###autoload
(defun sacha-subed-forward-word (&optional arg)
  "Skip timestamps."
  (interactive "^p")
  (setq arg (or arg 1))
  (let ((end (or (save-excursion (subed-jump-to-subtitle-end)) (point))))
    (loop while (> arg 0)
      do
        (forward-word 1)
        (skip-syntax-forward "^\\s")
        (setq arg (1- arg))
        (when (> (point) end)
          (subed-jump-to-subtitle-text)
          (forward-word 1)
          (skip-syntax-forward "^\\s")
          (setq end (or (save-excursion (subed-jump-to-subtitle-end)) (point)))))))

;;;###autoload
(defun sacha-subed-backward-word (&optional arg)
  "Skip timestamps."
  (interactive "^p")
  (setq arg (or arg 1))
  (let ((end (or (save-excursion (subed-jump-to-subtitle-text)) (point))))
    (loop while (> arg 0)
      do
        (backward-word 1)
        (setq arg (1- arg))
        (when (< (point) end)
          (subed-backward-subtitle-text)
          (setq end (point))
          (subed-jump-to-subtitle-end)
          (backward-word 1))))))
```

```
(defhydra sacha-subed ()
  "Make it easier to split and merge"
  ("e" subed-jump-to-subtitle-end "End")
  ("s" subed-jump-to-subtitle-text "Start")
  ("f" sacha-subed-forward-word "Forward word")
  ("b" sacha-subed-backward-word "Backward word")
  ("w" avy-goto-word-1-below "Jump to word")
  ("n" subed-forward-subtitle-text "Forward subtitle")
  ("p" subed-backward-subtitle-text "Backward subtitle")
  (".p" (subed-split-and-merge-dwim 'prev) "Split and merge with previous")
  (".n" (subed-split-and-merge-dwim 'next) "Split and merge with next")
  ("mp" subed-merge-with-previous "Merge previous")
  ("mn" subed-merge-with-next "Merge next")
  ("j" subed-mpv-jump-to-current-subtitle "MPV current")
  ("1" (subed-mpv-playback-speed 1.0) "1x speed")
  ("2" (subed-mpv-playback-speed 0.7) "0.7x speed")
  ("3" (subed-mpv-playback-speed 0.5) "0.5x speed")
  (" " subed-mpv-pause "Pause")
  ("[" (subed-mpv-seek -1000) "-1s")
  ("]" (subed-mpv-seek 1000) "-1s")
  (";" (re-search-forward "[,\\.\\.;]") "Search for break")
  ("uu" (subed-split-and-merge-dwim 'prev) "Split and merge with previous")
  ("hh" (subed-split-and-merge-dwim 'next) "Split and merge with next")
```

```

("hu" subed-merge-with-previous "Merge with previous")
("uh" subed-merge-with-next "Merge with next")
("lf" subed-mpv-find-video "Find video file")
("lu" subed-mpv-play-url "Find video at URL")
("x" kill-word "Kill word")
("S" save-buffer "Save")
("o" (insert "\n") (let ((fill-column (point-max))) (fill-paragraph))))

```

```

(use-package subed
  :if sachalaptop-p
  :preface (load "~/proj/subed/subed-autoloads.el" nil t)
  :load-path "~/proj/subed/subed"
  :config
  (setq subed-subtitle-spacing 1)
  (setq subed-align-mfa-conda-env "/home/sacha/vendor/miniconda3/envs/aligner")
  (key-chord-define subed-mode-map "hu" 'sacha-subed/body)
  (key-chord-define subed-mode-map "ht" 'sacha-subed/body)
  (setq subed-loop-seconds-before 0 subed-loop-seconds-after 0)
  (setq subed-align-mfa-command '("mfa" "align"))
  (setq subed-align-mfa-conda-env "/home/sacha/vendor/miniconda3/envs/aligner")
  (setq subed-align-command
    '("/home/sacha/vendor/aeneas/venv/bin/python3" "-m" ↵
      ↵ "aeneas.tools.execute_task"))
  :bind
  (:map subed-mode-map
    ("M-j" . avy-goto-char-timer)
    ("M-j" . subed-mpv-jump-to-current-subtitle)
    ("M-!" . subed-mpv-seek)))
(use-package subed-record
  :load-path "~/proj/subed-record"
  :config
  (remove-hook 'subed-sanitize-functions 'subed-sort)
  (setq subed-record-ffmpeg-args (split-string "-y -f pulse -i VirtualMicSink.monitor -r ↵
    ↵ 48000"))
  :bind
  (:map subed-mode-map ("C-c C-c" . subed-record-compile-video)))

```

9.6.10 Simplify inserting audio links

```

(defvar sachal-subed-audio-link-list nil)
;; ###autoload
(defun sachal-subed-remove-audio-links (beg end)
  (interactive (if (region-active-p)
    (list (region-beginning)
      (region-end))
    (save-excursion
      (org-back-to-heading)
      (org-end-of-meta-data t)
      (list (point)
        (save-excursion (org-end-of-subtree)
          (point)))))))
  (save-excursion
    (goto-char beg)
    (while (re-search-forward "vtime:[0-9:]+ +" nil t)
      (replace-match ""))))

```

```

;;;###autoload
(defun sacha-subed-load-audio-links (&optional op)
  (interactive (list
    (cond
      ((null current-prefix-arg) 'insert)
      ((equal current-prefix-arg '(4)) 'list)
      ((equal current-prefix-arg '(16)) 'skip))))
  (cond
    ((derived-mode-p 'subed-mode)
     (setq sacha-subed-audio-link-list (subed-subtitle-list)))
    ((derived-mode-p 'org-mode)
     (save-excursion
      (unless (eq 'link (org-element-type (org-element-context)))
        (re-search-backward "\\(audio\\|video\\):"))
      (let ((filename
        (car
          (url-path-and-query (url-generic-parse-url (org-element-property :path ↵
↵ (org-element-context)))))))
        (setq sacha-subed-audio-link-list
          (seq-filter
            (lambda (o)
              (and (elt o 3) (not (string= (elt o 3) ""))))
            (subed-parse-file
              (concat (file-name-sans-extension filename)
                ".vtt"))))))
      (pcase op
        ('insert (call-interactively #'sacha-subed-insert-audio-links))
        ('list (sacha-subed-insert-audio-links-as-list)))
      sacha-subed-audio-link-list)))

;;;###autoload
(defun sacha-subed-remove-audio-links (beg end)
  "Remove audio links from region."
  (interactive (cond
    ((region-active-p)
     (list (region-beginning)
       (region-end)))
    ((org-in-block-p '("media-post"))
     (let ((block (org-element-lineage (org-element-context) 'special-block)))
       (list
        (org-element-begin block)
        (org-element-end block))))
    (t
     (list
      (point-min) (point-max)))))
  (save-excursion
    (goto-char beg)
    (while (re-search-forward "vtime:[0-9:]+ " end t)
      (replace-match "")))

```

```

;;;###autoload
(defun sacha-org-next-item-or-paragraph (&optional by-sentence)
  (cond
    ((org-in-item-p)
     (condition-case nil
       (progn
        (org-next-item)
        (when (looking-at org-list-full-item-re)

```

```

        (goto-char (match-end 0))))
      (error nil)))
    (by-sentence (forward-sentence))
    (t
     (forward-paragraph)
     (skip-syntax-forward " ")
     (when (looking-at org-list-full-item-re)
       (goto-char (match-end 0)))
     ; Move to start of a list item
     (when (looking-at org-heading-regexp)
       (org-end-of-meta-data t))
     (skip-syntax-forward " "))))

```

```

;;;###autoload
(defun sach-org-vtime-link (o &optional keep-hours)
  (concat "vtime:"
          (if (or keep-hours (>= (elt o 1) (* 60 60 1000)))
              (substring (car o) 0 8)
              (substring (car o) 3 8))))

```

```

;;;###autoload
(defun sach-subed-insert-next-audio-link (&optional by-sentence)
  (interactive (list current-prefix-arg))
  (let* ((candidates
          (seq-keep (lambda (o)
                      (unless (string= (string-trim (elt o 3)) "")
                        (cons (replace-regexp-in-string "<.+?>" "" (elt o 3))
                              (car o))))
                    sach-subed-audio-link-list))
         (sentence
          (sach-org-simplify-text
           (replace-regexp-in-string
            "\\*" ""
            (replace-regexp-in-string
             " *{.+?}" ""
             (let ((sentence (sentence-at-point))
                   (subs (buffer-substring (point) (line-end-position))))
               (if (and sentence (< (length sentence) (length subs)))
                   sentence
                   subs))))))
          (choice
           (or
            (cdr
             (seq-find
              (lambda (o)
                (subed-word-data-compare-normalized-string-distance
                 sentence
                 (replace-regexp-in-string
                  "\\*" ""
                  (replace-regexp-in-string " *{.+?}" "" (car o))))))
              candidates))
            (consult--read
             candidates
             :lookup 'consult--lookup-cdr
             :sort nil))))
         (save-excursion
          (insert "vtime:" (replace-regexp-in-string "\\.[0-9]+" "" choice) " ")))

```

```

(sacha-org-next-item-or-paragraph by-sentence)
(setq sacha-subed-audio-link-list
  (seq-remove
    (lambda (o) (string= (car o) choice))
    sacha-subed-audio-link-list)))

;;;###autoload
(defun sacha-subed-insert-audio-links (&optional beg end do-load)
  (interactive (cond
    ((region-active-p)
      (list (region-beginning)
            (region-end)
            current-prefix-arg))
    ((org-in-block-p '("media-post"))
      (let ((block (org-element-lineage (org-element-context) 'special-block)))
        (list
          (org-element-begin block)
          (org-element-end block)
          current-prefix-arg)))
    (t
      (list
        (point-min) (point-max)
        current-prefix-arg))))
  (setq beg (or beg (point)))
  (setq end (or end (point-max)))
  (when (or do-load (null sacha-subed-audio-link-list))
    (save-excursion
      (unless (eq 'link (org-element-type (org-element-context)))
        (re-search-backward "audio:" nil t))
      (let ((elem (org-element-context)))
        (when (and (eq 'link (org-element-type elem))
                    (string= "audio" (org-element-property :type elem)))
          (sacha-subed-load-audio-links)
          (sacha-org-next-item-or-paragraph)))))
  (save-restriction
    (narrow-to-region beg end)
    (while (and sacha-subed-audio-link-list
                 (not (eobp)))
      (sacha-subed-insert-next-audio-link)))

;;;###autoload
(defun sacha-subed-insert-audio-links-as-list ()
  (interactive)
  (dolist (cue sacha-subed-audio-link-list)
    (insert "- " (sacha-org-vtime-link cue) " " (elt cue 3) "\n")))

```

9.6.11 Using Emacs to fix automatically generated subtitle timestamps

emacs

I like how people are making more and more Emacs-related videos. I think subtitles, transcripts, and show notes would go a long way to helping people quickly search, skim, and squeeze these videos into their day.

Youtube's automatically-generated subtitles overlap. I think some players scroll the subtitles, but the ones I use just display them in alternating positions. I like to have non-overlapping subtitles, so here's some code that works with [subed.el](#) to fix the timestamps.

```

;;;###autoload
(defun sacha-subed-fix-timestamps ()

```

```
"Change all ending timestamps to the start of the next subtitle."
(interactive)
(goto-char (point-max))
(let ((timestamp (subed-subtitle-msecs-start)))
  (while (subed-backward-subtitle-time-start)
    (subed-set-subtitle-time-stop timestamp)
    (setq timestamp (subed-subtitle-msecs-start)))))
```

Then it's easy to [edit the subtitles](#) (punctuation, capitalization, special terms), especially with the shortcuts for splitting and merging subtitles.

For transcripts with starting and ending timestamps per paragraph, I like using the merge shortcut to merge all the subtitles for a paragraph together. Here's a sample: <https://emacsconf.org/2020/talks/05/>

Tonight I edited automatically-generated subtitles for a screencast that was about 40 minutes long. The resulting file had 1157 captions, so about 2 seconds each. I finished it in about 80 minutes, pretty much the 2x speed that I've been seeing. I can probably get a little faster if I figure out good workflows for:

- jumping: avy muscle memory, maybe?
- splitting things into sentences and phrases
- fixing common speech recognition errors (ex: emax -> Emacs, which I handle with regex replaces; maybe a list of them?)

I experimented with making a hydra for this before, but thinking about the keys to use slowed me down a bit and it didn't flow very well. Might be worth tinkering with.

Transcribing from scratch takes me about 4-5x playtime. I haven't tweaked out my workflow for that one yet because I've only transcribed one talk with subed.el, and there's a pretty big backlog of talks that already have automatically generated subtitles to edit.

So that's another thing I (or other people) can occasionally do to help out even if I don't have enough focused time to think about a programming challenge or do a podcast myself. And I get to learn more in the process, too. Fun!

9.6.12 Using word-level timing information when editing subtitles or captions in Emacs emacs

2022-10-26: [Merged word-level timing support into subed.el](#), so I don't need my old caption functions.

2022-04-18: Switched to using yt-dlp.

I like to split captions at logical points, such as at the end of a phrase or sentence. At first, I used subed.el to play the video for the caption, pausing it at the appropriate point and then calling `subed-split-subtitle` to split at the playback position. Then I modified `subed-split-subtitle` to split at the video position that's proportional to the text position, so that it's roughly in the right spot even if I'm not currently listening. That got me most of the way to being able to quickly edit subtitles.

It turns out that word-level timing is actually available from YouTube if I download the autogenerated SRV2 file using yt-dlp, which I can do with the following function:

```
;; ##autoload
(defun sach-caption-download-srv2 (id)
  (interactive "MID: "))
```



```
(require 'subed-word-data)
(when (string-match "v=\\([^\&]+\\)" id) (setq id (match-string 1 id)))
(let ((default-directory "/tmp"))
  (call-process "yt-dlp" nil nil nil "--write-auto-sub" "--write-sub" "--no-warnings" ←
    ↪ "--sub-lang" "en" "--skip-download" "--sub-format" "srv2"
      (concat "https://youtu.be/" id))
  (subed-word-data-load-from-file (sacha-latest-file "/tmp" "\\srv2\\"))))
```

2022-10-26: I can also generate a SRV2-ish file using torchaudio, which I can then load with subed-word-data-load-from-file.

```
;;;###autoload
(defun sacha-caption-fix-common-errors (data)
  (mapc (lambda (o)
    (mapc (lambda (e)
      (when (string-match (concat "\\<" (regexp-opt (if (listp e) (seq-remove ↪
        ↪ (lambda (s) (string= "" s)) e)
          (list e)))
        "\\>")
      (alist-get 'text o))
      (map-put! o 'text (replace-match (car (if (listp e) e (list e))) t t ↪
        ↪ (alist-get 'text o))))))
    sacha-subed-common-edits))
  data))
```

Assuming I start editing from the beginning of the file, then the part of the captions file after point is mostly unedited. That means I can match the remainder of the current caption with the word-level timing to try to figure out the time to use when splitting the subtitle, falling back to the proportional method if the data is not available.

```
;;;###autoload
(defun subed-avy-set-up-actions ()
  (interactive)
  (make-local-variable 'avy-dispatch-alist)
  (add-to-list
    'avy-dispatch-alist
    (cons ?, 'subed-split-subtitle)))

;;;###autoload
(defun sacha-subed-maybe-save-place ()
  (when buffer-file-name (save-place-local-mode 1)))
```

```
(use-package subed
  :if sacha-laptop-p
  :load-path "~/proj/subed"
  :preface (load "~/proj/subed/subed-autoloads.el" nil t)
  :mode
  (("\\.vtt\\\"" . subed-vtt-mode)
   ("\\.srt\\\"" . subed-srt-mode)
   ("\\.ass\\\"" . subed-ass-mode))
  :init
  (autoload 'subed-vtt-mode "subed-vtt" nil t)
  (autoload 'subed-srt-mode "subed-srt" nil t)
  (autoload 'subed-ass-mode "subed-ass" nil t)
  (autoload 'subed-txt-mode "subed-txt" nil t))
```

```

:hook
(subed-mode . display-fill-column-indicator-mode)
(subed-mode . subed-avy-set-up-actions)
:bind
(:map subed-mode-map
  ("M-," . subed-split-subtitle)
  ("M-." . subed-merge-dwim))
:config
;; Remember cursor position between sessions
(add-hook 'subed-mode-hook 'sacha-subed-maybe-save-place)
;; Some reasonable defaults
;; Replay subtitles as you adjust their start or stop time with M-[, M-], M-[, or M-}
(add-hook 'subed-mode-hook 'subed-enable-replay-adjusted-subtitle)
;; Loop over subtitles
(add-hook 'subed-mode-hook 'subed-enable-loop-over-current-subtitle)
;; Show characters per second
(add-hook 'subed-mode-hook 'subed-enable-show-cps)
(with-eval-after-load 'consult
  (advice-add 'consult-buffer :around
    (lambda (f &rest r)
      (let ←
        ↪ ((subed-auto-play-media nil))
          (apply f r))))))
)

```

That way, I can use the word-level timing information for most of the reformatting, but I can easily replay segments of the video if I'm unsure about a word that needs to be changed.

If I want to generate a VTT based on the caption data, breaking it at certain words, these functions help:

```

(defvar sacha-caption-breaks
  '("the" "this" "we" "we're" "I" "finally" "but" "and" "when")
  "List of words to try to break at.")
;; ###autoload
(defun sacha-caption-make-groups (list &optional threshold)
  (let (result
        current-item
        done
        (current-length 0)
        (limit (or threshold 70))
        (lower-limit 30)
        (break-regexp (concat "\\<" (regexp-opt sacha-caption-breaks) "\\>")))
    (while list
      (cond
        ((null (car list)))
        ((string-match "^\\n*$" (alist-get 'text (car list))))
        (push (cons '(text . " ") (car list)) current-item)
        (setq current-length (1+ current-length)))
      ((< (+ current-length (length (alist-get 'text (car list)))) limit)
        (setq current-item (cons (car list) current-item)
              current-length (+ current-length (length (alist-get 'text (car list))) 1)))
      (t (setq done nil)
         (while (not done)
           (cond
            ((< current-length lower-limit)
             (setq done t))

```

```

        ((and (string-match break-regexp (alist-get 'text (car current-item)))
              (not (string-match break-regexp (alist-get 'text (cadr current-item)))))
         (setq current-length (- current-length (length (alist-get 'text (car ←
→ current-item)))))
         (push (pop current-item) list)
         (setq done t))
        (t
         (setq current-length (- current-length (length (alist-get 'text (car ←
→ current-item)))))
         (push (pop current-item) list)))
        (push nil list)
        (setq result (cons (reverse current-item) result) current-item nil current-length 0)))
    (setq list (cdr list)))
    (reverse result)))

;;;###autoload
(defun sacha-caption-format-as-subtitle (list &optional word-timing)
  "Turn a LIST of the form ((start . ms) (end . ms) (text . s)) ... into VTT.
If WORD-TIMING is non-nil, include word-level timestamps."
  (format "%s --> %s\n%s\n\n"
          (subed-vtt--msecs-to-timestamp (alist-get 'start (car list)))
          (subed-vtt--msecs-to-timestamp (alist-get 'end (car (last list))))
          (s-trim (mapconcat (lambda (entry)
                              (if word-timing
                                  (format " <%s>%s"
                                          (subed-vtt--msecs-to-timestamp (alist-get 'start ←
→ entry))
                                          (string-trim (alist-get 'text entry)))
                                  (alist-get 'text entry)))
                             list ""))))))

;;;###autoload
(defun sacha-caption-to-vtt (&optional data)
  (interactive)
  (with-temp-file "captions.vtt"
    (insert "WEBVTT\n\n"
            (mapconcat
             (lambda (entry) (sacha-caption-format-as-subtitle entry))
             (sacha-caption-make-groups
              (or data (sacha-caption-fix-common-errors subed-word-data--cache)))
             ""))))

```

9.6.13 Using WhisperX to get word-level timestamps for audio editing with Emacs and subed-record emacs:subed

- [2024-12-09 Mon]: Split whisperx to pass one file per call.
- [2024-11-16 Sat]: Removed `highlight_words`, made `max_line_width` use the environment variable if specified.
- [2024-10-14 Mon]: Actually, WhisperX makes a JSON with word-level timing data, so let's use that instead.

I'm gradually shifting more things to this Lenovo P52 to take advantage of its newer processor, 64 GB of RAM, and 2 TB drive. (Whee!) One of the things I'm curious about is how I can make better use of multimedia. I couldn't get [whisper.cpp](#) to work on my Lenovo X230T, so I mostly relied on the automatic transcripts from Google Recorder (with timestamps generated by [aeneas](#)) or cloud-based transcription services like [Deepgram](#).

I have a lot of silences in my voice notes when I think out loud. [whisper.cpp](#) got stuck in loops during silent parts, but [WhisperX](#) handles them perfectly. WhisperX is also fast enough for me to handle audio files locally instead of relying on Deepgram. With the default model, I can process the files faster than real-time:

File length	Transcription time
42s	17s
7m48s	1m41s

I used this command to get word-level timing data. ([Experimenting with options from this post](#))

```
MODEL="${MODEL:-small}"
VENV_NVIDIA="/home/sacha/vendor/whisperx/.venv/lib/python3.12/site-packages/nvidia"
export ↵
    ↵ LD_LIBRARY_PATH="$VENV_NVIDIA/cudnn/lib:$VENV_NVIDIA/cublas/lib:$VENV_NVIDIA/cuda_runtime/lib:$LD_LIBRARY_PATH"
LANGUAGE="${LANGUAGE:-en}"
PROMPT="${PROMPT:-Omit filler words. Keep in original language. Keywords: Sacha Chua, Emacs, ↵
    ↵ Org Mode}"

export LD_LIBRARY_PATH="$(find ↵
    ↵ /home/sacha/vendor/whisperx/.venv/lib/python3.12/site-packages/nvidia -maxdepth 2 ↵
    ↵ -name lib -type d | tr '\n' ':')${LD_LIBRARY_PATH}"
if [[ "$MODEL" = *large* ]]; then
    DEVICE="cpu"
else
    DEVICE="cuda"
fi
echo $DEVICE
WHISPER_ARGS=("${(z)WHISPER_FLAGS}")
MAX_LINE_WIDTH="${MAX_LINE_WIDTH:-50}"
for FILE in "$@"; do
    text="${FILE%.*}.txt"
    if [ -f "$text" ]; then
        echo "Skipping $FILE as it's already been transcribed."
    else
        ~/vendor/whisperx/.venv/bin/whisperx --model "$MODEL" --align_model ↵
        ↵ WAV2VEC2_ASR_LARGE_LV60K_960H --compute_type int8 --print_progress True ↵
        ↵ --max_line_width $MAX_LINE_WIDTH --language "$LANGUAGE" --device "$DEVICE" ↵
        ↵ --segment_resolution chunk --max_line_count 1 --initial_prompt "$PROMPT" "$FILE" ↵
        ↵ $WHISPER_FLAGS
        rm -f "${FILE%.*}.srt"
    fi
done
```

Among other things, it makes a text file that looks like this:

I often need to... I sometimes need to replace or navigate by symbols.

Casual symbol overlays a new package that adds those shortcuts so that I don't have to remember the other

and a JSON file that looks like this:

```
{
  "segments": [
    {
      "start": 0.427,
      "end": 7.751,
      "text": " I often need to... I sometimes need to ↵
        ↵ replace or navigate by symbols.",
      "words": [
        {
          "word": "I",
          "start": 0.427,
          "end": ↵
          ↵ 0.507,
          "score": 0.994,
        },
        {
          "word": "often",
          "start": 0.587,
          "end": 0.887,
          "score": ↵
          ↵ 0.856,
        },
        {
          "word": "need",
          "start": 0.987,
          "end": 1.227,
          "score": 0.851,
        },
        {
          "word": ↵
          ↵ "to ...",
          "start": 1.267,
          "end": 1.508,
          "score": 0.738,
        },
        {
          "word": "I",
          "start": 4.329,
          ↵
          ↵ "end": 4.429,
          "score": 0.778,
        },
        ...
      ]
    },
    ...
  ]
}
```

Sometimes I just want the text so that I can use [an audio braindump as the starting point for a blog post or for notes](#) (p. 121). WhisperX is way more accurate than Google Recorder, so that will probably be easier once I update my workflow for that.

Sometimes I want to make an edited audio file that sounds smooth so that I can use it in a podcast, a video, or some audio notes. For that, I'd like word-level timing data so that I can cut out words or sections. Aeneas didn't give me word-level timestamps, but WhisperX does, so I can get the time information before I start editing. I can extract the word timestamps from the JSON like this:

```
;; ##autoload
(defun sach-subed-word-tsv-from-whisperx-json (file)
  (interactive "FJSON: ")
  (let* ((json-array-type 'list)
         (json-object-type 'alist)
         (data (json-read-file file))
         (filename (concat (file-name-sans-extension file) ".tsv"))
         (base (seq-mapcat
                  (lambda (segment)
                    (seq-map (lambda (word)
                              ↵
                                (let-alist word
                                  ↵
                                    (list nil
                                          ↵
                                            (and .start (* 1000 .start))
                                            ↵
                                            (and .end (* 1000 .end))
                                            ↵
                                            .word))))
                              ↵
                                (alist-get 'words segment)))
                            (alist-get 'segments data)))
         (current base)
         (last-end 0))
    ;; numbers at the end of a sentence sometimes don't end up with times
    ;; so we need to fix them
    (while current
      (unless (elt (car current) 1) ↵
        ↵ ; start
          (setf (elt (car current) 1) (1+ last-end)))
      (unless (elt (car current) 2)
        (setf (elt (car current) 2) (1- (elt (cadr current) 1))))
      (setq
        last-end (elt (car current) 2)
        current (cdr current)))
    (subed-create-file
      filename
      base
      t
      'subed-tsv-mode)
    (find-file filename)))
```

Here's my old code for parsing the highlighted VTT or SRT files that underline each word:

```
;; ##autoload
(defun sach-subed-load-word-data-from-whisperx-highlights (file)
  "Return a list of word cues from FILE.
FILE should be a VTT or SRT file produced by whisperx with the
```

```

--highlight_words True option."
  (seq-keep (lambda (sub)
              (when (string-match "<u>\\(.+?\\)</u>" ↵
                    (setf (elt sub 3) ↵
                          sub)))
            (subed-parse-file file)))

;;###autoload
(defun sacha-subed-word-tsv-from-whisperx-highlights (file)
  (interactive "FVT: ")
  (with-current-buffer (find-file-noselect (concat (file-name-nondirectory file) ".tsv"))
    (erase-buffer)
    (subed-tsv-mode)
    (subed-auto-insert)
    (mapc (lambda (sub) (apply #'subed-append-subtitle nil (cdr sub)))
          (sacha-subed-load-word-data-from-whisperx-highlights ↵
            ↵ file))
    (switch-to-buffer (current-buffer))))

```

I like to use the TSV format for this one because it's easy to scan down the right side. Incidentally, this format is compatible with [Audacity](#) labels, so I could import that there if I wanted. I like Emacs much more, though. I'm used to having all my keyboard shortcuts at hand.

```

0.427000 0.507000 I
0.587000 0.887000 often
0.987000 1.227000 need
1.267000 1.508000 to ...
4.329000 4.429000 I
4.469000 4.869000 sometimes
4.950000 5.170000 need
5.210000 5.410000 to
5.530000 6.090000 replace

```

Once I've deleted the words I don't want to include, I can merge subtitles for phrases so that I can keep the pauses between words. A quick heuristic is to merge subtitles if they don't have much of a pause between them.

```

(defvar sacha-subed-merge-close-subtitles-threshold 500)
;;###autoload
(defun sacha-subed-merge-close-subtitles (threshold)
  "Merge subtitles with the following one if there is less than THRESHOLD msec gap ↵
↵ between them."
  (interactive (list (read-number "Threshold in msec: " ↵
                                ↵ sacha-subed-merge-close-subtitles-threshold)))
  (goto-char (point-min))
  (while (not (eobp))
    (let ((end (subed-subtitle-msecs-stop)
          (next-start (save-excursion
                        ↵
                        (and (subed-forward-subtitle-time-start)
                            ↵
                            (subed-subtitle-msecs-stop))))))
      (if (and end next-start (< (- next-start end) threshold))
          (subed-merge-with-next)
          (or (subed-forward-subtitle-end) (goto-char (point-max)))))))

```

Then I can use `subed-waveform-show-all` to tweak the start and end timestamps. Here I switch to another file I've been editing...

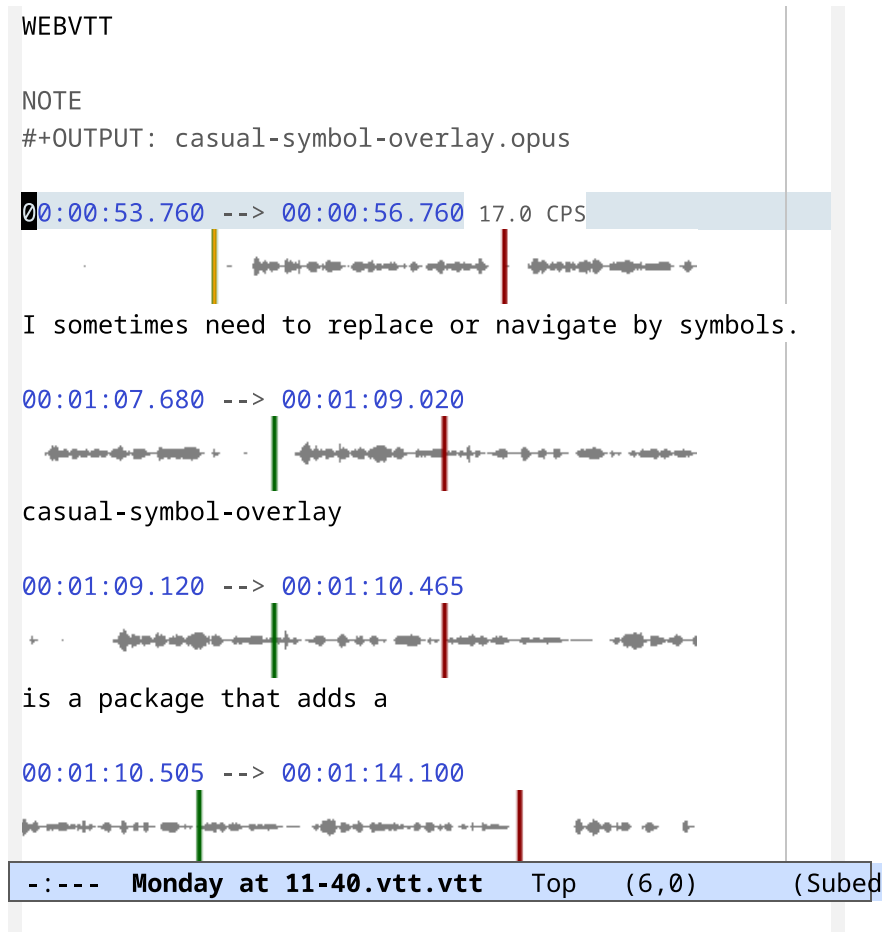


Figure 17: Screenshot of subed-waveform

After that, I can use `subed-record` to compile the audio into an `.opus` file that sounds reasonably smooth.

:

I want to make lots of quick audio notes that I can shuffle and listen to in order to remember things I'm learning about Emacs (might even come up with some kind of spaced repetition system), and I'd like to make more videos someday too. I think WhisperX, subed, and Org Mode will be fun parts of my workflow.

9.6.14 Testing subtitle start times by skimming the first second

```
(defvar sachasubedskim-msecs 1000 "Number of milliseconds to play when skimming.")
;;###autoload
```

```

(defun sacha-subed-skim-starts ()
  (interactive)
  (subed-mpv-unpause)
  (subed-disable-loop-over-current-subtitle)
  (catch 'done
    (while (not (eobp))
      (subed-mpv-jump-to-current-subtitle)
      (let ((ch
        (read-char "(q)uit? " nil (/ ←
          ↪ sacha-subed-skim-msecs 1000.0))))
        (when ch
          (throw 'done t)))
      (subed-forward-subtitle-text)
      (when (and subed-waveform-minor-mode
        (not subed-waveform-show-all))
        (subed-waveform-refresh))
      (recenter)))
    (subed-mpv-pause))

```

9.6.15 DONE Remove filler words at the start and upcase the next word speechtotext:emacs

[2024-11-21 Thu]: Fixed the second filler words regexp, and make it work at the start of lines too. Thanks to [@arialdo@mastodon.online](https://masto.online/@arialdo) for the feedback!

Like many people, I tend to use "So", "And", "You know", and "Uh" to bridge between sentences when thinking. WhisperX does a reasonable job of detecting sentences and splitting them up anyway, but it leaves those filler words in at the start of the sentence. I usually like to remove these from transcripts so that they read more smoothly.

Here's a short Emacs Lisp function that removes those filler words when they start a sentence, capitalizing the next word. When called interactively, it prompts while displaying an overlay. When called from Emacs Lisp, it changes without asking for confirmation.

```

(defvar sacha-filler-words-regexp "\\(\\. \\|^\\)\\(?:So?\\|And\\|You ↪
  ↪ know\\|Uh\\)\\(?:,\\.\\.\\.\\|)? \\(\\.\\)"
;; ##autoload
(defun sacha-remove-filler-words-at-start ()
  (interactive)
  (save-excursion
    (let ((case-fold-search nil))
      (while (re-search-forward sacha-filler-words-regexp nil t)
        (if (and (called-interactively-p) (not current-prefix-arg))
          (let ((overlay (make-overlay (match-beginning 0)
            ↪ (match-end 0))))
            (overlay-put overlay 'common-edit t)
            (overlay-put overlay 'display
              (propertize (concat (match-string 0) ↪
                ↪ " -> "
                ↪ (match-string 1)
                ↪ (upcase (match-string 2))))
            ↪

```



```

↪ 'face 'modus-themes-mark-sel))
                                (unwind-protect
                                (pcase ↪
↪ (save-match-data (read-char-choice "Replace (y/n/!/q)? " "yn!q"))
                                (?!
                                ↪
↪ (replace-match (concat (match-string 1) (upcase (match-string 2)))) t)
                                (while ↪
↪ (re-search-forward sachafillerwords-regexp nil t)
                                ↪
↪ (replace-match (concat (match-string 1) (upcase (match-string 2)))) t)))
                                (?y
                                ↪
↪ (replace-match (concat (match-string 1) (upcase (match-string 2)))) t))
                                (?n nil)
                                (?q (goto-char ↪
↪ (point-max))))
                                (delete-overlay overlay)))
↪ (replace-match (concat (match-string 1) (upcase ↪
↪ (match-string 2))) t))))))

```

9.6.16 Showing captions

This tidbit displays a buffer with the text of the subtitles so that I can quickly skim it.

```

;; ##autoload
(defun sachacaption-show (url)
  (interactive (list
    (let ((link (and (derived-mode-p 'org-mode)
                     (org-element-context))))
      (if (and link
        (eq (org-element-type link) 'link))
        (read-string (format "URL (%s): " (org-element-property :raw-link link)) ↪
        ↪ nil nil
        (org-element-property :raw-link link))
        (read-string "URL: "))))))
  (when (and (listp url) (org-element-property :raw-link url)) (setq url (org-element-property ↪
    ↪ :raw-link url)))
  (delete-other-windows)
  (split-window-right)
  (if (string-match "http" url)
    (with-current-buffer-window "*Captions*"
      'display-buffer-same-window
      nil
      (org-mode)
      (save-excursion
        (sachacorginsertyoutubevideowithtranscript url)))
    (unless (file-exists-p (concat (file-name-sans-extension url) ".vtt"))
      (sachadeepgramrecognizeaudio url))
    (find-file (concat (file-name-sans-extension url) ".vtt"))))

```

9.6.17 Edit text

```

;; Use the saved version of this instead of forcing the reevaluation
(defcustom sachasubedit-common-edits
  '("I"

```

```

"I've"
"I'm"
"Mendeley"
"JavaScript"
"RSS"
    ("stop section" "subsection")
    ("EmacsConf" "EmacsCon" "emacs conf" "imaxconf")
("going to" "gonna")
("want to" "wanna")
("transient" "transit")
("C-c" "control c" "Ctrl+C")
("C-x" "control x" "Ctrl+X")
("C-f" "control f")
(" " "uh" "um")
("Magit" "maggot")
("Emacs" "e-max" "emex" "emax" "bmx" "imax")
("Emacs News" "emacs news")
    ("Emacs Lisp" "emacs list")
("ivy" "iv")
("UI" "ui")
("TECO" "tico")
("org-roam" "orgrim" "orgrom" "Org Rome")
("non-nil" "non-nail")
("commits" "comets")
"SQL"
"arXiv"
"Montessori"
"SVG"
"YouTube" "GitHub" "GitLab" "OmegaT" "Linux" "SourceForge"
"LaTeX"
"Lisp"
"Org"
"IRC"
"Reddit"
"PowerPoint"
"SQLite"
"SQL"
"I'll"
("<f9>" "F-9" "f9")
"I'd"
"PDFs"
"PDF"
"ASCII"
("Spacemacs" "spacemax")
"Elisp"
    "Reddit"
    "TextMate"
    "macOS"
    "API"
    "IntelliSense"
("EXWM" "axwm")
("Emacs's" "emax's")
("BIDI" "bd")
("Perso-Arabic" "personal arabic")
"Persian"
"URL"
"HTML"
    ("vdo.ninja" "Video Ninja"))
"Commonly-misrecognized words or words that need special capitalization."

```

```

:group 'sachac
:type '(repeat (choice string
↪          (repeat string)))

```

```

;;;###autoload
(defun sachac-subed-add-common-edit (beg end replacement)
  "Add this word to the misrecognized words."
  (interactive
   (let ((beg (if (region-active-p) (min (point) (mark))
                  (skip-syntax-backward "w")
                  (point)))
         (end (if (region-active-p) (max (point) (mark))
                  (save-excursion (forward-word 1) (point)))))
     (list beg end)))
  (completing-read
   (format "Replacement (%s): " (buffer-substring beg end))
   (mapcar (lambda (o) (if (stringp o) o (car o)))
           sachac-subed-common-edits)))
  (customize-set-variable
   'sachac-subed-common-edits
   (cond
    ((member replacement sachac-subed-common-edits)
     (cons (list replacement (buffer-substring-no-properties beg end))
           (delete replacement sachac-subed-common-edits)))
    ((assoc replacement sachac-subed-common-edits)
     (setcdr (assoc replacement sachac-subed-common-edits)
              (append (list replacement) (cdr (assoc replacement sachac-subed-common-edits)))))
    (t
     (push (list replacement (buffer-substring-no-properties beg end))
           sachac-subed-common-edits))))
  (delete-region beg end)
  (insert replacement))

;;;###autoload
(defun sachac-subed-find-next-fix-point ()
  (when (re-search-forward
         (format "\\<%s\\>"
                 (downcase
                  (regexp-opt (seq-mapcat
                              (lambda (o)
                                (if (listp o)
                                    (if (string= (car o) "") (cdr o) o)
                                    (list o)))
                              sachac-subed-common-edits))))
         nil t)
    (goto-char (match-beginning 0))
    (seq-find (lambda (o)
                (if (listp o)
                    (seq-find (lambda (s) (string= (downcase s) (downcase (match-string 0)))) o)
                    (string= (downcase o) (downcase (match-string 0)))))
              sachac-subed-common-edits)))

;;;###autoload

```

```

(defun sacha-subed-fix-common-error ()
  (interactive)
  (let ((entry (sacha-subed-find-next-fix-point)))
    (replace-match (if (listp entry) (car entry) entry) t t)))

;;;###autoload
(defun sacha-subed-fix-common-errors ()
  (interactive)
  (let (done entry correction)
    (while (and
            (not done)
            (setq entry (sacha-subed-find-next-fix-point)))
      (setq correction (if (listp entry) (car entry) entry))
      (if (called-interactively-p 'any)
          (let* ((c (read-char (format "%s (yn.): " correction)))
                 (cond
                  ((= c ?y) (replace-match correction t t))
                  ((= c ?n) (goto-char (match-end 0)))
                  ((= c ?j) (subed-mpv-jump-to-current-subtitle))
                  ((= c ?.) (setq done t))))
            (replace-match correction t t))))))

;;;###autoload
(defun sacha-subed-fix-common-errors-from-start ()
  (goto-char (point-min))
  (sacha-subed-fix-common-errors))

```

9.6.18 Working with media

You can get these from <https://github.com/sachac>

```

(use-package waveform :load-path "~/proj/waveform-el" :if sacha-laptop-p :defer t)
(use-package compile-media :load-path "~/proj/compile-media" :if sacha-laptop-p :defer t
  :autoload compile-media-timestamp-to-msecs
)

```

9.6.19 DONE Working with sections defined by NOTE comments

```

;;;###autoload
(defun sacha-subed-group-sections (subtitles)
  "Return a list of (:comment ... :start-ms ... :stop-ms ... :subtitles ...) ."
  (reverse
   (seq-reduce (lambda (prev val)
                 (if (elt val 4)
                     (cons (list ↵
                             ↵
                             :start-ms (elt val 1) ↵
                             :stop-ms (elt val 2) ↵
                             :subtitles (list val))
                           prev)
                     (when (> (elt val 2) ↵
                               (plist-get (car prev) :stop-ms))

```

```

                                (setcar prev ←
↪ (plist-put (car prev) :stop-ms (elt val 2))))
                                (setcar
                                prev
                                (plist-put ←
↪ (car prev) :subtitles (nconc (plist-get (car prev) :subtitles)
                                ←
↪ (list val))))
                                (list val))))
                                prev))
                                (cdr subtitles)
                                (list
↪ subtitles) 4)
                                (list :comment (elt (car ←
↪ :start-ms (elt (car subtitles) 1)
↪ :stop-ms (elt (car subtitles) 2)
↪ :subtitles (list (car subtitles))))))

(ert-deftest sacha-subed-group-sections ()
  (should
    (equal (sacha-subed-group-sections '((nil 0 99 "Test" "Intro")
↪ (nil 100 199 "A")
↪ (nil 200 299 "B" "Conclusion")
↪ (nil 300 399 "C")
↪ (nil 400 499 "D"))))
    '(:comment "Intro" :start-ms 0 :stop-ms 199
      :subtitles
      ((nil 0 99 ←
↪ "Test" "Intro")
      (nil ←
↪ 100 199 "A")))
      (:comment "Conclusion" :start-ms 200 :stop-ms 499
      :subtitles
      ((nil 200 299 ←
↪ "B" "Conclusion")
      (nil ←
↪ 300 399 "C") (nil 400 499 "D"))))))

;;###autoload
(defun sacha-subed-mark-section ()
  "Return the start and end of the current section.
The current section is defined by NOTE comments."
  (interactive)
  (let* ((start
          (save-excursion
            (if (subed-subtitle-comment)
                (progn ←
↪ (subed-jump-to-subtitle-comment) (point))
                ;; keep going backwards
                (while (and (not (bobp))
↪ (if (subed-backward-subtitle-start-pos)

```

```

↪                                     (not (subed-subtitle-comment))                                     ↪
↪                                     (goto-char (point-min))))                                     ↪
                                     (subed-jump-to-subtitle-comment)
                                     (point))))
                                (end
                                (save-excursion
                                ;; keep going backwards
                                (while (and (not (eobp))
                                ↪
↪      (if (subed-forward-subtitle-start-pos)
                                ↪
↪      (not (subed-jump-to-subtitle-comment))
                                ↪
↪      (goto-char (point-max))))))
                                (subed-jump-to-subtitle-comment))))
                                (when (and start end)
                                (push-mark start)
                                (goto-char end)
                                (activate-mark))))

```

9.6.20 Split up oops better

```

;;;###autoload
(defun sacha-split-oops ()
  "Look for oops and make it easier to split."
  (interactive)
  (let ((scan-window 300))
    (while (re-search-forward "oops[,\\.]?[ \\n]+" nil t)
      (let ((start (min (line-beginning-position) (- (point) scan-window)))
            start-search
            found
            search-for)
        (if (bolp)
            (progn
              (backward-char)
              (setq start (min ↪
↪ (line-beginning-position) (- (point) scan-window))))
            (insert "\\n"))
          (save-excursion
            (setq start-search (point))
            ;; look for 1..5 words back
            (goto-char
              (or
                (cl-loop
                  for n downfrom 5 downto 1
                  do
                    (save-excursion
                      (dotimes (_ n) (forward-word))
                      (setq search-for (downcase ↪
↪ (string-trim (buffer-substring start-search (point)))))
                      (goto-char start-search)
                      (when (re-search-backward ↪
↪ (regexp-quote search-for) start t)
                                (goto-char (match-beginning 0))
                                (cl-return (point))))))

```

```

↪ (point))))
                                (and (call-interactively 'isearch-backward) ↪
                                (insert "\n"))))))

```

```

(setq subed-align-options "task_adjust_boundary_offset_percent=0.5")

```

```

;;;###autoload
(defun sacha-subed-delete-oops (&optional skip-only)
  (interactive (list current-prefix-arg))
  (atomic-change-group
    (subed-for-each-subtitle (point-min) (point-max) t
      (when (string-match "\\boops\\b" (subed-subtitle-text))
        (if skip-only
            (subed-set-subtitle-comment "#+SKIP")
            (subed-kill-subtitle))))))

(ert-deftest sacha-subed-delete-oops ()
  (let ((test '((nil 0 99 "Hello")
                 (nil 100 199 "Hello oops")
                 (nil 200 299 "Hello world")
                 (nil 299 300 "Hello again ↪
↪ oops")))))
    (should
      (equal
        (with-temp-buffer
          (subed-vtt-mode)
          (subed-append-subtitle-list test)
          (sacha-subed-delete-oops)
          (subed-subtitle-list-text (subed-subtitle-list) t))
        "Hello\nHello world\n"))
    (should
      (equal
        (with-temp-buffer
          (subed-vtt-mode)
          (subed-append-subtitle-list test)
          (sacha-subed-delete-oops t)
          (subed-subtitle-list-text (subed-subtitle-list) t))
        "Hello\n\n#+SKIP\n\nHello oops\nHello world\n\n\n#+SKIP\n\nHello again ↪
↪ oops\n"))))

;;;###autoload
(defun sacha-subed-skip-oops ()
  (interactive)
  (sacha-subed-delete-oops t))

;;;###autoload
(defun sacha-subed-record-wpm ()
  (interactive)
  (let ((wpm (subed-wpm
                (seq-remove (lambda (o) (and (elt o 4) ↪
↪
↪ (subed-subtitle-list))))))
    (apply 'message
      "%d wpm (%d words / %.1f minutes)" wpm)))

;;;###autoload

```

```

(defun sacha-subed-prepare-for-cleaning ()
  (interactive)
  (sacha-subed-delete-oops)
  (goto-char (point-min))
  (subed-forward-subtitle-id)
  (subed-set-subtitle-comment (concat "#+OUTPUT: " (file-name-sans-extension ↵
↵ (buffer-file-name)) "-cleaned.opus")))

(defvar sacha-phone-recording-dir "~/sync/Phone")
;;###autoload
(defun sacha-subed-copy-recording (filename destination)
  (interactive
   (list
    (buffer-file-name)
    (file-name-directory
     (read-file-name (format "Copy %s to: "
↵
↵ (file-name-base (buffer-file-name)))
↵ nil nil nil ↵
↵ nil #'file-directory-p))))
  (dolist (ext '("m4a" "txt" "json" "vtt"))
    (when (file-exists-p (concat (file-name-sans-extension filename) "." ext))
      (copy-file (concat (file-name-sans-extension filename) "." ext)
                  destination t)))
    (when (get-file-buffer filename)
      (kill-buffer (get-file-buffer filename))
      (dired destination)))

;;###autoload
(defun sacha-subed-copy-latest-phone-recording (destination)
  "Copy the latest recording transcript and audio to DESTINATION."
  (interactive
   (list
    (file-name-directory
     (read-file-name (format "Move %s to: "
↵
↵ (file-name-base (sacha-latest-file sacha-phone-recording-dir ↵
↵ ".txt"))))
↵ nil nil nil ↵
↵ nil #'file-directory-p))))
  (let ((base (file-name-base (sacha-latest-file sacha-phone-recording-dir ".txt"))))
    (rename-file (expand-file-name (concat base ".txt") sacha-phone-recording-dir)
                  destination)
    (rename-file (expand-file-name (concat base ".m4a") sacha-phone-recording-dir)
                  destination)
    (find-file (expand-file-name (concat base ".txt") destination))
    (save-excursion (sacha-split-oops))
    (goto-char (point-min))
    (flush-lines "^$")
    (goto-char (point-min))
    (subed-forward-subtitle-id)
    (subed-set-subtitle-comment
     (concat "#+OUTPUT: "
↵ (file-name-base (buffer-file-name))
↵ "-cleaned.opus"))))

```


9.6.21 TODO Org Mode: Insert YouTube video with separate captions

emacs

I'm playing around with some ideas for making it easier to post a video with its captions on a webpage or in an Org file so that it's easier to skim or search.

This requires the `yt-dlp` command. I'm also learning how to use `dash.el`'s threading macro, so you'll need to install that as well if you want to run it.

```
;; ##autoload
(defun sacha-msecs-to-timestamp (msecs)
  "Convert MSECs to string in the format HH:MM:SS.MS."
  (concat (format-seconds "%02h:%02m:%02s" (/ msecs 1000))
    "." (format "%03d" (mod msecs 1000))))

(require 'dash)

;; ##autoload
(defun sacha-org-insert-youtube-video-with-transcript (url)
  (interactive "MURL: ")
  (let* ((id (if (string-match "\\(?:v=\\|youtu\\.be\\/\\)\\([^\&]+\\)" url) (match-string 1 url)
    ↪ url))
    (temp-file (make-temp-name "org-youtube-"))
    (temp-file-name (concat temp-file ".en.srv1"))
    data)
    (when (and (call-process "yt-dlp" nil nil nil
      ↪ "--write-sub" "--write-auto-sub" "--no-warnings" "--sub-lang"
      ↪ "en" "--skip-download" "--sub-format" "srv1"
      ↪ "-o" temp-file
      ↪ (format "https://youtube.com/watch?v=%s" id))
      (file-exists-p temp-file-name))
      (insert
        (format "#+begin_export html
<iframe width=\"560\" height=\"315\" src=\"https://www.youtube.com/embed/%s\" title=\"YouTube ↪
↪ video player\" frameborder=\"0\" allow=\"accelerometer; autoplay; clipboard-write; ↪
↪ encrypted-media; gyroscope; picture-in-picture\" ↪
↪ allowfullscreen></iframe>\n#+end_export\n" id)
        "\n"
        (mapconcat (lambda (o)
          (format "| [[https://youtube.com/watch?v=%s&t=%ss][%s]] | %s |\n"
            id
            (dom-attr o 'start)
            (sacha-msecs-to-timestamp (* 1000 (string-to-number (dom-attr o ↪
↪ 'start)))))
          (-> (dom-text o)
            (replace-regexp-in-string "[\n]+" " ")
            (replace-regexp-in-string "&#39;" "'")
            (replace-regexp-in-string "&quot;" "\""))))
        (dom-by-tag (xml-parse-file temp-file-name) 'text)
        ""))
      (delete-file temp-file-name))))
```

It makes an embedded Youtube video and a table with captions below it. The Org file doesn't look too bad, either.

```

#+begin_export html
<iframe width="560" height="315" src="https://www.youtube.com/embed/RuK7lvuyRo" title="YouTube video player" frameborder="0" allow="accelerometer;
autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>
#+end_export

| 00:00:00.000 | I'm Sacha Chua, and welcome to EmacsConf 2020. |
| 00:00:04.000 | To kick things off, here are ten cool things |
| 00:00:07.000 | that people have been working on |
| 00:00:08.000 | since the conference last year. |
| 00:00:10.000 | If you want to follow the links |
| 00:00:11.000 | or if you'd like to add something I've missed, |
| 00:00:14.000 | add them to the collaborative pad |
| 00:00:16.000 | if you're watching this live |
| 00:00:17.000 | or check out the EmacsConf wiki page for this talk. |

```

I decided to stick to standard Org syntax so that I can read it in Emacs too. With the current implementation, clicking on the timestamps jumps to that position in the video, but on the Youtube website. I haven't coded anything fancy like keeping the embedded video at a fixed position, controlling it from the clicks, or highlighting the current position. It's a start, though!

Here's the output of running it with my talk from the last EmacsConf.

```

00:00:00.000 I'm Sacha Chua, and welcome to EmacsConf 2020.
00:00:04.000 To kick things off, here are ten cool things
00:00:07.000 that people have been working on
00:00:08.000 since the conference last year.
00:00:10.000 If you want to follow the links
00:00:11.000 or if you'd like to add something I've missed,
00:00:14.000 add them to the collaborative pad
00:00:16.000 if you're watching this live
00:00:17.000 or check out the EmacsConf wiki page for this talk.

```

... (omitted for brevity)

9.6.22 Export transcript as list

```

(ert-deftest sach-subed-org-format-by-speaker ()
  "Tests `sacha-subed-org-format-by-speaker'."
  (should
    (string=
      (sacha-subed-org-format-by-speaker
        '((nil 0 10 "<v Sacha>This is a test</v>")
          (nil 20 30 "<v Sacha>This is another</v>")
          (nil 40 50 "<v Guest>Next sentence</v>"))
        "Sacha: This is a test\nThis is another\n\nGuest: Next sentence"))))

(defun sacha-subed-org-format-by-speaker (subtitles)
  "Return a string of

Speaker: text ...

Speaker: text ...
"
  (let (last-speaker)
    (replace-regexp-in-string
      "</?i>" ""
      (string-trim
        (mapconcat (lambda (sub)
                      (let ((text (elt sub 3)))
                        (if (string-match "<v \\([^\>]+\>\\)(\\.+\\)</v>" text)
                            (if (not (string= last-speaker (match-string 1 text)))
                                (progn
                                  (setq last-speaker (match-string 1 text))
                                  (format "\n%s: %s" last-speaker
                                          (replace-regexp-in-string

```

```

      (save-match-data (concat "^" (regexp-quote
        ↪ last-speaker) ": ") ""
      (match-string 2 text))))
      (match-string 2 text))
      text)))
      subtitles
      "\n")))))

;;;###autoload
(defun sacha-subed-as-org-list-with-times (file &key from to)
  (interactive "FVT: ")
  (when (stringp from) (setq from (compile-media-timestamp-to-msecs from)))
  (when (stringp to) (setq to (compile-media-timestamp-to-msecs to)))
  (let ((s (mapconcat
    (lambda (o)
      (let ((text (org-ascii--indent-string
        (sacha-subed-org-format-by-speaker
          (plist-get o :subtitles))
          2)))
        (format "- @html:<span ↪
        ↪ class=\"audio-time\" data-start=\"%0.3f\" data-stop=\"%0.3f\">%s</span>@: *%s*\n%s\n\n"
        ↪ (plist-get o :start-ms) 1000.0)
        ↪ (plist-get o :stop-ms) 1000.0)
        ↪ (replace-regexp-in-string "^00:0?\\|\\.\\.[0-9]+$" "" (sacha-msecs-to-timestamp
        ↪ (plist-get o :start-ms)))
        ↪ (plist-get o
        ↪ :comment)
        text)))
    (sacha-subed-group-sections
      (seq-filter (lambda (sub)
        ↪ (and (or (not from) (≥ (elt sub 1) from))
        ↪ (or (not to) (< (elt sub 2) to))))
        ↪ (subed-parse-file file)))
    "")))
  (if (called-interactively-p 'any)
    (insert s)
    s)))

```

9.6.23 Removing gaps and merging subtitles

```

;;;###autoload
(defun sacha-subed-remove-gaps (&optional threshold)
  "Remove gaps between cues below threshold.
If threshold is 0, remove all gaps."
  (interactive "NThreshold: ")
  (goto-char (point-min))
  (unless (subed-jump-to-subtitle-time-start)
    (subed-forward-subtitle-time-start))
  (subed-set-subtitle-time-start 0)
  (let (last-start)

```

```

        (subed-for-each-subtitle (point) (point-max) t
          (if (and last-start (< (- last-start (subed-subtitle-msecs-stop)) ←
            ↪ threshold))
              (subed-set-subtitle-time-stop (1- last-start))
              (setq last-start (subed-subtitle-msecs-start))))))
;;###autoload
(defun sacha-subed-merge-to-min-length (threshold)
  "Merge cues until the duration is at least THRESHOLD."
  (interactive "NThreshold (msecs): ")
  (goto-char (point-min))
  (while (not (eobp))
    (subed-jump-to-subtitle-text)
    (while (not (eobp))
      (let ((duration (- (subed-subtitle-msecs-stop)
        ↪ (subed-subtitle-msecs-start)))
        ↪ (next-duration (save-excursion
        ↪ (when (subed-forward-subtitle-start-pos)
        ↪ (- (subed-subtitle-msecs-stop)
        ↪ (subed-subtitle-msecs-start))))))
        (while (and next-duration (< (+ duration next-duration) ←
        ↪ threshold))
          (subed-merge-with-next)
          (setq duration (- (subed-subtitle-msecs-stop)
        ↪ (subed-subtitle-msecs-start))
        ↪ next-duration (save-excursion
        ↪ (when (subed-forward-subtitle-start-pos)
        ↪ (- (subed-subtitle-msecs-stop)
        ↪ (subed-subtitle-msecs-start))))))
        ↪ (subed-subtitle-msecs-start))))))
    (unless (subed-forward-subtitle-start-pos)
      (goto-char (point-max))))))

```

9.6.24 Using scripts to correct transcripts

I usually write my scripts with phrases that could be turned into the subtitles. I figured I might as well combine that information with the whisperX transcripts which I use to cut out my false starts and oopses. To do that, I use the `string-distance` function, which calculates how similar strings are, based on the Levenshtein algorithm. If I take each line of the script and compare it with the list of words in the transcription, I can add one transcribed word at a time, until I find the number with the minimum distance from my current script phrase. This lets me approximately match strings despite misrecognized words. I use oopses to signal mistakes. When I detect those, I look for the previous script line that is closest to the words I restart with. I can then skip the previous lines automatically. When the script and the transcript are close, I can automatically correct the words. If not, I can use comments to easily compare them at that point. Even though I haven't optimized anything, it runs well enough for my short videos. With these subtitles as a base, I can get timestamps with `subed-align` and then there's just the matter of tweaking the times and adding the visuals.

```
;; (sacha-combine-script-and-transcript '("I have a script" "that's broken up" "into ↵
↵ phrases.") (split-string "I have, oops, I have a script oops. I have a script that's ↵
↵ broken up in to faces." " ") "\\<oops\\>")
;; (sacha-combine-script-and-transcript '("I already talk quickly," "so I'm not going to ↵
↵ speed that up" "into phrases.") (split-string "I already talk pretty quickly. Oops. I ↵
↵ already talk quickly, so I'm not going to speed that up, but I can trim the pauses in ↵
↵ between phrases,"))
;; (subed-word-data-find-approximate-match "I already talk quickly" (split-string "I already ↵
↵ talk pretty quickly oops I already talk quickly" " "))
```

9.7 Creating

See also [subed](#) (p. 463)

9.7.1 Prepare for EmacsConf screenshots or recordings

```
;; ##autoload
(defun sacha-emacsconf-prepare-for-screenshots ()
  (interactive)
  (shell-command "xrandr --output LVDS-1 --mode 1280x720")
  (modus-themes-load-theme 'modus-operandi-tinted)
  (sacha-hl-sexp-update-overlay)
  (set-face-attribute 'default nil :height 170)
  (keycast-header-line-mode))

;; ##autoload
(defun sacha-emacsconf-back-to-normal ()
  (interactive)
  (shell-command "xrandr --output LVDS-1 --mode 1366x768")
  (modus-themes-load-theme (car modus-themes-to-toggle))
  (sacha-hl-sexp-update-overlay)
  (set-face-attribute 'default nil :height 115)
  (keycast-header-line-mode -1))
```

9.7.2 Preparing to record YouTube shorts

```
;; ##autoload
(defun sacha-youtube-prepare-for-shorts ()
  (interactive)
  (keycast-header-line-mode 1)
  (modus-themes-load-theme (car modus-themes-to-toggle))
  (setq compile-media-output-video-width 1080
        compile-media-output-video-height 1920
        compile-media-output-video-fps 30)
  (shell-command "wmctrl -r :ACTIVE: -e 0,300,0,554,984"))
```

```
;; ##autoload
(defun sacha-prepare-for-landscape ()
  (let ((width 6) (height 9))
    (setq compile-media-output-video-width 1080
          compile-media-output-video-height 1920
          compile-media-output-video-fps 30)
    (shell-command "wmctrl -r :ACTIVE: -e 0,300,0,554,984"))
```

9.8 Working with or renaming a set of files

```

;;;###autoload
(defun sachafile-set (file)
  (let ((base (file-name-base file)))
    (seq-filter (lambda (o) (string= base (file-name-base o)))
                 (directory-files (or (file-name-directory file) default-directory) t))))

;;;###autoload
(defun sacharefresh-dired-buffers (dir)
  (dolist (buf (match-buffers '(derived-mode . dired-mode)))
    (when (string= default-directory dir)
      (revert-buffer))))

;;;###autoload
(defun sachadelete-file-set (file)
  (interactive (list (read-file-name
                      "File: "
                      nil
                      (if (derived-mode-p 'dired-mode)
                          (dired-get-filename)
                          (buffer-file-name))))))
  (mapc #'delete-file (sachafile-set file))
  (when (and (buffer-file-name)
              (string= (file-name-base (buffer-file-name))
                       (file-name-base file)))
    (let ((buffer-modified-p nil))
      (kill-buffer))
    (sacharefresh-dired-buffers (file-name-directory file))))

;;;###autoload
(defun sachadelete-current-file-set ()
  (interactive)
  (sachadelete-file-set (if (derived-mode-p 'dired-mode)
                             (dired-get-filename)
                             (buffer-file-name))))

;;;###autoload
(defun sacharename-file-set (file new-prefix &optional force)
  (interactive (let ((file (read-file-name "File: ")))
                 (list file
                       (read-file-name
                        (format "New prefix (%s): "
                               (file-name-base file))
                        nil
                        (if (derived-mode-p 'dired-mode)
                            (dired-get-filename)
                            (buffer-file-name)))))))

```

```

                                current-prefix-arg)))
  (unless force
    (dolist (file (sacha-file-set file))
      (let ((new-file (concat
                                ↵
↵ new-prefix
                                ↵
                                ".")
                                ↵
↵ (file-name-extension file))))
        (when (and (not (string= file new-file))
                    (file-exists-p ↵
↵ new-file))
          (error "%s already exists."
                  new-file))))
    (dolist (file (sacha-file-set file))
      (let ((new-file (expand-file-name
                                ↵
↵ new-prefix
                                ↵
                                ".")
                                ↵
↵ (file-name-extension file))))
        (when (not (string= file new-file))
          (rename-file file new-file t)))
      (sacha-refresh-dired-buffers (file-name-directory file))
      (concat
        new-prefix
        ".")
      (file-name-extension file)))
;;###autoload
(defun sacha-rename-current-file-set (new-name)
  (interactive (list (read-string "New name: "
                                ↵
↵ (file-name-base
                                ↵
↵ (if (derived-mode-p 'dired-mode)
                                ↵
↵ (dired-get-filename)
                                ↵
↵ (buffer-file-name))))))
  (sacha-rename-file-set (if (derived-mode-p 'dired-mode)
                              ↵
↵ (dired-get-filename)
                              ↵
↵ (buffer-file-name))
    new-name))
;;###autoload
(defun sacha-move-current-file-set (new-dir)
  (interactive (list (read-file-name "Destination: " "~/proj/" nil t nil 'file-directory-p)))
  (dolist (file (sacha-file-set
                  (if (derived-mode-p 'dired-mode)
                      ↵
↵ (dired-get-filename)
                      (buffer-file-name))))
    (rename-file file (expand-file-name (file-name-nondirectory file) new-dir)))

```

9.9 Elfeed

Haven't quite figured out how to get this working. I have `elfeed-protocol-feeds` set in my `.emacs.secrets` file.

```
(use-package elfeed :defer t)
(use-package elfeed-protocol
  :after elfeed
  :defer t
  :custom
  (elfeed-use-curl nil)
  (elfeed-curl-extra-arguments '("--insecure"))
  (elfeed-protocol-enabled-protocols '(fever newsblur owncloud ttrss))
  (elfeed-protocol-fever-update-unread-only nil)
  (elfeed-protocol-fever-fetch-category-as-tag t)
  (elfeed-protocol-fever-maxsize 5)
  (elfeed-protocol-log-trace t)
  (elfeed-log-level 'debug)
  :config
  (elfeed-protocol-enable))
```

```
(use-package elfeed-tube
  :defer t
  :vc (:url "https://github.com/karthink/elfeed-tube")
  :after elfeed
  :commands
  (elfeed-tube-fetch)
  :config
  ;; (setq elfeed-tube-auto-save-p nil) ;; t is auto-save (not default)
  ;; (setq elfeed-tube-auto-fetch-p t) ;; t is auto-fetch (default)
  (elfeed-tube-setup)
  :bind (:map elfeed-show-mode-map
    ("F" . elfeed-tube-fetch)
    ([remap save-buffer] . elfeed-tube-save)
    :map elfeed-search-mode-map
    ("F" . elfeed-tube-fetch)
    ([remap save-buffer] . elfeed-tube-save)))
(use-package elfeed-tube-mpv
  :vc (:url "https://github.com/karthink/elfeed-tube")
  :bind (:map elfeed-show-mode-map
    ("C-c C-f" . elfeed-tube-mpv-follow-mode)
    ("C-c C-w" . elfeed-tube-mpv-where)))
```

```
(use-package emms
  :defer t
  :config
  (require 'emms-player-simple)
  (require 'emms-source-file)
  (require 'emms-source-playlist)
  (require 'emms-player-mpv)
  (require 'emms-info-native)
  (require 'emms-info-exiftool)
  (emms-all)
  (add-to-list 'emms-info-functions 'emms-info-native)
  (add-to-list 'emms-info-functions 'emms-info-exiftool)

  (setq emms-player-list '(emms-player-mpv)))
```



```
(-filter #'file-executable-p (directory-files directory t))))))
```

```
(use-package dash
  :config
  (sacha-convert-shell-scripts-to-interactive-commands "~/bin"))
```

Let's see how that goes!

10.3 CSVs

```
(use-package pcsv :defer t)
```

10.4 Whitespace

```
(use-package ws-butler
  :config (ws-butler-global-mode))
```

10.5 Python

```
(use-package elpy
  :defer t
  :config
  (elpy-enable)
  (setq python-shell-interpreter "ipython3"
        python-shell-interpreter-args "-i --simple-prompt")
  (setq python-indent-offset 4)
  (add-hook 'python-mode-hook
    (lambda ()
      (setq-local tab-width 4)
      (setq-local python-flymake-command '("flake8" ↵
↵ "--append-config" "/home/sacha/.config/flake8" "-"))
      (setq-local python-check-command "flake8 --append-config ↵
↵ /home/sacha/.config/flake8"))
      70)
    )
  ;; (use-package lsp-pyright
  ;;   :ensure t
  ;;   :hook (python-mode . (lambda ()
  ;;                         (require 'lsp-pyright)
  ;;                         (lsp))))
  (require 'ansi-color)
  (add-hook 'compilation-filter-hook 'colorize-compilation-buffer))
```

```
;;;###autoload
(defun colorize-compilation-buffer ()
  (when (eq major-mode 'compilation-mode)
    (let ((inhibit-read-only t))
      (ansi-color-apply-on-region compilation-filter-start (point-max)))))
```

10.6 Web development

From [Mixed content HTML files in Emacs web-mode with context-aware completion and documentation](#):

```
(use-package tide :defer t)
(use-package css-eldoc :defer t)
```

```
;; ###autoload
(defun themkat/activate-tide ()
  (interactive)
  (tide-setup)
  (eldoc-mode 1)
  (tide-hl-identifier-mode 1))

;; ###autoload
(defun themkat/complete-web-mode ()
  (interactive)
  (let ((current-scope (web-mode-language-at-pos (point))))
    (cond ((string-equal "javascript" current-scope)
           (company-tide 'interactive))
          ((string-equal "css" current-scope)
           (company-css 'interactive))
          (t
           (company-dabbrev-code 'interactive)))))

;; ###autoload
(defun themkat/eldoc-web-mode ()
  (let ((current-scope (web-mode-language-at-pos (point))))
    (cond ((string-equal "javascript" current-scope)
           (tide-eldoc-function))
          ((string-equal "css" current-scope)
           (css-eldoc-function))
          (t
           nil))))

;; ###autoload
(defun themkat/setup-web-mode-mixed ()
  (web-mode)
  (themkat/activate-tide)
  (setq-local eldoc-documentation-function #'themkat/eldoc-web-mode))
```

```
(defvar sachacopy-append "" "String to append.")
;; ###autoload
(defun sachacopy-and-append (beg end string)
  (interactive (list (if (region-active-p) (region-beginning) (point-min))
                    (if (region-active-p) (region-end) (point-max))
                    (if (region-active-p) (current-prefix-arg)
                        (read-string "Append: "))))
  (sachacopy-append))
(setq sachacopy-append string)
(kill-new (concat (buffer-substring beg end) string))
```

```
;; ##autoload
(defun sachareplace-buffer-with-clipboard ()
  (interactive)
  (erase-buffer)
  (insert (car kill-ring)))
```

```
;; from FAQ at http://web-mode.org/ for smartparens
;; ###autoload
(defun sacha-sp-web-mode-is-code-context (id action context)
  (when (and (eq action 'insert)
              (not (or (get-text-property (point) 'part-side)
                       (get-text-property (point) 'block-side))))
    t))
```

```
; ; Avoid lockfiles because they mess up React projects
(when sachal-laptop-p
  (setq create-lockfiles nil))

(use-package web-mode
  :if sachal-laptop-p
  :mode "\\(\\.html?\\|\\.njk\\)\\'"
  :custom
  (web-mode-enable-current-element-highlight t)
  (web-mode-markup-indent-offset 2)
  (web-mode-code-indent-offset 2)
  (web-mode-enable-auto-pairing nil)
  (web-mode-ac-sources-alist
   '(("css" . (ac-source-css-property))
      ("html" . (ac-source-words-in-buffer ac-source-abbrev))))
  :mode
  ((("\\.html?$" . themkat/setup-web-mode-mixed))
   :bind
   ("C-c RET" . themkat/complete-web-mode)
   ("C-c C-r" . sachal-copy-and-append)))
```

10.7 LSP

<https://emacs-lsp.github.io/lsp-mode/tutorials/reactjs-tutorial/> <https://www.mattduck.com/lsp-python-getting-started.html>

```
;; ##autoload
(defun sacha-local-lsp ()
  (unless (file-remote-p default-directory)
    (lsp)))
```

```
(use-package lsp-mode
  :if sachalaptop-p
  :config
  (setq lsp-headerline-breadcrumb-enable t
        gc-cons-threshold (* 100 1024 1024)
        read-process-output-max (* 1024 1024)
        company-idle-delay 0.5
        company-minimum-prefix-length 1
```

```

        create-lockfiles nil ;; lock files will kill `npm start'
        lsp-enable-file-watchers nil
        lsp-auto-register-remote-clients nil
    )
    (lsp-register-custom-settings
      '(("pyls.plugins.pyls_mypy.enabled" t t)
        ("pyls.plugins.pyls_mypy.live_mode" nil t)
        ("pyls.plugins.pyls_black.enabled" t t)
        ("pyls.plugins.pyls_isort.enabled" t t)))
      (add-to-list 'lsp-file-watch-ignored-directories "/blog\\'")
      (add-to-list 'lsp-file-watch-ignored-directories "/_site\\'")
      (add-to-list 'lsp-file-watch-ignored-directories "/_local\\'")
      :hook ((js-mode . sachalocal-lsp)
              (python-mode . sachalocal-lsp)
              (lsp-mode-hook . lsp-enable-which-key-integration)))
    (use-package lsp-ui
      :if sachalaptop-p
      :commands lsp-ui-mode
      :after lsp-mode)
    (use-package dap-mode
      :if sachalaptop-p
      :after lsp-mode)

```

10.8 Turbo log

```

(use-package tree-sitter-langs
  :ensure t
  :defer t)

(use-package tree-sitter
  :ensure t
  :after tree-sitter-langs
  :config
  (global-tree-sitter-mode))

(use-package turbo-log
  :vc (:url "https://github.com/Artawower/turbo-log")
  :bind (("C-s-l" . turbo-log-print)
         ("C-s-i" . turbo-log-print-immediately)
         ("C-s-h" . turbo-log-comment-all-logs)
         ("C-s-s" . turbo-log-uncomment-all-logs)
         ("C-s-[" . turbo-log-paste-as-logger)
         ("C-s-]" . turbo-log-paste-as-logger-immediately)
         ("C-s-d" . turbo-log-delete-all-logs))
  :config
  (setq turbo-log-msg-format-template "%\": %s\\")
  (setq turbo-log-allow-insert-without-tree-sitter-p t))

```

10.9 Tab width of 2 is compact and readable

```
(setq-default tab-width 2)
```

10.10 More indentation things

From <https://github.com/purcell/emacs.d/blob/master/lisp/init-editing-utils.el>

```
(defun sanityinc/kill-back-to-indentation ()
  "Kill from point back to the first non-whitespace character on the line.

From https://github.com/purcell/emacs.d/blob/master/lisp/init-editing-utils.el"
  (interactive)
  (let ((prev-pos (point)))
    (back-to-indentation)
    (kill-region (point) prev-pos)))
```

```
(keymap-global-set "C-M-<backspace>" 'sanityinc/kill-back-to-indentation)
```

10.11 Alignment

From <https://blog.lambda.cx/posts/emacs-align-columns/>

```
;; ##autoload
(defun sach-align-non-space (beg end)
  "Align non-space columns in region BEG END."
  (interactive "r")
  (align-regexp beg end "\\(\\s-*\\)\\S-+" 1 1 t))
```

10.12 YAML

```
(use-package yaml-mode
  :if sach-laptop-p
  :mode "\\\\.yaml\\'")
```

10.13 Expand region with expreg

This is something I have to get the hang of too. It gradually expands the selection. Handy for Emacs Lisp.

```
(use-package expreg
  :defer t
  :bind
    ("C-=" . expreg-expand)
    ("C-+" . expreg-contract)
    ("C-<prior>" . expreg-expand)
    ("C-<next>" . expreg-contract))
```

10.14 Compilation

```
(eval-after-load 'python-mode
  '(keymap-set python-mode-map "C-c C-c" 'compile))
```

10.15 Emacs Lisp

Autocompile, but don't interrupt me with native compilation warnings.

```
(use-package auto-compile
  :if sachalaptop-p
  :config (auto-compile-on-load-mode)
           :defer t)
(setq native-comp-async-report-warnings-errors 'silent)
```

```
;;;###autoload
(defun sachaset-sentence-end-double-space ()
  (setq-local sentence-end-double-space t))
```

```
(setq eval-expression-print-length nil)
(setq print-length nil)
(setq edebug-print-length nil)
(add-hook 'emacs-lisp-mode-hook
          'sachaset-sentence-end-double-space)
```

<https://emacsredux.com/blog/2026/04/07/stealing-from-the-best-emacs-configs/>

```
(setq reb-re-syntax 'string)
```

```
(use-package which-func)
```

```
(use-package let-completion :vc (:url "https://github.com/gggion/let-completion.el")
  :hook (emacs-lisp-mode . let-completion-mode))
```

10.15.1 Prefix for writing functions

```
(defvar sachafunction-prefix "sacha")
;;;###autoload
(defun sachafunction-prefix ()
  (if (and (buffer-file-name) (string-match "\\\\.el\\\\" (buffer-file-name)))
      (concat (file-name-base (buffer-file-name)) ".")
      sachafunction-prefix))
```

```
(setq sachafunction-prefix "sacha")
```

10.15.2 Easily override existing functions

```
;;;###autoload
(defun sachaovertide-function (symbol)
  (interactive (list (completing-read
    ↵ "Function: "
    ↵
    ↵ #'help--symbol-completion-table
    #'fboundp
```

```

↩ 'confirm nil nil)))
  (let (function-body function-name)
    (save-window-excursion
      (find-function (intern symbol))
      (setq function-name (lisp-current-defun-name))
      (setq function-body (buffer-substring (point)

↩
↩
      (progn (forward-sexp) (point))))))
    (save-excursion
      (insert function-body (format "\n\n(advice-add '%s :override ↩
↩ #'sacha-%s)\n" function-name function-name)))
    (save-excursion
      (forward-char 1)
      (forward-sexp 1)
      (skip-syntax-forward " ")
      (insert "sacha-")
      (forward-sexp 1)
      (skip-syntax-forward " ")
      (forward-char 1))))

```

10.15.3 Lispy

```
(use-package lispy :hook (emacs-lisp-mode . lispy-mode))
```

Might need to tweak it because I use the Dvorak layout, so hjkl doesn't make as much sense for me.

```

(with-eval-after-load 'lispy
  (advice-add
    'lispy-tab
    :around
    (lambda (fn &rest args)
      (let ((print-length nil)
            (print-level nil))
        (apply fn args)))))

```

SOMEDAY Keep track of the number of times specified commands have been called Skip this for now

```

(use-package keyfreq
  :after lispy
  :config
  (keyfreq-mode 1)
  (keyfreq-autosave-mode 1)
  (setq keyfreq-excluded-commands
    '(self-insert-command
      forward-char
      backward-char
      previous-line
      next-line))
  :commands keyfreq-mode
  ;; :hook
  ;; (lispy-mode . keyfreq-mode)
  ;; (lispy-mode . keyfreq-autosave-mode)

```



```
;; :config
;; (defvar sach-keyfreq-included-commands (seq-filter (lambda (sym)
;;             (and (commandp sym)
;;                  (string-match "^lisp-" (symbol-name sym))))
;;             obarray))
;; (advice-add 'keyfreq-pre-command-hook :around
;;             (lambda (orig-fun)
;;               "Limit to `sacha-keyfreq-included-commands'."
;;               (let ((command real-last-command) count)
;;                 (when (and command (symbolp command)
;;                             (memq command sach-keyfreq-included-commands))
;;                   (funcall orig-fun))))
;;             (list :name "track-lisp"))
)
```

Emacs: Making a hydra cheatsheet for Lispy

EMACS

I wanted to get the hang of Lispy thanks to Leo Vivier's presentation at EmacsSF, but there are [a lot of keyboard shortcuts to explore](#). In [Karl Voit's demo of Org Mode at GLT21](#), he showed how he uses Hydra to make cheat sheets. That makes perfect sense, of course, as Hydra can display text and allow you to run commands while the text is displayed. I wanted to make a Hydra that would show me categorized commands to make it easier to look up and eventually remember them. I also wanted to skip the commands that I already knew or that I didn't want to focus on just yet.

Fortunately, the function reference had a link to [the Org file used to generate it](#). I copied the tables, merged them together, named them with `#+NAME: bindings`, replaced the links with plain text, and added a third column with the category I wanted to put commands into.

nil

I wrote this Emacs Lisp code with the header arguments `#+begin_src emacs-lisp :var bindings=bindings :colnames yes`:

```
(eval
 (append
  '(defhydra sach-lispy-cheat-sheet (:hint nil :foreign-keys run)
    ("<f14>" nil "Exit" :exit t))
  (cl-loop for x in bindings
    unless (string= "" (elt x 2))
    collect
    (list (car x)
          (intern (elt x 1))
          (when (string-match "lisp-\\(?:eval-\\)?\\(\\.+\\)"
                              (elt x 1))
            (match-string 1 (elt x 1)))
          :column
          (elt x 2)))))
 (with-eval-after-load 'lispy
  (define-key lispy-mode-map (kbd "<f14>") 'sach-lispy-cheat-sheet/body)
  (define-key lispy-mode-map (kbd "C-?") 'sach-lispy-cheat-sheet/body))
 (with-eval-after-load 'evil-lispy
  (evil-define-key nil evil-lispy-mode-map (kbd "<f14>") 'sach-lispy-cheat-sheet/body)))
```

Here's the result:

I'm experimenting with having my Windows key be **F14** if tapped and **Super_L** if held down. I use KDE, so I [disabled the Applications shortcut](#) with:

```
kwriteconfig5 --file ~/.config/kwinrc --group ModifierOnlyShortcuts --key Meta ""
```

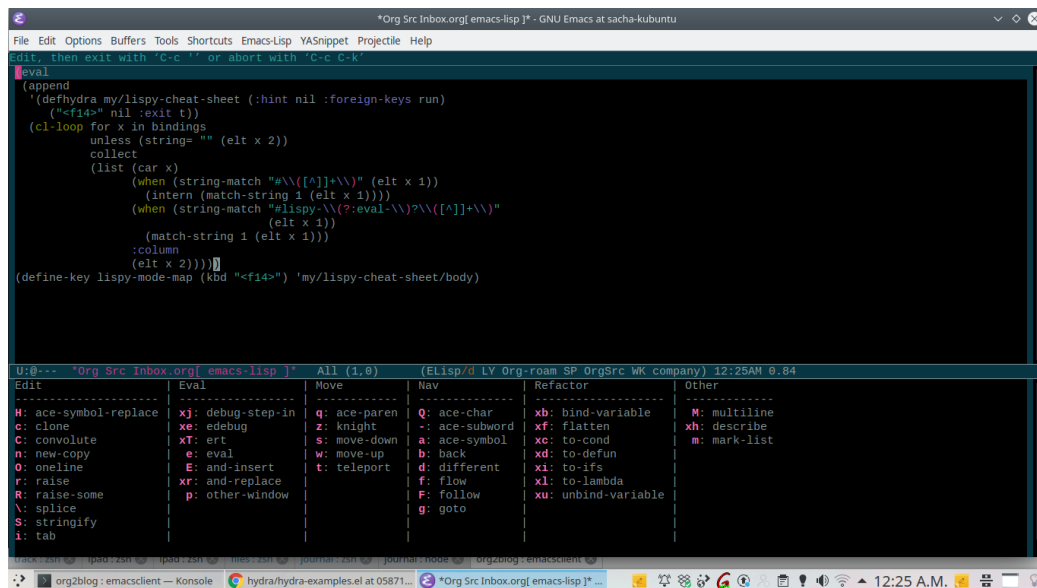


Figure 18: Hydra-based cheat sheet

qdbus org.kde.KWin /KWin reconfigure

and then used `xcap -e 'Super_L=F14'` to make it work.

Looking forward to getting the hang of this!

10.15.4 Smartparens mode

drill

```
(use-package smartparens
  :if saccha-laptop-p
  :config
  (progn
    (require 'smartparens-config)
    (add-hook 'emacs-lisp-mode-hook 'smartparens-mode)
    (add-hook 'emacs-lisp-mode-hook 'show-smartparens-mode)

    ;;;;;;;;;;;;;;
    ;; keybinding management

    (define-key sp-keymap (kbd "C-c s r n") 'sp-narrow-to-sexp)
    (define-key sp-keymap (kbd "C-M-f") 'sp-forward-sexp)
    (define-key sp-keymap (kbd "C-M-b") 'sp-backward-sexp)
    (define-key sp-keymap (kbd "C-M-d") 'sp-down-sexp)
    (define-key sp-keymap (kbd "C-M-a") 'sp-backward-down-sexp)
    (define-key sp-keymap (kbd "C-S-a") 'sp-beginning-of-sexp)
    (define-key sp-keymap (kbd "C-S-d") 'sp-end-of-sexp)

    (define-key sp-keymap (kbd "C-M-e") 'sp-up-sexp)
    (define-key emacs-lisp-mode-map (kbd " ") 'sp-up-sexp)
    (define-key sp-keymap (kbd "C-M-u") 'sp-backward-up-sexp)
    (define-key sp-keymap (kbd "C-M-t") 'sp-transpose-sexp)

    (define-key sp-keymap (kbd "C-M-n") 'sp-next-sexp)
    (define-key sp-keymap (kbd "C-M-p") 'sp-previous-sexp)
```

```

(define-key sp-keymap (kbd "C-M-k") 'sp-kill-sexp)
(define-key sp-keymap (kbd "C-M-w") 'sp-copy-sexp)

(define-key sp-keymap (kbd "M-<delete>") 'sp-unwrap-sexp)
(define-key sp-keymap (kbd "M-<backspace>") 'sp-backward-unwrap-sexp)

(define-key sp-keymap (kbd "C-<right>") 'sp-forward-slurp-sexp)
(define-key sp-keymap (kbd "C-<left>") 'sp-forward-barf-sexp)
(define-key sp-keymap (kbd "C-M-<left>") 'sp-backward-slurp-sexp)
(define-key sp-keymap (kbd "C-M-<right>") 'sp-backward-barf-sexp)

(define-key sp-keymap (kbd "M-D") 'sp-splice-sexp)
(define-key sp-keymap (kbd "C-M-<delete>") 'sp-splice-sexp-killing-forward)
(define-key sp-keymap (kbd "C-M-<backspace>") 'sp-splice-sexp-killing-backward)
(define-key sp-keymap (kbd "C-S-<backspace>") 'sp-splice-sexp-killing-around)

(define-key sp-keymap (kbd "C-J") 'sp-select-next-thing-exchange)
(define-key sp-keymap (kbd "C-<left_bracket>") 'sp-select-previous-thing)
(define-key sp-keymap (kbd "C-M-J") 'sp-select-next-thing)

(define-key sp-keymap (kbd "M-F") 'sp-forward-symbol)
(define-key sp-keymap (kbd "M-B") 'sp-backward-symbol)

(define-key sp-keymap (kbd "C-c s t") 'sp-prefix-tag-object)
(define-key sp-keymap (kbd "C-c s p") 'sp-prefix-pair-object)
(define-key sp-keymap (kbd "C-c s c") 'sp-convolute-sexp)
(define-key sp-keymap (kbd "C-c s a") 'sp-absorb-sexp)
(define-key sp-keymap (kbd "C-c s e") 'sp-emit-sexp)
(define-key sp-keymap (kbd "C-c s p") 'sp-add-to-previous-sexp)
(define-key sp-keymap (kbd "C-c s n") 'sp-add-to-next-sexp)
(define-key sp-keymap (kbd "C-c s j") 'sp-join-sexp)
(define-key sp-keymap (kbd "C-c s s") 'sp-split-sexp)

;;
;; pair management

(sp-local-pair 'minibuffer-inactive-mode "" nil :actions nil)
(sp-local-pair 'web-mode "<" nil :when '(sacha-sp-web-mode-is-code-context))

;; markdown-mode
(sp-with-modes '(markdown-mode gfm-mode rst-mode)
  (sp-local-pair "*" "*" :bind "C-✱")
  (sp-local-tag "2" "✱✱" "✱✱")
  (sp-local-tag "s" "``scheme" "``")
  (sp-local-tag "<" "<_" "</_>" :transform 'sp-match-sgml-tags))

;; tex-mode latex-mode
(sp-with-modes '(tex-mode plain-tex-mode latex-mode)
  (sp-local-tag "i" "1d5f8e69396c521f645375107197ea4dfbc7b792quot;<" ↵
    ↵ "1d5f8e69396c521f645375107197ea4dfbc7b792quot;>"))

;; html-mode
(sp-with-modes '(html-mode sgml-mode web-mode)
  (sp-local-pair "<" ">"))

;; lisp modes
(sp-with-modes sp--lisp-modes
  (sp-local-pair "(" nil :bind "C-(")))

```

10.15.5 Edit list

drill:embark

M-x edit-list makes it easier to edit an Emacs Lisp list.

```
(use-package edit-list
  :commands edit-list
  :config
  (with-eval-after-load 'embark
    (define-key embark-variable-map "l" 'edit-list)))
```

10.15.6 General-purpose Emacs Lisp libraries

```
(use-package dash :ensure t)
(use-package s :ensure t)
```

10.15.7 Let's try this setup

Copied from https://www.reddit.com/r/emacs/comments/1051bfu/comment/j38ymkn/?utm_source=reddit&utm_medium=web2x&context=3

```
(with-eval-after-load 'elisp-mode
  (define-key emacs-lisp-mode-map (kbd "C-c C-d C-d") 'describe-function)
  (define-key emacs-lisp-mode-map (kbd "C-c C-d d") 'describe-function)
  (define-key emacs-lisp-mode-map (kbd "C-c C-k") 'eval-buffer))

(use-package highlight-quoted
  :ensure t
  :hook
  (emacs-lisp-mode . highlight-quoted-mode))

(use-package eros
  :ensure t
  :hook
  (emacs-lisp-mode . eros-mode))

(use-package suggest
  :ensure t
  :defer t)

(use-package ipretty
  :defer t
  :ensure t
  :config
  (ipretty-mode 1))

;; Hide package namespaces
(use-package nameless
  :ensure t
  :after 'lispy
  :hook
  (emacs-lisp-mode . nameless-mode)
  :bind
  (:map emacs-lisp-mode-map
    ("C-c -" . nameless-insert-name)
    ("_" . nameless-insert-name-or-self-insert)
  :map lispy-mode-map
```

```

      ("_" . nameless-insert-name-or-self-insert))
:custom
(nameless-global-aliases '())
(nameless-private-prefix t))

(use-package erefactor
  :ensure t
  :defer t)

;; Emacs Lisp Static Analyzer
(use-package elsa
  :defer t
  :ensure t)

```

10.15.8 Edebug

From <https://xenodium.com/inline-previous-result-and-why-you-should-edebug/>

```

(require 'eros)
;;;###autoload
(defun advised:edebug-previous-result (_ &rest r)
  "Advised `edebug-previous-result'."
  (eros--make-result-overlay edebug-previous-result
    :where (point)
    :duration eros-eval-result-duration))

;;;###autoload
(defun advised:edebug-compute-previous-result (_ &rest r)
  "Advised `edebug-compute-previous-result'."
  (let ((previous-value (nth 0 r)))
    (if edebug-unwrap-results
      (setq previous-value
            (edebug-unwrap* previous-value)))
    (setq edebug-previous-result
          (edebug-safe-prin1-to-string previous-value))))

```

```

(advice-add #'edebug-previous-result
  :around
  #'advised:edebug-previous-result)
(advice-add #'edebug-compute-previous-result
  :around
  #'advised:edebug-compute-previous-result)

```

10.15.9 Testing

```

(use-package buttercup
  :hook '(buttercup-minor-mode . sacha-buttercup-set-up-imenu))
(use-package bug-hunter
  :load-path "~/vendor/elisp-bug-hunter")
(use-package package-lint :defer t)

```

ERT From <https://www.reddit.com/r/emacs/comments/1fub66z/comment/lpyv8no/>:

```

;;;###autoload

```

```

(defun sacha-eval-buf-and-run-ert-test-at-point ()
  "Evaluate the current buffer and run the ERT test at point."
  (interactive)
  (save-excursion
    (beginning-of-defun)
    (unless (looking-at "(ert-deftest\\s-+")
      (user-error "Not at an ERT test"))
    (goto-char (match-end 0))
    (let ((test-name (thing-at-point 'symbol)))
      (unless test-name
        (user-error "Couldn't get ERT test name"))
      (eval-buffer)
      (ert-run-tests-interactively test-name))))

;;;###autoload
(defun sacha-ert-visit-test-file (&optional base)
  (let* ((base (or base (buffer-file-name)))
    (filename
      (and base
        (seq-find
          #'file-exists-p
          (list
            (expand-file-name
              (concat "tests/" (file-name-base) "-test.el")
              (file-name-directory base))
            (expand-file-name
              (concat "../tests/" (file-name-base) "-test.el")
              (file-name-directory base))
            (concat (file-name-sans-extension base) "-test.el"))))))
    (when filename
      (find-file filename))))

;;;###autoload
(defun sacha-ert-find-insert-point (func-name)
  (goto-char (point-min))
  (cond
    ((re-search-forward (format "(ert-deftest %s"
      (if (symbolp func-name)
        (regexp-quote (symbol-name func-name))
        func-name)) nil t)
      (goto-char (match-beginning 0)))
    ((re-search-forward ";;; Code" nil t)
      (forward-line))
    ((re-search-forward "^$" nil t)
      (t (goto-char (point-max)))))

(declare-function which-function "which-function")
(require 'which-func)
;;;###autoload
(defun sacha-ert-deftest-from-function-at-point ()
  "Create an ERT test template for the function at point."
  (interactive)
  (let* ((func (which-function))
    (test-def (and func
      (format
        "(ert-deftest %s ()
\nTests `%s'.\n
(should (equal (%s) t)))\n\n"
        func func func))))))

```

```

(if (not func)
  (error "No function at point")
  ;; Open the test file
  (sacha-ert-visit-test-file)
  (sacha-ert-find-insert-point func)
  (insert test-def)
  (backward-char 7))))

```

```

(use-package ert
  :defer t
  :commands ert
  :config
  ;; handle truncated lists
  (advice-add 'ert--pp-with-indentation-and-newline
    :around (lambda (oldfunc &rest args) ↵
    ↵ (condition-case nil (apply oldfunc args) (error nil))))
  :bind
  (:map
    emacs-lisp-mode-map ("C-c C-t" . #'sacha-eval-buf-and-run-ert-test-at-point)))

```

Buttercup

```

(defvar sacha-buttercup-source-buffer nil)
(defvar sacha-buttercup-tests nil)
;;;###autoload
(defun sacha-buttercup-track-source ()
  (interactive)
  (setq sacha-buttercup-source-buffer (current-buffer))
  (setq sacha-buttercup-tests (sacha-buttercup-tests-and-positions)))

;;;###autoload
(defun sacha-buttercup-run-dwim ()
  (interactive)
  (let ((lexical-binding t))
    (if buttercup-minor-mode
      (sacha-buttercup-run-closest-at-point)
      (buttercup-run))))

;; (advice-remove 'buttercup-run 'sacha-buttercup-track-source)
;;;###autoload
(defun sacha-buttercup-run-closest-at-point ()
  "Run the buttercup suite at point."
  (interactive)
  (let ((lexical-binding t)
        start)
    (setq buttercup-suites nil)
    (save-selected-window
      (save-excursion
        (save-restriction
          ;; go up until we find a describe form
          (while (not (looking-at ↵
            ↵ "([[:space:]]*describe[[:space:]]+"))
              (backward-up-list nil t))
            (setq start (point))
            (forward-sexp)
            (narrow-to-region start (point))
            (eval-last-sexp nil)
            (sacha-buttercup-track-source))))

```

```

    (buttercup-run))
    (message "Suite executed successfully"))

;;;###autoload
(defun sacha-buttercup-find-test ()
  (interactive)
  (if (re-search-backward (make-string 40 ?=) nil t)
      (progn
        (forward-line)
        (let ((pos (assoc-default (buffer-substring ↵
↵ (line-beginning-position)
↵
↵
↵ (line-end-position))
↵
↵ sacha-buttercup-tests)))
          (when pos
            (pop-to-buffer sacha-buttercup-source-buffer)
            (goto-char pos))))
        (let ((tests (sacha-buttercup-tests-and-positions)))
          (goto-char (assoc-default (completing-read "Test: "
↵
↵
↵ (sacha-presorted-completion-table tests))
↵
↵ tests))))))

;;;###autoload
(defun sacha-buttercup-test-name ()
  (save-excursion
    (let (list)
      (condition-case err
        (progn
          (while (not (bobp))
            (let ((form (save-excursion ↵
↵
↵ (ignore-errors
↵
↵ (read (current-buffer))))))
              (when (listp form) (and ↵
↵ (member (car form) '(describe it)))
                (setq ↵
↵ list (cons (cadr form) list)))
              (backward-up-list nil t)))
            (string-join list " "))
          (error
            (string-join list " "))))))

;;;###autoload
(defun sacha-buttercup-tests-and-positions-lookup ()
  "Return a list of test names and points, for easier jumping."
  ;; This is a very inefficient implementation. I wonder how to walk the tree ...
  (goto-char (point-min))
  (cl-loop while (re-search-forward "[[:space:]]*it[[:space:]]+\"" nil t)
    collect (cons (sacha-buttercup-test-name) (point))))

;;;###autoload
(defun sacha-buttercup-tests-as-tree ()
  "Return the tests as nested lists ending with (description . point)."

```



```

Useful as `imenu-create-index-function'."
  (goto-char (point-min))
  (let (result)
    (condition-case _
      (progn
        (down-list)
        (while (not (eobp))
          (cond
            ((looking-at "describe\\_>")
             (forward-sexp)
             (setq result (cons
                           ↵
                           (cons (read (current-buffer))
                               ↵
                               (save-restriction
                               ↵
                               (narrow-to-region
                               ↵
                               (point)
                               ↵
                               (progn
                               ↵
                               (up-list)
                               ↵
                               (1- (point))))
                               ↵
                               (sacha-buttercup-tests-as-tree)))
                               ↵
                               result))))
            ((looking-at "it\\_>")
             (forward-sexp)
             (setq result (cons
                           ↵
                           (cons (read (current-buffer)) (point))
                               ↵
                               result))
             (up-list)
             (down-list))
            (t
             ;; todo, handle other things
             (up-list)
             (down-list))))))
    (scan-error
     ;; can't go down or forward
     (reverse result))))

;; ###autoload
(defun sacha-buttercup-set-up-imenu ()
  (setq-local imenu-generic-expression nil)
  (setq-local imenu-create-index-function #'sacha-buttercup-tests-as-tree))

;; ###autoload
(defun sacha-buttercup-tests-and-positions ()
  "Return test names and points to jump to."
  (save-excursion
    (goto-char (point-min))
    (condition-case _
      (progn
        (down-list)

```



```

                                                    (progn
↪                                                    ↪
↪        ;; Try to go down. If we can, continue
↪                                                    ↪
↪        ;; processing. If we can't, go up until we
↪                                                    ↪
↪        ;; can go down.
↪                                                    ↪
↪        (while (condition-case _
↪                                                    ↪
↪                (down-list)
↪                                                    ↪
↪                (error t))
↪                                                    ↪
↪                (up-list)
↪                                                    ↪
↪                (setq breadcrumbs (cdr breadcrumbs))))
                                                    (scan-error
↪                                                    (error ↪
↪
↪        (throw 'done (reverse result)))))))))
                                                    (reverse result))
                                                    (error nil))))

(ert-deftest sacha-buttercup-tests-and-positions ()
  (with-temp-buffer
    (insert "(describe \"test\""
      :var ((test))
      (it \"1\")
      (it \"2\")
      (describe \"b\"
        (before-each \"do this\"
          (it \"3\")
          (it \"4\")))
      (describe \"c\"
        (it \"5\")
        (it \"6\")
        (it \"7\")
        (describe \"d\"
          (it \"8\")))
      (describe \"e\"
        (it \"5\")
        (it \"6\")
        (it \"7\")
        (describe \"f\"
          (it \"8\")))
      )))
    (let ((tests (sacha-buttercup-tests-and-positions)))
      (expect (assoc "test 1" tests))
      (expect (assoc "test 2" tests))
      (expect (assoc "test b 3" tests))
      (expect (assoc "test b 4" tests))
      (expect (assoc "test c 5" tests))
      (expect (assoc "test e f 8" tests))))))

```

```
(use-package undercover
  :vc (:url "https://github.com/undercover-el/undercover.el")
  :defer t
  )
(use-package coverage :defer t)
```

10.15.10 Eldoc

Eldoc provides minibuffer hints when working with Emacs Lisp.

```
(use-package eldoc
  :if sachalaptop-p
  :diminish eldoc-mode
  :commands turn-on-eldoc-mode
  :defer t
  :init
  (progn
    (add-hook 'emacs-lisp-mode-hook 'turn-on-eldoc-mode)
    (add-hook 'lisp-interaction-mode-hook 'turn-on-eldoc-mode)
    (add-hook 'ielm-mode-hook 'turn-on-eldoc-mode))
  :config
  (eldoc-add-command-completions "paredit-")
  (eldoc-add-command-completions "lisp-"))
```

Related:

- [Seamlessly Merge Multiple Documentation Sources with Eldoc - Mastering Emacs](#)

```
(add-to-list 'display-buffer-alist
  `((rx bos "*Flycheck errors*" eos)
    (display-buffer-in-side-window)
    (side . bottom)
    (reusable-frames . visible)
    (window-height . 0.33)))
```

```
;; ##autoload
(defun mp-flycheck-eldoc (callback &rest _ignored)
  "Print flycheck messages at point by calling CALLBACK."
  (when-let ((flycheck-errors (and flycheck-mode (flycheck-overlay-errors-at (point)))))
    (mapc
     (lambda (err)
       (funcall callback
        (format "%s: %s"
          (let ((level (flycheck-error-level err)))
            (pcase level
              ('info (propertize "I" 'face 'flycheck-error-list-info))
              ('error (propertize "E" 'face 'flycheck-error-list-error))
              ('warning (propertize "W" 'face 'flycheck-error-list-warning))
              (_ level)))
          (flycheck-error-message err))
        :thing (or (flycheck-error-id err)
                   (flycheck-error-group err))
        :face 'font-lock-doc-face))
       flycheck-errors)))
;; ##autoload
(defun mp-flycheck-prefer-eldoc ()
  (add-hook 'eldoc-documentation-functions #'mp-flycheck-eldoc nil t))
```

```

(setq eldoc-documentation-strategy 'eldoc-documentation-compose-eagerly)
(setq flycheck-display-errors-function nil)
(setq flycheck-help-echo-function nil))

;;;###autoload
(defun mp-eglot-eldoc ()
  (setq eldoc-documentation-strategy
        'eldoc-documentation-compose-eagerly))

```

```

(use-package flycheck
  :if sachalaptop-p
  :hook (flycheck-mode . mp-flycheck-prefer-eldoc)
  :bind (:map flycheck-mode-map
            ("s-n" . flycheck-next-error))
  )
(use-package eglot
  :if sachalaptop-p
  :preface
  ;;;###autoload
  :hook ((eglot-managed-mode . mp-eglot-eldoc)))

```

10.15.11 Refactoring

drill

More things that I need to get used to...

```

;; C-c C-v l : elint current buffer in clean environment.
;; C-c C-v L : elint current buffer by multiple emacs binaries.
;;           See `erefactor-lint-emacsen'
;; C-c C-v r : Rename symbol in current buffer.
;;           Resolve `let' binding as long as i can.
;; C-c C-v R : Rename symbol in requiring modules and current buffer.
;; C-c C-v h : Highlight current symbol in this buffer
;;           and suppress `erefacthr-highlight-mode'.
;; C-c C-v d : Dehighlight all by above command.
;; C-c C-v c : Switch prefix bunch of symbols.
;;           ex: '(hoge-var hoge-func) -> '(foo-var foo-func)
;; C-c C-v ? : Display flymake elint warnings/errors

(use-package erefactor
  :if sachalaptop-p
  :defer t
  :bind (:map emacs-lisp-mode-map ("C-c C-v" . erefactor-map)))

(use-package redshank
  :if sachalaptop-p
  :disabled t
  :defer t
  :init (add-hook 'emacs-lisp-mode-hook 'redshank-mode))

```

10.15.12 Jumping to code

```

(define-key emacs-lisp-mode-map (kbd "C-c .") 'find-function-at-point)

```

10.15.13 YE11: Fix find-function for Emacs Lisp from org-babel or scratchorg:emacs:elisp:stream:YayEmacs

<https://archive.org/download/yay-emacs-11-fix-find-function-for-emacs-lisp-from-org-babel-or-scratch/ye11-find-function.mp4?captions=/home/sacha/proj/yay-emacs/transcripts/2026-04-05-ye11-find-function.vtt&thumbnail=/recordings/2026-04-05-19-25-03.png>

[Watch on Internet Archive](#), [watch/comment on YouTube](#), [download captions](#), or [email me](#)

Where can you define an Emacs Lisp function so that you can use `find-function` to jump to it again later?

- A: In an indirect buffer from Org Mode source block with your favorite eval function like `eval-defun` ([hint](#))
 - `C-c ' (org-edit-special)` inside the block; execute the defun with `C-M-x (eval-defun)`, `C-x C-e (eval-last-sexp)`, or `eval-buffer`.

```
(defun sacha-test-1 () (message "Hello"))
```

- B: In an Org Mode file by executing the block with `C-c C-c` ([hint](#))

```
(defun sacha-test-2 () (message "Hello"))
```

- C: In a `.el` file ([hint](#))

`file:///tmp/test-search-function.el` : execute the defun with `C-M-x (eval-defun)`, `C-x C-e (eval-last-sexp)`, or `eval-buffer`

- D: In a scratch buffer, other temporary buffer, or really any buffer thanks to `eval-last-sexp` ([hint](#))

```
(defun sacha-test-4 () (message "Hello"))
```

Only option C works - it's gotta be in an `.el` file for `find-function` to find it. But I love jumping to function definitions using `find-function` or `lispy-goto-symbol` (which is bound to `M-. if you use lispy and set up lispy-mode`) so that I can look at or change how something works. It can be a little frustrating when I try to jump to a definition and it says, "Don't know where blahblahblah is defined." I just defined it five minutes ago! It's there in one of my other buffers, don't make me look for it myself. Probably this will get fixed in Emacs core someday, but no worries, we can work around it today with a little bit of advice.

I did some digging around in the source code. Turns out that `symbol-file` can't find the function definition in the `load-history` variable if you're not in a `.el` file, so `find-function-search-for-symbol` gets called with `nil` for the library, which causes the error. ([emacs:subr.el](#))

I wrote some advice that searches in any open `emacs-lisp-mode` buffers or in a list of other files, like my Emacs configuration. This is how I activate it:

```
(setq sacha-elisp-find-function-search-extra '("~/sync/emacs/Sacha.org"))  
(advice-add 'find-function-search-for-symbol :around ↵  
  ↵ #'sacha-elisp-find-function-search-for-symbol)
```

Now I should be able to jump to all those functions wherever they're defined.

```
(sacha-test-1)  
(sacha-test-2)  
(sacha-test-3)  
(sacha-test-4)
```

Note that by default, `M-. in emacs-lisp-mode` uses `xref-find-definitions`, which seems to really want files. I haven't figured out a good workaround for that yet, but `lispy-mode` makes `M-. work` and gives me a bunch of other great shortcuts, so I'd recommend checking that out.

Here's the source code for the find function thing:

```
(defvar sachaelispfindfunctionsearchextra
  nil
  "List of filenames to search for functions.")

;;###autoload
(defun sachaelispfindfunctionsearchforsymbol (fn symbol type library &rest _)
  "Find SYMBOL with TYPE in Emacs Lisp buffers or `sachaelispfindfunctionsearchextra'.
  Prioritize buffers that do not have associated files, such as Org Src
  buffers or *scratch*. Note that the fallback search uses \"^([ ])+\" so that
  it isn't confused by preceding forms.

  If LIBRARY is specified, fall back to FN.

  Activate this with:

  (advice-add 'find-function-search-for-symbol
    :around #'sachaelispfindfunctionsearchforsymbol-in-dotemacs)
  (if (null library)
      ;; Could not find library; search sachaelispfindfunctionsearchextra just in case
      (progn
        (while (and (symbolp symbol) (get symbol 'definition-name))
          (setq symbol (get symbol 'definition-name)))
        (catch 'found
          (mapc
            (lambda (buffer-or-file)
              (with-current-buffer (if (bufferp buffer-or-file)
                                      buffer-or-file
                                      (find-file-noselect buffer-or-file)))
                (let* ((regexp-symbol
                      (or (and (symbolp symbol)
                              (alist-get type (get symbol 'find-function-type-alist)))
                          (alist-get type find-function-regexp-alist)))
                  (form-matcher-factory
                    (and (functionp (cdr-safe regexp-symbol))
                        (cdr regexp-symbol)))
                  (regexp-symbol (if form-matcher-factory
                                    (car regexp-symbol)
                                    regexp-symbol)))
                  (case-fold-search)
                  (regexp (if (functionp regexp-symbol) regexp-symbol
                              (format (symbol-value regexp-symbol)
                                      ;; Entry for ` (backquote) macro in loaddefs.el,
                                      ;; (defalias (quote `)...) , has a \ but
                                      ;; (symbol-name symbol) doesn't. Add an
                                      ;; optional \ to catch this.
                                      (concat "\\\\"?
                                              (regexp-quote (symbol-name symbol)))))))
                  (save-restriction
                    (widen)
                    (with-syntax-table emacs-lisp-mode-syntax-table
                      (goto-char (point-min))
                      (if (if (functionp regexp)"))
```

```

      (funcall regexp symbol)
      (or (re-search-forward regexp nil t)
          ;; `regexp' matches definitions using known forms like
          ;; `defun', or `defvar'. But some functions/variables
          ;; are defined using special macros (or functions), so
          ;; if `regexp' can't find the definition, we look for
          ;; something of the form "(SOMETHING <symbol> ...)".
          ;; This fails to distinguish function definitions from
          ;; variable declarations (or even uses thereof), but is
          ;; a good pragmatic fallback.
          (re-search-forward
            (concat "^([^\n])+" find-function-space-re "[']?"
              (regexp-quote (symbol-name symbol))
              "\\_>")
            nil t)))
      (progn
        (beginning-of-line)
        (throw 'found
          (cons (current-buffer) (point))))
      (when-let* ((find-expanded
        (when (trusted-content-p)
          (find-function--search-by-expanding-macros
            (current-buffer) symbol type
            form-matcher-factory))))
        (throw 'found
          (cons (current-buffer)
            find-expanded))))))
      (delq nil
        (append
          (sort
            (match-buffers '(derived-mode . emacs-lisp-mode))
            :key (lambda (o) (or (buffer-file-name o) "")))
          sacha-elisp-find-function-search-extra))))
      (funcall fn symbol type library))

```

I even figured out how to [write tests for it](#):

```

(ert-deftest sacha-elisp--find-function-search-for-symbol--in-buffer ()
  (let ((sym (make-temp-name "--test-fn"))
        buffer)
    (unwind-protect
      (with-temp-buffer
        (emacs-lisp-mode)
        (insert (format ";; Comment\n(defun %s () (message \"Hello\"))" sym))
        (eval-last-sexp nil)
        (setq buffer (current-buffer))
        (with-temp-buffer
          (let ((pos (sacha-elisp-find-function-search-for-symbol nil (intern sym) nil nil)))
            (should (equal (car pos) buffer))
            (should (equal (cdr pos) 12))))
          (fmakunbound (intern sym))))
      (ert-deftest sacha-elisp--find-function-search-for-symbol--in-file ()
        (let* ((sym (make-temp-name "--test-fn"))
              (temp-file (make-temp-file
                "test-" nil ".org"
                (format
                  "#+begin_src emacs-lisp\n;; Comment\n(defun %s () (message ↵

```



```

↪ \("Hello\)"))\n#+end_src"
      sym)))
    (sacha-elisp-find-function-search-extra (list temp-file))
    buffer)
  (unwind-protect
    (with-temp-buffer
      (let ((pos (sacha-elisp-find-function-search-for-symbol nil (intern sym) nil nil)))
        (should (equal (buffer-file-name (car pos)) temp-file))
        (should (equal (cdr pos) 35))))
      (delete-file temp-file))))

```

10.15.14 Sorting

```

;;;###autoload
(defun sacha-sort-sexps-in-region (beg end)
  "Can be handy for sorting out duplicates.
  Sorts the sexps from BEG to END. Leaves the point at where it
  couldn't figure things out (ex: syntax errors)."
  (interactive "r")
  (let ((input (buffer-substring beg end))
        list last-point form result)
    (save-restriction
      (save-excursion
        (narrow-to-region beg end)
        (goto-char (point-min))
        (setq last-point (point-min))
        (setq form t)
        (while (and form (not (eobp)))
          (setq form (ignore-errors (read (current-buffer))))
          (when form
            (add-to-list
              'list
              (cons
                (prin1-to-string form)
                (buffer-substring last-point (point))))
            (setq last-point (point))))
          (setq list (sort list (lambda (a b) (string< (car a) (car b))))))
        (delete-region (point-min) (point))
        (insert (mapconcat 'cdr list "\n")))))

```

10.15.15 Evaluation

Borrowed from Steve Purcell's config. This pretty-prints the results.

```

;;;###autoload
(defun sanityinc/eval-last-sexp-or-region (prefix)
  "Eval region from BEG to END if active, otherwise the last sexp."
  (interactive "P")
  (if (and (mark) (use-region-p))
      (eval-region (min (point) (mark)) (max (point) (mark)))
      (pp-eval-last-sexp prefix)))

```

```

(keymap-global-set "M-:" 'pp-eval-expression)
(keymap-set emacs-lisp-mode-map "C-x C-e" 'sanityinc/eval-last-sexp-or-region)

```

10.15.16 Auto insert

```
(auto-insert-mode)
(with-eval-after-load 'auto-insert
  (add-to-list 'auto-insert-alist
    '(((("\\.el\\") . "Emacs Lisp header")
      "Short description: "
      ";;; " ←
      ↵ (file-name-nondirectory (buffer-file-name)) " — " str
      (make-string (max 2 (- 80 (current-column) 27)) ?\s)
      "-*- lexical-binding: t; -*-" '(setq lexical-binding t)
      "

;; Copyright (C) " (format-time-string "%Y") " "
  (getenv "ORGANIZATION") | (progn user-full-name) "

;; Author: " (user-full-name)
  '(if (search-backward "&" (line-beginning-position) t)
      (replace-match (capitalize (user-login-name)) t t))
  '(end-of-line 1) " <" (progn user-mail-address) ">
  "

;; Keywords and completing-read with a require-match don't give me a way to break out
;; ;; Keywords: "
;;   '(require 'finder)
;;   ;; '(setq v1 (apply 'vector (mapcar 'car finder-known-keywords)))
;;   '(setq v1 (mapcar (lambda (x) (list (symbol-name (car x))))
;;     finder-known-keywords)
;;     v2 (mapconcat (lambda (x) (format "%12s: %s" (car x) (cdr x)))
;;     finder-known-keywords
;;     "\n"))
;;   ((let ((minibuffer-help-form v2))
;;     (completing-read "Keyword, C-h: " v1 nil t))
;;     str " ")
;;   & -2
  "

\\;; This program is free software; you can redistribute it and/or modify
\\;; it under the terms of the GNU General Public License as published by
\\;; the Free Software Foundation, either version 3 of the License, or
\\;; (at your option) any later version.

\\;; This program is distributed in the hope that it will be useful,
\\;; but WITHOUT ANY WARRANTY; without even the implied warranty of
\\;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
\\;; GNU General Public License for more details.

\\;; You should have received a copy of the GNU General Public License
\\;; along with this program. If not, see <https://www.gnu.org/licenses/>.

\\;;; Commentary:

\\;; " _ "

\\;;; Code:

\\(provide '
  (file-name-base (buffer-file-name))
```

```

    ")
\;;; " (file-name-nondirectory (buffer-file-name)) " ends here\n"))))

```

10.15.17 Stubbing

From <https://ag91.github.io/blog/2020/12/31/top-down-elisp-ing-a-simple-snippet-to-stub-a-function-while-you-are-designing-your-code/>

```

;;###autoload
(defun sacha-stub-elisp-defun ()
  "Stub an elisp function from symbol at point."
  (interactive)
  (let* ((fun (thing-at-point 'list 'no-properties)))
    (when fun
      (let* ((fun-list (car (read-from-string fun)))
              (name (symbol-name (nth 0 fun-list)))
              (args (cdr fun-list)))
        (save-excursion
          (or (search-backward "(defun" nil 't) (goto-char (point-min)))
          (insert
            (s-concat
              "(defun "
              name
              " "
              (format "%s" (--map (s-concat "arg" (number-to-string it)) (number-sequence 1 <=
                (length args))))
              "\n  \"SomeDocs\" \n  nil)\n\n"))))))))

```

```

(keymap-set emacs-lisp-mode-map "C-:" #'sacha-stub-elisp-defun)

```

10.15.18 Helpful

```

(use-package helpful
  :bind
  ([remap describe-key] . helpful-key)
  ([remap describe-command] . helpful-command)
  ([remap describe-variable] . helpful-variable)
  ([remap describe-function] . helpful-callable))

```

10.15.19 Looking up help

From [GNU Emacs configuration](#) | Protesilaos Stavrou:

```

(keymap-global-set "C-h K" #'describe-keymap)
(keymap-global-set "C-h c" #'describe-char)

```

10.15.20 elisp-demos

elisp-demos lets you add text to a symbol's help documentation from entries in an Org file. The Org file at <https://github.com/xuchunyang/elisp-demos> has many examples. I've modified my version to allow me to have personal note files and a button to add more examples. My diff: <https://github.com/xuchunyang/elisp-demos/compare/master...sachac:elisp-demos:user-files>

```
(use-package elisp-demos
  :load-path "~/vendor/elisp-demos"
  :commands
  elisp-demos-advice-helpful-update
  elisp-demos-add-demo
  :init
  (with-eval-after-load 'helpful
    (advice-add 'helpful-update :after #'elisp-demos-advice-helpful-update))
  :custom
  elisp-demos-user-files '("~/sync/orgzly/elisp-demos.org"))
```

10.15.21 Democratize

Democratize is similar to elisp-demos. I found out about it from [Democratize](#) [Populate your help buffers with usage examples](#) (Emacs package) – Listful Andrew, and I want to give it a try.

```
(use-package xht
  :vc (:url "https://git.sr.ht/~flandrew/xht"))
(use-package democratize
  :vc (:url "https://git.sr.ht/~flandrew/democratize")
  :config
  (democratize-enable-examples-in-helpful)
  (democratize-enable-examples-in-help))
```

10.15.22 JSON

```
(setq json-object-type 'alist
      json-array-type 'list)
```

10.15.23 Useful libraries

```
(use-package plz)
(use-package tzc)
```

10.15.24 Other useful functions

```
;; ##autoload
(defun sach-weekly-average (count start end)
  "Report weekly average for COUNT from START to END."
  (/ (* 7.0 count) (days-between end start)))
```

10.15.25 Try getting the differences between two lists of strings in terms of edit operations `emacs`

I want to be able to compare two lists of strings so that I can tell, for example, which sentences I've updated and therefore need to re-record. Diff can do that for me, but instead of having to split up the formatted output into different lines, it might be interesting to just reimplement it so that I can get the output format that I want.

I've moved this to the function `learn-lang-string-list-diff` in [learn-lang/learn-lang.el](#).

10.16 Snippets

```
;;;###autoload
(defun sacha-use-yasnippet-capf () (add-to-list 'completion-at-point-functions #'yasnippet-capf))
```

```
(use-package yasnippet
  :diminish yas-minor-mode
  :init (yas-global-mode)
  :config
    (push '(yasnippet backquote-change) warning-suppress-types)
    (yas-global-mode)
  (add-hook 'hippie-expand-try-functions-list 'yas-hippie-try-expand)
  (setq yas-key-syntaxes '("w_" "w_." "^ ")))
(setq yas-installed-snippets-dir "~/elisp/yasnippet-snippets")
(setq yas-expand-only-for-last-commands nil)
(yas-global-mode 1))

(use-package yasnippet-capf
  :after cape
  :config
  (add-to-list 'completion-at-point-functions #'yasnippet-capf)
  :hook
  (emacs-lisp-mode . sacha-use-yasnippet-capf)
  (lisp-interaction-mode . sacha-use-yasnippet-capf)
  (org-mode . sacha-use-yasnippet-capf)
  (js2-mode . sacha-use-yasnippet-capf)
  )
;;      (global-set-key (kbd "C-c y") (lambda () (interactive)
;;                                         (yas/load-directory "~/elisp/snippets")))
```

From <http://emacswiki.org/emacs/Yasnippet>

```
;;;###autoload
(defun shk-yas/helm-prompt (prompt choices &optional display-fn)
  "Use helm to select a snippet. Put this into `yas/prompt-functions.'"
  (interactive)
  (setq display-fn (or display-fn 'identity))
  (if (require 'helm-config)
      (let (tmpsource cand result rmap)
        (setq cand (mapcar (lambda (x) (funcall display-fn x)) choices))
        (setq rmap (mapcar (lambda (x) (cons (funcall display-fn x) x)) choices))
        (setq tmpsource
          (list
            (cons 'name prompt)
            (cons 'candidates cand)
            '(action . (("Expand" . (lambda (selection) selection))))
          ))
        (setq result (helm-other-buffer '(tmpsource) "*helm-select-yasnippet"))
        (if (null result)
            (signal 'quit "user quit!")
            (cdr (assoc result rmap))))
      nil))
```

From <https://github.com/pcmantz/elisp/blob/master/sacha-bindings.el>

```
(setq default-cursor-color "gray")
(setq yasnippet-can-fire-cursor-color "purple")
```

```
;; It will test whether it can expand, if yes, cursor color -> green.
;;;###autoload
(defun yasnippet-can-fire-p (&optional field)
  (interactive)
  (setq yas--condition-cache-timestamp (current-time))
  (let (templates-and-pos)
    (unless (and yas-expand-only-for-last-commands
                  (not (member last-command yas-expand-only-for-last-commands)))
      (setq templates-and-pos (if field
                                  (save-restriction
                                    (narrow-to-region (yas--field-start field)
                                                         (yas--field-end field))
                                    (yas--templates-for-key-at-point))
                                  (yas--templates-for-key-at-point))))
      (and templates-and-pos (first templates-and-pos))))))

;;;###autoload
(defun sacha-change-cursor-color-when-can-expand (&optional field)
  (interactive)
  (when (eq last-command 'self-insert-command)
    (set-cursor-color (if (sacha-can-expand)
                          yasnippet-can-fire-cursor-color
                          default-cursor-color))))

;;;###autoload
(defun sacha-can-expand ()
  "Return true if right after an expandable thing."
  (or (abbrev--before-point) (yasnippet-can-fire-p)))

;;;###autoload
(defun sacha-insert-space-or-expand ()
  "For binding to the SPC SPC keychord."
  (interactive)
  (condition-case nil (or (sacha-hippie-expand-maybe nil) (insert " "))))
```

```
;; As pointed out by Dmitri, this will make sure it will update color when needed.
(remove-hook 'post-command-hook 'sacha-change-cursor-color-when-can-expand)
```

This requires me to modify the behaviour of hippie-expand so that it doesn't ding so much.

```
;;;###autoload
(defun sacha-hippie-expand-maybe (arg)
  "Try to expand text before point, using multiple methods.
The expansion functions in `hippie-expand-try-functions-list' are
tried in order, until a possible expansion is found. Repeated
application of `hippie-expand' inserts successively possible
expansions.
With a positive numeric argument, jumps directly to the ARG next
function in this list. With a negative argument or just \\[universal-argument],
undoes the expansion."
  (interactive "P")
  (require 'hippie-exp)
  (if (or (not arg)
          (and (integerp arg) (> arg 0)))
```

```

(let ((first (or (= he-num -1)
                 (not (equal this-command last-command)))))
  (if first
      (progn
        (setq he-num -1)
        (setq he-tried-table nil)))
      (if arg
          (if (not first) (he-reset-string))
          (setq arg 0))
      (let ((i (max (+ he-num arg) 0)))
        (while (not (or (≥ i (length hippie-expand-try-functions-list))
                        (apply (nth i hippie-expand-try-functions-list)
                              (list (= he-num i))))))
          (setq i (1+ i)))
        (setq he-num i))
      (if (≥ he-num (length hippie-expand-try-functions-list))
          (progn (setq he-num -1) nil)
          (if (and hippie-expand-verbose
                  (not (window-minibuffer-p)))
              (message "Using %s"
                       (nth he-num hippie-expand-try-functions-list))))))
  (if (and (≥ he-num 0)
          (eq (marker-buffer he-string-beg) (current-buffer)))
      (progn
        (setq he-num -1)
        (he-reset-string)
        (if (and hippie-expand-verbose
                  (not (window-minibuffer-p)))
            (message "Undoing expansions"))))))

```

10.17 Show column number

I sometimes need to know where I am in a line.

```
(column-number-mode 1)
```

10.18 Don't show whitespace in diff, but show context

```
(setq vc-diff-switches '("-b" "-B" "-u"))
(setq vc-git-diff-switches nil)
```

10.19 Javascript

Handy shortcuts:

```
(add-to-list 'auto-mode-alist '("\\.c?js\\'" . js-mode))
```

```
(use-package coffee-mode
  :if sacha-laptop-p
  :mode "\\*.coffee\\'"
  :bind (:map coffee-mode-map ("C-c C-c" . compile)))
```

```
(use-package jasminejs-mode
  :if sacha-laptop-p
  :after js2-mode
  :hook ((js2-mode . jasminejs-mode)
        (jasminejs-mode-hook . jasminejs-add-snippets-to-yas-snippet-dirs)))
```

This makes script blocks easier to copy:

```
(defvar sacha-javascript-test-regexp (concat (regexp-quote "/* Testing */") "\\(.*\\n\\)*")
  "Regular expression matching testing-related code to remove.
  See `sacha-copy-javascript-region-or-buffer'."
  )

;; ##autoload
(defun sacha-copy-javascript-region-or-buffer (beg end)
  "Copy the active region or the buffer, wrapping it in script tags.
  Add a comment with the current filename and skip test-related
  code. See `sacha-javascript-test-regexp' to change the way
  test-related code is detected."
  (interactive "r")
  (unless (region-active-p)
    (setq beg (point-min) end (point-max)))
  (kill-new
   (concat
    "<script type=\"text/javascript\">\n"
    (if (buffer-file-name) (concat "// " (file-name-nondirectory (buffer-file-name)) "\n") "")
    (replace-regexp-in-string
     sacha-javascript-test-regexp
     ""
     (buffer-substring (point-min) (point-max)))
    nil)
   "\n</script>"))
```

This makes it easier to debug:

```
(defvar sacha-debug-counter 1)
;; ##autoload
(defun sacha-insert-or-flush-debug (&optional reset beg end)
  (interactive "pr")
  (cond
   ((= reset 4)
    (save-excursion
     (flush-lines "console.log('DEBUG: [0-9]+'" (point-min) (point-max))
     (setq sacha-debug-counter 1)))
   ((region-active-p)
    (save-excursion
     (goto-char end)
     (insert ");\n")
     (goto-char beg)
     (insert (format "console.log('DEBUG: %d', " sacha-debug-counter))
     (setq sacha-debug-counter (1+ sacha-debug-counter))
     (js2-indent-line)))
   (t
    ;; Wrap the region in the debug
    (insert (format "console.log('DEBUG: %d');\n" sacha-debug-counter))
    (setq sacha-debug-counter (1+ sacha-debug-counter))
    (backward-char 3)
    (js2-indent-line))))
```


And the rest of the js2 config:

```
(use-package js2-mode
  :if sacha-laptop-p
  :commands js2-mode
  :defer t
  :interpreter "node"
  :init (setq js-indent-level 2)
         :mode "\\.\\[mc]?js\\'")
  :bind (:map js2-mode-map
    ("C-x C-e" . js-send-last-sexp)
    ("C-M-x" . js-send-last-sexp-and-go)
    ("C-c d" . sacha-insert-or-flush-debug)
    ("C-c C-b" . js-send-buffer-and-go)
    ("C-c w" . sacha-copy-javascript-region-or-buffer))
  :config (js2-imenu-extras-setup))
```

```
(use-package coffee-mode
  :if sachalaptop-p
  :defer t
  :config (setq-default coffee-js-mode 'js2-mode coffee-tab-width 2))
```

10.19.1 Node

```
(with-eval-after-load 'compile
  (add-to-list 'compilation-error-regexp-alist 'node)
  (add-to-list 'compilation-error-regexp-alist
    '(node "^[[[:blank:]]*at \\(?:.* (\\\\|\\\\\\)\\\\(.+?\\\\):\\\\([0-9]+\\\\):\\\\([0-9]+\\\\))?$" ↵
      ↵ 1 2 3)))
```

10.19.2 Indium

```
(use-package indium
:hook ((js2-mode . indium-interaction-mode)))
```

10.19.3 React

```
(use-package rjsx-mode
  :defer t
  :if sacha-laptop-p)
```

10.20 Typescript

```
(use-package typescript-mode
  :mode "\\\\.ts\\'")
```

10.21 HTML

Convenience function for getting rid of annoying spans offby1 says there's (setq nxml-sexp-element-flag t)

Hello world

```
;;;###autoload
(defun sacha-clean-up-spans-in-region (beg end)
  (interactive "r")
  (save-excursion
    (let ((changed t))
      (while changed
        (setq changed nil)
        (goto-char beg)
        (while (re-search-forward "<span>\\([<]*\\)</span>" end t)
          (replace-match "\\1")
          (setq changed t)))
        (setq changed t)
        (while changed
          (setq changed nil)
          (goto-char beg)
          (while (re-search-forward "<span>*\\(<a[<]+>[<]*</a>\\)</span>" end t)
            (replace-match "\\1")
            (setq changed t))))))

;;;###autoload
(defun sacha-clean-up-spans-in-string (string)
  (with-temp-buffer
    (insert string)
    (sacha-clean-up-spans-in-region (point-min) (point-max))
    (buffer-string)))

(ert-deftest sacha-clean-up-spans-in-string ()
  (should (string= (sacha-clean-up-spans-in-string "<span><span>Hello world</span></span>")
    "Hello world"))
  (should (string= (sacha-clean-up-spans-in-string "<span><span><a ↵
    ↵ href=\"http://example.com\">Hello another world</a></span></span>"
    "<a href=\"http://example.com\">Hello another world</a>"))
  (should (string= (sacha-clean-up-spans-in-string "<span><h1>Leave alone</h1></span>" ↵
    ↵ "<span><h1>Leave alone</h1></span>"))
  (should (string= (sacha-clean-up-spans-in-string "<span><a ↵
    ↵ href=\"http://example.com\">Leave</a> alone</span>"
    "<span><a href=\"http://example.com\">Leave</a> alone</span>"))))

;; (ert "sacha-clean-up-spans-in-string")
```

10.22 Shell

Make files executable if the first file has a shebang (ex: `#!/bin/bash`)

```
(add-hook 'after-save-hook
  'executable-make-buffer-file-executable-if-script-p)
```

Don't let me mess with the prompts.

```
(setq comint-prompt-read-only t)
(setq comint-completion-autolist t)
(setq comint-input-ignoredups t)
```

10.22.1 Shellcheck

<https://amitp.blogspot.com/2023/10/emacs-and-shellcheck.html>

```
;;;###autoload
(defun sacha-consult-flymake-project ()
  (interactive)
  (consult-flymake t))

(use-package flymake
  :bind (("S-e" . flymake-show-project-diagnostics)))

(use-package sh-script
  :hook (sh-mode . flymake-mode))

(use-package flymake-shellcheck :defer t)
(use-package flymake
  :bind (("S-e" . sacha-consult-flymake-project))
  :custom
  (flymake-suppress-zero-counters t)
  :config
  (defface sacha-flymake-modeline-error-echo
    '((t :inherit 'flymake-error-echo :background "red"))
    "Mode line flymake errors")
  (defface sacha-flymake-modeline-warning-echo
    '((t :inherit 'flymake-warning-echo :background "orange"))
    "Mode line flymake warnings")
  (put 'flymake-error 'mode-line-face 'sacha-flymake-modeline-error-echo)
  (put 'flymake-warning 'mode-line-face 'sacha-flymake-modeline-warning-echo))
```

10.22.2 dwim-shell-command

```
;;;###autoload
(defun sacha-dwim-shell-command (prefix)
  "Execute DWIM shell command asynchronously using noweb templates."

  Which files

  `dwim-shell-command' attempts to guess which file(s) you may want
  the command to operate on.

  1. If visiting a `dired' buffer, draw the marked file(s).
  2. If visiting any other buffer with an associated file, use that.

  Templates

  Operate on drawn files using either the following:

  <<f>> (file path,used by default)
  <<fne>> (file path without extension)
  <<e>> (extension)
  <<td>> (generate a temporary directory)
  <<*>> (all files joined)
  <<cb>> (clipboard)
  <<n>>, <<ln>>, or <<An>> (for current iteration)

  For example:
```

With drawn files '("\path/to/image1.png\" \"path/to/image2.png\"")

\\"convert '<<f>>' '<<fne>>.jpg'\" expands to

```
\\"convert 'path/to/image1.png' 'path/to/image1.jpg'\"  
\\"convert 'path/to/image2.png' 'path/to/image2.jpg'\"
```

while \\"ls -lh <<*>>\" expands to

```
\\"ls -lh path/to/image1.png path/to/image2.png\"
```

Focus

``dwim-shell-command'` creates a process buffer to capture command output, but doesn't display or focus on it by default. Instead, it tries to guess what's more convenient to focus on.

While the process is busy, show a spinner in the minibuffer. No focus changes.

After process is finished:

1. If there were any files created in the ``default-directory'`, jump to a ``dired'` buffer and move point to the new file (via ``dired-jump'`).
2. If no new files were created, automatically switch focus to the process buffer and display its output.

Note: You can prevent this automatic focus by prepending your command with whitespace.

```
|  
V  
\\" convert '<<f>>' '<<fne>>.jpg'\"
```

3. If the shell command caused any errors, offer to focus the process buffer and display its output.

Quick exit

Process buffers are read-only and can be quickly closed by pressing ``q'`.

Prefix

With PREFIX, execute command that number of times."

```
(interactive "p")  
  (require 'dwim-shell-command)  
(let ((script (read-shell-command dwim-shell-command-prompt)))  
  (unless (string-match "<<" script) (setq script (concat script " <<f>>")))  
  (dwim-shell-command-on-marked-files  
   dwim-shell-command-buffer-name script  
   :repeat prefix  
   :shell-util dwim-shell-command-shell-util  
   :shell-args dwim-shell-command-shell-args  
   :silent-success (string-prefix-p " " script)  
   :error-autofocus (not dwim-shell-command-prompt-on-error))))
```

```
(use-package dwim-shell-command
  :if sachalaptop-p
  :bind (([remap shell-command] . sachadwim-shell-command)
         :map dired-mode-map
         ([remap dired-do-async-shell-command] . sachadwim-shell-command)
         ([remap dired-do-shell-command] . sachadwim-shell-command)
         ([remap dired-smart-shell-command] . sachadwim-shell-command))
  )
```

10.22.3 Exec path from shell

I need to copy some environment variables from my profile.

```
(use-package exec-path-from-shell
  :init
  (exec-path-from-shell-initialize)
  :custom
  (exec-path-from-shell-variables
   '("PATH" "MANPATH"
     "GOOGLE_API_KEY"
     "AZURE_SPEECH_KEY"
     "AZURE_SPEECH_REGION"
     "GEMINI_API_KEY"
     "GEMINI_PAID_API_KEY"
     "MISTRAL_API_KEY")
  ))
```

10.23 Magit – nice git interface

Thanks to sheijk for hints on tweaking magit to limit it to the current directory!

```
(defvar sachamagit-limit-to-directory nil "Limit magit status to a specific directory.")
;; ###autoload
(defun sachamagit-stage-all-and-commit (message)
  (interactive (list (progn (magit-diff-unstaged) (read-string "Commit Message: "))))
  (magit-stage-modified)
  (magit-commit-create (list "-m" message))
  (call-interactively #'magit-push-current-to-pushremote))
;; ###autoload
(defun sachamagit-status-in-directory (directory)
  "Displays magit status limited to DIRECTORY.
Uses the current `default-directory', or prompts for a directory
if called with a prefix argument. Sets `sachamagit-limit-to-directory'
so that it's still active even after you stage a change. Very experimental."
  (interactive (list (expand-file-name
                      (if current-prefix-arg
                          (read-directory-name "Directory: ")
                          default-directory))))
  (setq sachamagit-limit-to-directory directory)
  (magit-status directory))
```

```
(defvar sachamagit-limit-to-directory)
```

```

(use-package magit
  :config
  (setq magit-diff-options '("-b")) ; ignore whitespace
  (setq sachamagit-limit-to-directory nil)
  (defadvice magit-insert-untracked-files (around sachamagit-activate)
    (if sachamagit-limit-to-directory
      (magit-with-section (section untracked 'untracked "Untracked files:" t)
        (let ((files (cl-mapcan
                      (lambda (f)
                        (when (eq (aref f 0) ??) (list f)))
                      (magit-git-lines
                       "status" "--porcelain" "--" ←
                       sachamagit-limit-to-directory))))
          (if (not files)
              (setq section nil)
              (dolist (file files)
                (setq file (magit-decode-git-path (substring file 3)))
                (magit-with-section (section file file)
                  (insert "\t" file "\n"))
                (insert "\n")))))
        ad-do-it))
      (defadvice magit-insert-unstaged-changes (around sachamagit-activate)
        (if sachamagit-limit-to-directory
          (let ((magit-current-diff-range (cons 'index 'working))
                (magit-diff-options (copy-sequence magit-diff-options)))
            (magit-git-insert-section (untracked "Unstaged changes:")
                                      #'magit-wash-raw-diffs
                                      "diff-files"
                                      "--" sachamagit-limit-to-directory
                                      ))
          ad-do-it))
          (defadvice magit-insert-staged-changes (around sachamagit-activate)
            "Limit to `sachamagit-limit-to-directory' if specified."
            (if sachamagit-limit-to-directory
              (let ((no-commit (not (magit-git-success "log" "-1" "HEAD"))))
                (when (or no-commit (magit-anything-staged-p))
                  (let ((magit-current-diff-range (cons "HEAD" 'index))
                        (base (if no-commit
                                   (magit-git-string "mktree")
                                   "HEAD")))
                    (magit-diff-options (append '("--cached") magit-diff-options)))
                    (magit-git-insert-section (staged "Staged changes:")
                                              (apply-partially #'magit-wash-raw-diffs t)
                                              "diff-index" "--cached" base "--" ←
                                              sachamagit-limit-to-directory))))
                ad-do-it))
              :bind ((("C-x v C-d" . sachamagit-status-in-directory)
                     ("C-c g" . magit-file-dispatch)
                     ("C-x g" . magit-status)
                     ("C-x v p" . magit-push)
                     ("C-x v c" . sachamagit-stage-all-and-commit)))

```

The proper way to implement this is probably to patch or override the definition of `magit-git-insert-section` so that it takes a list of options to add at the end of the command, but that can wait for another time (or braver souls).

10.23.1 TODO Make this better by adding a post command options variable

10.23.2 Finding repos with uncommitted changes

Based on <http://yitang.uk/2024/01/14/atomic-habit-in-emacs-keep-git-repos-clean/>

```
;;;###autoload
(defun sacha-git-find-unclean-repo (root-dir)
  "Find repo with modified files."
  ;; (interactive)
  (setq out nil)
  (dolist (dir (directory-files-recursively root-dir "\\\\.git$" t))
    (message "checking repo %s" dir)
    (let* ((git-dir (file-name-parent-directory dir))
           (default-directory git-dir))
      (unless (string= "" (shell-command-to-string "git status --untracked=no --porcelain"))
        (push git-dir out))))
  out)

;;;###autoload
(defun sacha-list-uncommitted-projects ()
  (interactive)
  (let ((s (string-join
            (seq-keep
              (lambda (root)
                (when-let ((repo ↵
                          (sacha-git-find-unclean-repo root)))
                  (concat "- "
                          (org-link-make-string
                           (format "elisp:(magit-status \"%s\")"
                               (car repo))
                           (file-name-nondirectory (replace-regexp-in-string "/" "$" "" root))))))
              (seq-uniq
                (mapcar (lambda (row)
                          (or (projectile-project-root
                               (car row))
                              (car row))
                            (car row))))
                (cons ↵
                  '("~/sync/emacs") sacha-project-web-base-list))))
            "\n"))
    (when (called-interactively-p 'any)
      (switch-to-buffer (get-buffer-create "*Uncommitted*"))
      (erase-buffer)
      (insert s)
      (org-mode))
    s))
```

```
(sacha-list-uncommitted-projects)
```

10.23.3 Use difftastic

From [Difftastic diffing with Magit](#), modified `GIT_EXTERNAL_DIFF` to `difftastic` (installed with `snap install difftastic`, and with `TMPDIR` set to `~/tmp` because the snap can't access `/tmp`):

```
;;###autoload
(defun th/magit--with-difftastic (buffer command)
  "Run COMMAND with GIT_EXTERNAL_DIFF=difftastic then show result in BUFFER."
  (let ((process-environment
        (cons (concat "TMPDIR=~/.tmp GIT_EXTERNAL_DIFF=difftastic --width="
                      (number-to-string (frame-width)))
              process-environment)))
    ;; Clear the result buffer (we might regenerate a diff, e.g., for
    ;; the current changes in our working directory).
    (with-current-buffer buffer
      (setq buffer-read-only nil)
      (erase-buffer))
    ;; Now spawn a process calling the git COMMAND.
    (make-process
     :name (buffer-name buffer)
     :buffer buffer
     :command command
     ;; Don't query for running processes when emacs is quit.
     :noquery t
     ;; Show the result buffer once the process has finished.
     :sentinel (lambda (proc event)
                  (when (eq (process-status proc) 'exit)
                    (with-current-buffer (process-buffer proc)
                      (goto-char (point-min))
                      (ansi-color-apply-on-region (point-min) (point-max))
                      (setq buffer-read-only t)
                      (view-mode)
                      (end-of-line)
                      ;; difftastic diffs are usually 2-column side-by-side,
                      ;; so ensure our window is wide enough.
                      (let ((width (current-column)))
                        (while (zerop (forward-line 1))
                          (end-of-line)
                          (setq width (max (current-column) width)))
                        ;; Add column size of fringes
                        (setq width (+ width
                                       (fringe-columns 'left)
                                       (fringe-columns 'right)))
                        (goto-char (point-min))
                        (pop-to-buffer
                         (current-buffer)
                         `((; If the buffer is that wide that splitting the frame in
                           ;; two side-by-side windows would result in less than
                           ;; 80 columns left, ensure it's shown at the bottom.
                           ,(when (> 80 (- (frame-width) width))
                              #'display-buffer-at-bottom)
                           (window-width
                            . ,(min width (frame-width))))))))))
    ;;###autoload
  (defun th/magit-show-with-difftastic (rev)
    "Show the result of \"git show REV\" with GIT_EXTERNAL_DIFF=diff."
    (interactive
     (list (or
            ;; If REV is given, just use it.
```



```

    (when (boundp 'rev) rev)
    ;; If not invoked with prefix arg, try to guess the REV from
    ;; point's position.
    (and (not current-prefix-arg)
         (or (magit-thing-at-point 'git-revision t)
             (magit-branch-or-commit-at-point)))
    ;; Otherwise, query the user.
    (magit-read-branch-or-commit "Revision"))))
(if (not rev)
    (error "No revision specified")
    (th/magit--with-difftastic
     (get-buffer-create (concat "*git show difftastic " rev "*"))
     (list "git" "--no-pager" "show" "--ext-diff" rev))))
;; ###autoload
(defun th/magit-diff-with-difftastic (arg)
  "Show the result of \"git diff ARG\" with GIT_EXTERNAL_DIFF=difft."
  (interactive
   (list (or
          ;; If RANGE is given, just use it.
          (when (boundp 'range) range)
          ;; If prefix arg is given, query the user.
          (and current-prefix-arg
               (magit-diff-read-range-or-commit "Range"))
          ;; Otherwise, auto-guess based on position of point, e.g., based on
          ;; if we are in the Staged or Unstaged section.
          (pcase (magit-diff--dwim)
              ('unmerged (error "unmerged is not yet implemented"))
              ('unstaged nil)
              ('staged "--cached")
              (`(stash . ,value) (error "stash is not yet implemented"))
              (`(commit . ,value) (format "%s^..%s" value value))
              ((and range (pred stringp)) range)
              (_ (magit-diff-read-range-or-commit "Range/Commit"))))))
  (let ((name (concat "*git diff difftastic"
                     (if arg (concat " " arg) "")
                     "*")))
    (th/magit--with-difftastic
     (get-buffer-create name)
     `("git" "--no-pager" "diff" "--ext-diff" ,@ (when arg (list arg)))))
  (transient-define-prefix th/magit-aux-commands ()
    "My personal auxiliary magit commands."
    ["Auxiliary commands"
     ("d" "Difftastic Diff (dwim)" th/magit-diff-with-difftastic)
     ("s" "Difftastic Show" th/magit-show-with-difftastic)])

```

```

(with-eval-after-load 'magit
  (transient-append-suffix 'magit-dispatch "!"
    '("#" "My Magit Cmds" th/magit-aux-commands))
  (define-key magit-status-mode-map (kbd "#") #'th/magit-aux-commands))

```

10.24 Forge

```

(use-package forge
  :after magit)

```

10.25 Checking things out

Based on <http://xenodium.com/emacs-clone-git-repo-from-clipboard/> :

```
(defvar sach-git-clone-destination "~/vendor")
;;;###autoload
(defun sach-git-clone-clipboard-url ()
  "Clone git URL in clipboard asynchronously and open in dired when finished."
  (interactive)
  (cl-assert (string-match-p "^\\(http\\|https\\|ssh\\)://" (current-kill 0)) nil "No URL in ↵
↵ clipboard")
  (let* ((url (current-kill 0))
         (download-dir (expand-file-name sach-git-clone-destination))
         (project-dir (concat (file-name-as-directory download-dir)
                               (file-name-base url)))
         (default-directory download-dir)
         (command (format "git clone %s" url))
         (buffer (generate-new-buffer (format "%s*" command))))
    (proc)
    (when (file-exists-p project-dir)
      (if (y-or-n-p (format "%s exists. delete?" (file-name-base url)))
          (delete-directory project-dir t)
          (user-error "Bailed")))
    (switch-to-buffer buffer)
    (setq proc (start-process-shell-command (nth 0 (split-string command)) buffer command))
    (with-current-buffer buffer
      (setq default-directory download-dir)
      (shell-command-save-pos-or-erase)
      (require 'shell)
      (shell-mode)
      (view-mode +1))
    (set-process-sentinel proc (lambda (process state)
                                (let ((output (with-current-buffer (process-buffer process)
                                                         (buffer-string))))
                                  (kill-buffer (process-buffer process))
                                  (if (= (process-exit-status process) 0)
                                      (progn
                                       (message "finished: %s" command)
                                       (dired project-dir))
                                      (user-error (format "%s\n%s" command output))))))
    (set-process-filter proc #'comint-output-filter)))
```

10.26 git-messenger – shows commit message

```
(use-package git-messenger
  :bind (("C-x v m" . git-messenger:popup-message)))
```

10.27 Tag files

I don't often use a TAGS file, but when I do, I don't want to have to set my tags file per project. I search for it in the directory tree instead.

```
;;;###autoload
(defun sach-recursive-find-file (file &optional directory)
  "Find the first FILE in DIRECTORY or its parents."
```

```

(setq directory (or directory (file-name-directory (buffer-file-name)) (pwd)))
(if (file-exists-p (expand-file-name file directory))
    (expand-file-name file directory)
    (unless (string= directory "/")
        (sacha-recursive-find-file file (expand-file-name ".." directory))))))

;;;###autoload
(defun sacha-find-tags ()
  "Set the TAGS file."
  (set (make-variable-buffer-local 'tags-table-list) nil)
  (set (make-variable-buffer-local 'tags-file-name)
    (sacha-recursive-find-file "TAGS")))

(with-eval-after-load 'drupal-mode
  (add-hook 'drupal-mode-hook 'sacha-find-tags))

```

10.28 Projects and projectile

```

;;;###autoload
(defun sacha-projectile-open-notes ()
  (interactive)
  (find-file-other-window (expand-file-name "notes.org" (projectile-project-root))))

```

```

(use-package projectile
  :diminish projectile-mode
  :config
    (define-key projectile-mode-map (kbd "C-c p") 'projectile-command-map)
  (projectile-mode +1)
  (setq projectile-completion-system 'default)
  (setq projectile-enable-caching t)
  (setq projectile-indexing-method 'alien)
  (add-to-list 'projectile-globally-ignored-files "node_modules")
  (add-to-list 'projectile-globally-ignored-files ".cache")
  (add-to-list 'projectile-globally-ignored-files "_cache")
  (add-to-list 'projectile-globally-ignored-files "~")
  (add-to-list 'projectile-globally-ignored-files "#"))
;; Call with C-c p m m
(use-package makefile-executor
  :if sacha-laptop-p
  :defer t
  :config
  (add-hook 'makefile-mode-hook 'makefile-executor-mode))

```

10.28.1 Capturing notes to per-project files

Based on [Juggling multiple projects and leveraging org-projectile](#) | Shreyas Ragavan

```

(use-package org-project-capture :defer t)
(use-package org-projectile
  :after org-project-capture
  :config
  (setq org-projectile-per-project-filepath "notes.org")
  (org-projectile-per-project)
  (org-project-capture-per-project))

```

```
(push (org-projectile-project-todo-entry) org-capture-templates)
;; I have some remote files I don't want included.
;; (setq org-agenda-files (append org-agenda-files (org-projectile-todo-files)))
:bind
(("C-c p n" . org-projectile-capture-for-current-project)))
```

10.29 Exploring MELPA recipes

10.30 Ruby

```
(use-package rinari :if sachalaptop-p)
(use-package bundler :if sachalaptop-p)
(use-package robe
  :if sachalaptop-p
  :hook
  ((ruby-mode-hook . robe-mode)
   (robe-mode-hook . ac-robe-setup)
   (ruby-mode-hook . auto-complete-mode)))
(use-package haml-mode
  :if sachalaptop-p
  :mode "\\\\.haml\\'")
```

```
;;;###autoload
(defun sachalrspec-verify-single ()
  "Runs the specified example at the point of the current buffer."
  (interactive)
  (rspec-run-single-file
   (concat
    (rspec-spec-file-for (buffer-file-name))
    ".")
   (save-restriction
    (widen)
    (number-to-string (line-number-at-pos))))
  (rspec-core-options)))
```

```
(use-package rspec-mode
  :if sachalaptop-p
  :config
  (progn
   (setq rspec-command-options "--fail-fast --format documentation")
   (keymap-set rspec-mode-map "C-c , , " 'rspec-rerun)
   (fset 'rspec-verify-single 'sachalrspec-verify-single)))
```

SASS

```
(use-package sass-mode
  :if sachalaptop-p
  :hook (sass-mode-hook . (lambda () (setq indent-tabs-mode nil))))
(setq-default indent-tabs-mode nil)
```

```
(use-package inf-ruby
  :defer t
  :config
  (setq inf-ruby-prompt-format
```

```

(concat
  (mapconcat
    #'identity
    '("\(\(^%s> *\)\)"
      "\(\(^(rdb:1) *\)\)" ; Simple
      "\(\(^(rdbg[^\)]*) *\)\)" ; Debugger
      "\(\(^(rdbg[^\)]*) *\)\)" ; Ruby Debug Gem
      "\(\(^(byebug) *\)\)" ; byebug
      "\(\(^(irb([^\)]+)\)" ; IRB default
      "\(\([0-9]+\) \)\)?[Pp]ry ?([^\)]+)\)" ; Pry
      "\(\(^([^\s]+\)\)" ; new rails console ↵
    ↵ with project name and environment
    ↵
    ↵ "(\(jruby-\|JRUBY-\)\)?[1-9]\.[0-9]\(\([0-9]+\)\)*\(\(-?p?[0-9]+\)\)?" ; RVM
      "\^rbx-head\)\)" ; RVM ↵
    ↵ continued
      "\|")
    ;; Statement and nesting counters, common to the last four.
    " ?[0-9:]* ?%s *\)\)"
    inf-ruby-first-prompt-pattern
    (format inf-ruby-prompt-format ">" ">" ">")
    inf-ruby-prompt-pattern
    (format inf-ruby-prompt-format "[?>]" "*>" "[\]>*" '/''))))

```

10.31 Skewer

This lets you send HTML, CSS, and Javascript fragments to Google Chrome. You may need to start Chrome with `chrome --allow-running-insecure-content`, if you're using the user script with HTTPS sites.

```

(use-package skewer-mode
  :if sachalaptop-p
  :hook
  ((js2-mode-hook . skewer-mode)
   (css-mode-hook . skewer-css-mode)
   (html-mode-hook . skewer-html-mode)))

```

10.32 Autocomplete

```

(with-eval-after-load 'company
  (define-key company-mode-map (kbd "<tab>") 'company-indent-or-complete-common))
(use-package company
  :if sachalaptop-p
  ;;:init (add-hook 'prog-mode-hook 'company-mode)
  )
(use-package company-posframe :if sachalaptop-p :init (company-posframe-mode 1) :diminish)

```

10.33 Tern - for Javascript

Let's skip this for now.

```

(use-package tern
  :if sachalaptop-p
  :bind (:map tern-mode-keymap ("C-c C-c" . compile))
  :hook (js2-mode-hook . tern-mode))

```

```
:config
(when (eq system-type 'windows-nt) (setq tern-command '("cmd" "/c" "tern"))))
```

10.34 Docker

```
(use-package dockerfile-mode
:mode ("Dockerfile\\'" . dockerfile-mode))
```

10.35 Automation

```
;;###autoload
(defun sacha-insert-xdotool-click-as-shell-command ()
  (interactive)
  (insert
   (shell-command-to-string "xdotool getmouselocation | sed -E 's/x:([0-9]+) y:([0-9]+) ↵
↵ ↵ ./xdotool mousemove \\1 \\2 click 1/'")))
```

10.35.1 Multiple cursors mode

drill

I often define keyboard macros to process multiple lines in a region. Maybe `multiple-cursors` will be an even better way. Looks promising! See [Emacs Rocks episode 13 \(multiple-cursors\)](#) for a great demo.

```
(use-package multiple-cursors
:bind
(("C-c m t" . mc/mark-all-like-this)
 ("C-c m m" . mc/mark-all-like-this-dwim)
 ("C-c m l" . mc/edit-lines)
 ("C-c m e" . mc/edit-ends-of-lines)
 ("C-c m a" . mc/edit-beginnings-of-lines)
 ("C-c m n" . mc/mark-next-like-this)
 ("C-c m p" . mc/mark-previous-like-this)
 ("C-c m s" . mc/mark-sgml-tag-pair)
 ("C-c m d" . mc/mark-all-like-this-in-defun)))
(use-package phi-search :defer t)
(use-package phi-search-mc :config (phi-search-mc/setup-keys) :defer t)
(use-package mc-extras :config (define-key mc/keymap (kbd "C-. =") 'mc/compare-chars) :defer t)
```

Thanks to [Irreal](#) and [Planet Emacsen](#) for the link!

10.36 Eshell

https://www.reddit.com/r/emacs/comments/b6n3t8/what_would_it_take_to_get_terminal_colors_in/

```
(use-package xterm-color
:commands (xterm-color-filter))
(use-package eshell
:after xterm-color
:config
  (setq eshell-scroll-to-bottom-on-input t)
  (define-key eshell-mode-map (kbd "<tab>") #'company-complete))
```

```
(define-key eshell-hist-mode-map (kbd "M-r") #'consult-history)
(add-hook 'eshell-mode-hook
  (lambda ()
    (setenv "TERM" "xterm-256color")))
(add-hook 'eshell-before-prompt-hook (setq xterm-color-preserve-properties t))
(add-to-list 'eshell-preoutput-filter-functions 'xterm-color-filter)
(setq eshell-output-filter-functions
  (remove 'eshell-handle-ansi-color eshell-output-filter-functions)))
```

10.36.1 Eshell completion

Source

```
(use-package capf-autosuggest
  :hook
  (eshell-mode . capf-autosuggest-mode))
```

10.36.2 Correctly complete commands in subdirectories

From <https://www.n16f.net/blog/eshell-key-bindings-and-completion/>

```
;; ##autoload
(defun eshell--complete-commands-list ()
  "Generate list of applicable, visible commands."
  (let ((filename (pcomplete-arg)) glob-name)
    (if (file-name-directory filename)
        (if eshell-force-execution
            (pcomplete-dirs-or-entries nil ←
              (pcomplete-executables))
          (if (and (> (length filename) 0)
                (eq (aref filename 0) ←
                  eshell-explicit-command-char))
              (setq filename (substring filename 1)
                    pcomplete-stub filename
                    glob-name t))
              (let* ((paths (eshell-get-path))
                    (cwd (file-name-as-directory
                          ←
                            (expand-file-name default-directory))))
                (path "") (comps-in-path ())
                (file "") (filepath "") (completions ←
                  ;; Go thru each path in the search path, finding ←
                  (while paths
                    (setq path (file-name-as-directory
                              ←
                                (expand-file-name (or (car paths) ".")))
                          comps-in-path
                          (and ←
                            (file-accessible-directory-p path)
                            (file-name-all-completions filename path)))
                    ;; Go thru each completion found, to see ←
                    (whether it should
```

```

;; be used.
(while comps-in-path
  (setq file (car comps-in-path)
        filepath ←
    (concat path file))
    (if (and (not (member file ←
    completions)) ;
    (or ←
    (string-equal path cwd)
    (not (file-directory-p filepath)))
    (if ←
    eshell-force-execution
    (file-readable-p filepath)
    (file-executable-p filepath)))
    (setq completions ←
    (cons file completions)))
    (setq comps-in-path (cdr comps-in-path))
    (setq paths (cdr paths)))
;; Add aliases which are currently visible, and Lisp ←
functions.
(pcomplete-uniquify-list
  (if glob-name
    completions
    (setq completions
    (append (if (fboundp ←
    'eshell-alias-completions)
    (eshell-alias-completions filename))
    (eshell-winnow-list
    (mapcar
    (lambda (name)
    (substring name 7))
    (all-completions (concat "eshell/" filename)
    obarray #'functionp))
    nil '(eshell-find-alias-function))
    completions))
    (append (and (or eshell-show-lisp-completions
    (and eshell-show-lisp-alternatives
    (null completions)))
    (all-completions filename obarray #'functionp))
    completions))))))

```


10.37 SQLite

From <https://christiantietze.de/posts/2024/01/emacs-sqlite-mode-open-sqlite-files-automatically/>

```
;;###autoload
(defun ct/sqlite-view-file-magically ()
  "Runs `sqlite-mode-open-file' on the file name visited by the
current buffer, killing it."
  (require 'sqlite-mode)
  (let ((file-name buffer-file-name))
    (kill-current-buffer)
    (sqlite-mode-open-file file-name)))
```

```
(use-package sqlite-mode
  :commands sqlite-mode-open-file
  :config
  (add-to-list 'magic-mode-alist '("SQLite format 3\x00" . ct/sqlite-view-file-magically)))
```

10.38 Crontab

10.39 TODO Write a function to help with crontab entries

```
;;###autoload
(defun sachabbb-insert-translated-crontab-entries (time-zone start-time end-time)
  "Insert crontab entries.
Convert START-TIME and END-TIME from TIME-ZONE.
(Assume END-TIME is after START-TIME.)
Create a pair of crontab entries for the start (converted to local time)
and end."
  (interactive
   (progn
    (require 'tzc)
    (let* ((time-zone (completing-read "Timezone: " tzc-time-zones))
           (start-time
            (org-read-date t t nil "Start date and time: "))
           (end-time
            (org-read-date t t nil "End time: " start-time)))
      (list time-zone start-time end-time))))
  (when (stringp start-time)
    (setq start-time (org-read-date t t start-time)))
  (when (stringp end-time)
    (setq end-time (org-read-date t t end-time nil start-time)))
  (let ((tz-offset (format-time-string "%z" start-time time-zone))
        text)
    (setq start-time (date-to-time (concat (format-time-string "%Y-%m-%dT%H:%M:%S.000" ←
→ start-time) tz-offset)))
    (setq end-time (date-to-time (concat (format-time-string "%Y-%m-%dT%H:%M:%S.000" ←
→ end-time) tz-offset)))
    (setq text (concat
                 (format-time-string "%-M %-H %-d %-m * \n" start-time)
                 (format-time-string "%-M %-H %-d %-m * " end-time)))
    (when (called-interactively-p 'any)
      (insert text))
    text))
```

```
(ert-deftest sachabbb-insert-translated-crontab-entries ()
  (should (string= (sachabbb-insert-translated-crontab-entries
    "Europe/Berlin"
    "2025-11-12T07:00:00"
    "2025-11-12T10:30:00")
    "0 1 12 11 *
30 4 12 11 * "))))
```

10.40 Documentation

Hmm, disable this for now

```
(use-package dash-docs
  :init
  (unless (file-directory-p (dash-docs-docsets-path))
    (make-directory (dash-docs-docsets-path)))
  :dash (python-mode "NumPy" "OpenCV Python" "Pandas")
  (web-mode "HTML" "CSS" "Handlebars" "jQuery")
  (haml-mode "Haml")
  (conf-mode "Nginx")
  (markdown-mode "Markdown")
  (js2-mode "NodeJS" "Express" "MomentJS" "jQuery")
  (emacs-lisp-mode "Emacs Lisp")
  (sh-mode "Bash"))
(use-package consult-dash
  :bind ((("M-s d" . consult-dash))
  :config
  (consult-customize consult-dash :initial (thing-at-point 'symbol)))
```

dash-docs-search

11 Internet Relay Chat

IRC is a great way to hang out with other Emacs geeks.

```
(use-package erc
  :defer t
  :commands erc-select
  :if sachalaptop-p
  :config
  (setq erc-track-remove-disconnected-buffers t)
  (setq erc-hide-list '("PART" "QUIT" "JOIN"))
  (setq erc-autojoin-channels-alist '(("freenode.net"
    "#org-mode"
    "#emacs"
    "#emacs-beginners"
    "#emacs-ops")
    ("irc.chat.twitch.tv"
    "#sachachua")

    ("irc.tilde.chat"
    "#emacs.ch"))

  erc-server "irc.freenode.net"
  erc-nick "sachac"
```

```
erc-track '("NICK" "333" "353" "JOIN" "PART" "AWAY")))
```

```
;; ##autoload
(defun erc-cmd-OPME ()
  "Request chanserv to op me."
  (erc-message "PRIVMSG"
    (format "chanserv op %s %s"
      (erc-default-target)
      (erc-current-nick)) nil))

;; ##autoload
(defun erc-cmd-DEOPME ()
  "Deop myself from current channel."
  (erc-cmd-DEOP (format "%s" (erc-current-nick))))

;; ##autoload
(defun erc-cmd-BAN (nick)
  (let* ((chan (erc-default-target))
    (who (erc-get-server-user nick))
    (host (erc-server-user-host who))
    (user (erc-server-user-login who)))
    (erc-server-send (format "MODE %s +b *!%s@%s" chan user host))))

;; ##autoload
(defun erc-cmd-KICKBAN (nick &rest reason)
  (setq reason (mapconcat #'identity reason " "))
  (and (string= reason "")
    (setq reason nil))
  (erc-cmd-BAN nick)
  (erc-server-send (format "KICK %s %s %s"
    (erc-default-target)
    nick
    (or reason
      "Kicked (kickban)"))))

;; ##autoload
(defun sacha-erc-clean-up ()
  "Clean up dead ERC buffers."
  (interactive)
  (mapc #'kill-buffer (erc-buffer-list (lambda () (null (erc-server-process-alive)))))
  (erc-update-mode-line))
```

11.1 Search logs

```
;; ##autoload
(defun sacha-search-irc-logs ()
  (interactive)
  (let ((default-directory "~/backups/server/home/.znc/users/sachac/moddata/log"))
    (consult-ripgrep default-directory)))

;; ##autoload
(defun sacha-irc-log-dired ()
  (interactive)
  (dired "~/backups/server/home/.znc/users/sachac/moddata/log"))
```

12 Mastodon

mastodon

```

(use-package tp
  :vc (:url "https://codeberg.org/martianh/tp.el")
  )
(use-package mastodon
  :if sacha-laptop-p
  :load-path "~/vendor/mastodon.el/lisp"
  :config
  (require 'mastodon-tl)
  :bind
  (:map mastodon-mode-map
    ("g" . mastodon-tl-update)
    ;; see org-capture-templates addition
    ("o" . (lambda () (interactive) (org-capture nil "m"))))
    :map mastodon-toot-mode-map)
  :commands (mastodon-http--api
              mastodon-http--post
              mastodon-mode
              mastodon-http--get-search-json
              mastodon-tl-get-local-timeline)

  :custom
  (mastodon-tl--display-media-p nil)
  (mastodon-instance-url "https://social.sachachua.com")
  (mastodon-active-user "sacha")
  (mastodon-group-notifications nil))

```

```

;;;###autoload
(defun sacha-mastodon-clear-auth ()
  "Fix alist-get: Wrong type argument: listp, (error . \"The access token is invalid\") ↵
  ↵ error. Then you can use `mastodon-auth--access-token'."
  (interactive)
  (setq mastodon-client--active-user-details-plist nil)
  (delete-file (concat user-emacs-directory "mastodon.plstore"))
  (setq mastodon-auth--token-alist nil))

;;;###autoload
(defun sacha-mastodon-toot-public-string (message)
  (interactive "sMessage: ")
  (mastodon-toot--compose-buffer
   nil nil nil
   message)
  (condition-case nil (mastodon-toot-send)
    (error nil)))

;;;###autoload
(defun sacha-mastodon-show-sacha-followers ()
  (interactive)
  (mastodon-profile--make-profile-buffer-for
   (mastodon-profile--lookup-account-in-status (mastodon-auth--get-account-name) nil)
   "followers"
   #'mastodon-profile--add-author-bylines))

;;;###autoload
(defun sacha-yank-mastodon-link ()
  (interactive)
  (let* ((url (current-kill 0))
         (url-parsed (url-generic-parse-url url))
         (user (file-name-base (url-filename url-parsed))))

```

```
(cond
  ((derived-mode-p 'odd-muse-mode) (insert "[" url " " user
                                           "@" (url-host url-parsed) "]")))
  ((derived-mode-p 'org-mode) (insert "[" url "]" user
                                       "@" (url-host url-parsed) "]")))
  (t (insert url))))

(declare-function 'mastodon-notifications-get-mentions "mastodon-notifications")
```

I use [Tusky](#) on my Android phone in order to share post content with [Orgzly](#) (synchronized via [Syncthing](#)) so that I can add TODOs or notes to my Org Mode files. The following code makes it easy to open links to things that look like Mastodon URLs by using `mastodon.el`.

```
(autoload 'mastodon-url-lookup "mastodon")
(add-to-list 'browse-url-handlers '("https?://[^\s/]+/[^\s/]+.*" . sacha-mastodon-browse-url))
```

```
;; ##autoload
(defun sacha-mastodon-browse-url (url &rest _)
  "Open URL."
  (if (string-match "medium\\.com" url)
      (funcall browse-url-browser-function url)
      (mastodon-url-lookup url)))
```

(thanks, [@ParetoOptimalDev!](#))

12.1 Adding Mastodon toots as comments in my 11ty static blog

Following up on [@mrg@mastodon.sdf.org](#)'s recommendation of [C. Moe | Comments via Mastodon](#), I wanted to figure out how to copy that toot as a comment on Moving 18 years of comments out of Disqus and into my 11ty static site (p. ??).

I can start out with manually identifying which blog post to attach the comment to and finding or creating the appropriate comment info. To make this easier, I've added `inputPath` to <https://sachachua.com/blog/all/index.json>.

```
;; ##autoload
(defun sacha-11ty-comment-file (url)
  (interactive (list (sacha-complete-blog-post-url)))
  (let* ((permalink (replace-regexp-in-string "^https?://[^\s/]+" "" url))
         (post (seq-find (lambda (o)
                           (string= (alist-get 'permalink o)
                                         permalink)))
                      (json-read-file
                        (expand-file-name "_site/blog/all/index.json"
                                          sacha-11ty-base-dir))))
         (filename
          (if post
              (expand-file-name
               (concat
                (file-name-sans-extension
                 (alist-get 'inputPath
```



```
(find-file (sacha-11ty-add-blog-comment (sacha-mastodon-toot-comment-json))))
```

12.2 mastodon.el: Copy toot content as Org Mode

mastodon.org:emacs

Sometimes I want to copy a toot and include it in my Org Mode notes, like when I post a thought and then want to flesh it out into a blog post. This code defines `sacha-mastodon-org-copy-toot-content`, which converts the toot text to Org Mode format using Pandoc and puts it in the kill ring so I can yank it somewhere else.

```
;;;###autoload
(defun sacha-mastodon-toot-at-url (&optional url)
  "Return JSON toot object at URL.
If URL is nil, return JSON toot object at point."
  (if url
      (let* ((search (format "%s/api/v2/search" mastodon-instance-url))
              (params `(("q" . ,url)
                        ("resolve" . "t"))) ; webfinger
              (response (mastodon-http--get-json search
                                                    (params :silent))))
            (car (alist-get 'statuses response)))
      (mastodon-toot--base-toot-or-item-json)))

;;;###autoload
(defun sacha-mastodon-org-copy-toot-content (&optional url)
  "Copy the current toot's content as Org Mode.
Use pandoc to convert.

When called with \\[universal-argument], prompt for a URL."
  (interactive (list
                (when current-prefix-arg
                  (read-string "URL: "))))

  (let ((toot (sacha-mastodon-toot-at-url url)))
    (with-temp-buffer
      (insert (alist-get 'content toot))
      (call-process-region nil nil "pandoc" t t nil "-f" "html" "-t" "org")
      (kill-new
       (concat
        (org-link-make-string
         (alist-get 'url toot)
         (concat "@" (alist-get 'acct (alist-get 'account toot))))
        ":\n\n#+begin_quote\n"
        (string-trim (buffer-string)) "\n\n#+end_quote\n"))
      (message "Copied."))))
```

12.3 mastodon.el: Mention people based on regexp

I haven't quite figured out how to handle autocompletion that works well for me.

```
(defvar sacha-org-contacts-file "~/sync/orgzly/people.org")
;;;###autoload
(defun sacha-mastodon-insert-handle-from-contacts ()
  (interactive)
  (let ((collection
```

```

                                (with-temp-buffer
                                  (insert-file-contents sach-org-contacts-file)
                                  (org-mode)
                                  (goto-char (point-min))
                                  (org-map-entries
                                    (lambda ()
                                      (let ((handle (org-entry-get (point) ↵
                                                                (name ↵
                                                                (cons (format "%s (%s)" name ↵
                                                                handle))))
                                      "MASTODON={.}")))))
                                  (insert (assoc-default (completing-read "Name: " collection)
                                                        ↵
                                                        collection #'string= ""))))
↵

```

```

;;;###autoload
(defun sach-org-contacts-all-alist ()
  "Return a list of all contacts in `org-contacts-files'.
Each element has the form (NAME . (FILE . POSITION))."
  (seq-mapcat
    (lambda (file)
      (unless (buffer-live-p (get-buffer (file-name-nondirectory file)))
        (find-file-noselect file))
      (with-current-buffer (find-file-noselect file)
        (org-map-entries
          (lambda ()
            (let* ((name (substring-no-properties (org-get-heading t t t)))
                  (file (buffer-file-name))
                  (position (point))
                  (entry-properties (org-entry-properties position 'standard)))
              `(("NAME" . ,name)
                ("FILE" . ,file)
                ("POSITION" . ,position)
                ,@entry-properties))))))
      (org-contacts-files)))

;;;###autoload
(defun sach-org-contacts-to-mention (text)
  (seq-filter
    (lambda (o) (and (assoc-default "MENTION_REGEX" o #'string=)
                     (string-match (assoc-default "MENTION_REGEX" o #'string=) text)))
    (sach-org-contacts-all-alist))

```

```

;;;###autoload
(defun sach-mastodon-interested-handles (text)
  (seq-uniq
    (append
      (seq-keep
        (lambda (o)
          (assoc-default "MASTODON" o #'string=))
        (sach-org-contacts-to-mention text))
      (with-temp-buffer
        (insert text)
        (goto-char (point-min))

```



```

      (cl-loop while (re-search-forward "@[\\\.0-9A-Z_a-z-]+@[\\\.0-9A-Z_a-z-]+" nil t)
        collect (match-string 0))))))

;;;###autoload
(defun sacha-mastodon-insert-interested-handles (text)
  (interactive (list (mastodon-toot--remove-docs)))
  (when-let* ((handles (sacha-mastodon-interested-handles text)))
    (save-excursion
      (unless (looking-at " ") (insert " "))
      (insert (string-join handles " "))))))

```

Let's generalize this.

```

;;;###autoload
(defun sacha-org-contacts-best-contact (rec)
  (seq-find
    (lambda (type)
      (when (assoc-default type rec #'string=)
        type))
    '("MASTODON" "EMAIL" "X")))

;;;###autoload
(defun sacha-org-contacts-collect-context (contact text)
  "Return the list of links or keywords matching MENTION_REGEXP for CONTACT in TEXT."
  (with-temp-buffer
    (insert text)
    (goto-char (point-min))
    (org-mode)
    (let ((regexp (assoc-default "MENTION_REGEXP" contact #'string=))
          results)
      (while (re-search-forward regexp nil t)
        (let ((elem (org-element-context)))
          (if (eq (org-element-type elem) 'link)
              (push (org-element-property :raw-link elem)
                    results)
              (push (sentence-at-point) results))))
      (nreverse results))))

;;;###autoload
(defun sacha-org-contacts-suggest-mentions (text url)
  (interactive (list (if (region-active-p) (buffer-substring (region-beginning) (region-end))
    (sacha-11ty-post-text))
    (sacha-11ty-post-url)))

  (when-let* ((people (seq-group-by #'sacha-org-contacts-best-contact ↵
    ↵ (sacha-org-contacts-to-mention text))))
    (with-current-buffer (get-buffer-create "*mentions*")
      (erase-buffer)
      (org-mode)
      (dolist (type '("MASTODON" "EMAIL" "X"))
        (when (assoc-default type people #'string=)
          (insert "*" " " type "\n\n"
            (mapconcat (lambda (o)
              (assoc-default type o #'string=)
              (assoc-default type people #'string=) ", ")
              "\n"
            (string-join
              (seq-mapcat

```

```

        (lambda (o)
          (sacha-org-contacts-collect-context o text))
        (assoc-default type people #'string=))
      "\n")
      (if url (concat " " url) "")
      "\n\n"))))
(goto-char (point-min))
;; collect the links
(pop-to-buffer (current-buffer))))))

```

12.4 mastodon.el: Collect handles in clipboard (Emacs kill ring) mastodon:emacs

I sometimes want to thank a bunch of people for contributing to a Mastodon conversation. The following code lets me collect handles in a single kill ring entry by calling it with my point over a handle or a toot, or with an active region.

```

(defvar sacha-mastodon-handle "@sacha@social.sachachua.com")
;; ###autoload
(defun sacha-mastodon-copy-handle (&optional start-new beg end)
  "Append Mastodon handles to the kill ring.

Use the handle at point or the author of the toot. If called with a
region, collect all handles in the region.

Append to the current kill if it starts with @. If not, start a new
kill. Call with \\[universal-argument] to always start a new list.

Omit my own handle, as specified in `sacha-mastodon-handle'."
  (interactive (list current-prefix-arg

                  (when <=
                    (region-active-p) (region-beginning))

                    (when <=
                    (region-active-p) (region-end))))
  (let ((handle
        (if (and beg end)
            ;; collect handles in region
            (save-excursion
              (goto-char beg)
              (let (list)
                ;; Collect all handles from the specified region
                (while (< (point) end)
                  (let ((mastodon-handle (get-text-property (point) 'mastodon-handle))
                        (button (get-text-property (point) 'button)))
                    (cond
                     (mastodon-handle
                      (when <=
                        (and (string-match "@" mastodon-handle)
                            (or (null sacha-mastodon-handle)
                                (not (string=
                                     sacha-mastodon-handle mastodon-handle))))
                      (cl-pushnew
                       (concat (if (string-match "^@" mastodon-handle) ""

```



```

↳ (get-text-property (point) 'mastodon-handle)

(alist-get
 'acct
 (alist-get ↳

↳ 'account (mastodon-toot--base-toot-or-item-json))))))
  (if (or start-new (null kill-ring) (not (string-match "^@" (car kill-ring))))
      (kill-new handle)
      (dolist (h (split-string handle " "))
        (unless (member h (split-string " " (car kill-ring)))
          (setf (car kill-ring) (concat (car kill-ring) " " h))))))
    (message "%s" (car kill-ring))))

```

Another perk of tooting from Emacs using [mastodon.el](#). =)

12.5 mastodon.el: Copy toot URL after posting; also, copying just this post with 11ty mastodon:emacs:11ty

I often want to copy the toot URL after posting a new toot about a blog post so that I can update the blog post with it. Since I post from Emacs using [mastodon.el](#), I can probably figure out how to get the URL after tooting. A quick-and-dirty way is to retrieve the latest status.

```

(defvar sacha-mastodon-toot-posted-hook nil "Called with the item.")

;;;###autoload
(defun sacha-mastodon-copy-toot-url (toot)
  (interactive (list (sacha-mastodon-latest-toot)))
  (kill-new (alist-get 'url toot)))
(add-hook 'sacha-mastodon-toot-posted-hook #'sacha-mastodon-copy-toot-url)

;;;###autoload
(defun sacha-mastodon-latest-toot ()
  (interactive)
  (require 'mastodon-http)
  (let* ((json-array-type 'list)
         (json-object-type 'alist))
    (car
     (mastodon-http--get-json
      (mastodon-http--api
       (format "accounts/%s/statuses?count=1&limit=1&exclude_reblogs=t"
               (mastodon-auth--get-account-id)))
      nil :silent))))

```

```

(with-eval-after-load 'mastodon-toot
  (when (functionp 'mastodon-toot-send)
    (advice-add
     #'mastodon-toot-send
     :after
     (lambda (&rest _)
       (run-hook-with-args 'sacha-mastodon-toot-posted-hook ↳
        ↳ (sacha-mastodon-latest-toot))))))
  (when (functionp 'mastodon-toot--send)
    (advice-add
     #'mastodon-toot--send
     :after
     (lambda (&rest _)

```

```
(run-hook-with-args 'sacha-mastodon-toot-posted-hook ←
  ↪ (sacha-mastodon-latest-toot))))))
```

I considered overriding the keybinding in `mastodon-toot-mode-map`, but I figured using advice would mean I can copy things even after automated toots.

A more elegant way to do this might be to modify `mastodon-toot-send` to `run-hook-with-args` a variable with the `response` as an argument, but this will do for now.

I used a hook in my advice so that I can change the behaviour from other functions. For example, I have some code to compose a toot with [a link to the current post](#) (p. 575). After I send a toot, I want to check if the toot contains the current entry's permalink. If it has and I don't have a Mastodon toot field yet, maybe I can automatically set that property, assuming I end up back in the Org Mode file I started it from.

```
;;###autoload
(defun sacha-mastodon-org-maybe-set-toot-url (toot)
  (cond
    ((derived-mode-p 'org-mode)
     (let ((permalink (org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK")))
       (when (and permalink
                    (string-match (regexp-quote ↪
                                  ↪ permalink) (alist-get 'content toot))
                    (not ↪
                        ↪ (org-entry-get-with-inheritance "MASTODON")))
         (save-excursion
           (goto-char (org-find-property "EXPORT_ELEVENTY_PERMALINK"
                                         ↪
                                         ↪ permalink))
           (org-entry-put
            (point)
            "EXPORT_MASTODON"
            (alist-get 'url toot))
            (message "Toot URL set: %s, republish if needed" ↪
                    ↪ toot))))))
    (t
     (when (buffer-file-name)
       (let (filename data)
         (cond
          ((string-match "\\..11tydata\\.json" (buffer-file-name))
           (setq data (json-parse-string (buffer-string) ↪
                                          ↪ :object-type 'alist :array-type 'list)))
          ((and (buffer-file-name)
                 (file-exists-p (concat ↪
                                         ↪ (file-name-sans-extension (buffer-file-name)) ".11tydata.json")))
           (let ((json-object-type 'alist)
                 (json-array-type 'list))
             (setq
              filename (concat (file-name-sans-extension ↪
                                                            ↪ (buffer-file-name)) ".11tydata.json")
              data (json-read-file filename))))))
       (when (and
              data
              (string-match (regexp-quote ↪
                              ↪ (alist-get 'permalink data)) (alist-get 'content toot))
              (not (alist-get 'mastodon data)))
         (push (cons 'mastodon (alist-get 'url toot)) data)
         (if filename
```

```

                                (with-temp-file filename
                                  (insert (json-encode data)))
                                (erase-buffer)
                                (insert (json-encode data)))))))))
(add-hook 'sacha-mastodon-toot-posted-hook #'sacha-mastodon-org-maybe-set-toot-url)

```

If I combine that with a development copy of my blog that ignores most of my posts so it compiles faster and a function that copies just the current post's files over, I can quickly make a post available at its permalink (which means the link in the toot will work) before I recompile the rest of the blog, which takes a number of minutes.

```

;;###autoload
(defun sacha-org-11ty-copy-just-this-post (&optional url)
  (interactive)
  (cond
   (url
    (let* ((relative-path (replace-regexp-in-string
                          (concat "^" (regexp-quote sacha-blog-base-url) "\\|/"/)
                          ""
                          url))
           (local (expand-file-name relative-path (expand-file-name "_local" ←
→ sacha-11ty-base-dir)))
           (remote (concat "web:/var/www/static-blog/" relative-path))
           (remote-tramp (concat "/ssh:" remote)))
      (if (file-directory-p local)
          (progn
            (call-process "chmod" nil nil nil "ugo+rX" "-R" local)
            (unless (file-directory-p (file-name-directory remote-tramp))
              (make-directory (file-name-directory remote-tramp) t))
            (call-process "rsync" nil (get-buffer-create "*rsync*") nil "--chmod=ugo+rX" ←
→ "-avzpe" "ssh"
                        local
                        remote))
          (when (or (called-interactively-p 'any) ←
→ sacha-org-11ty-export-and-copy-browse)
            (if (string-match "^https://" url)
                (browse-url url)
                (browse-url (concat ←
→ (replace-regexp-in-string "/$" "" sacha-blog-base-url) url))))))
      (error "Could not find %s" local)))
    ((derived-mode-p 'org-mode)
     (let* ((subtreep (not (org-before-first-heading-p)))
            (params (org-combine-plists
                     ←
→ (org-export--get-export-attributes '11ty subtreep nil)
                     ←
→ (org-export--get-buffer-attributes)
                     ←
→ (org-export-get-environment '11ty subtreep)))
            (file (plist-get params :file-name))
            (permalink (plist-get params :permalink))
            (local (expand-file-name file (expand-file-name ←
→ "_local" (plist-get params :base-dir)))
            (remote (concat "web:/var/www/static-blog/" file))
            (remote-tramp (concat "/ssh:" remote)))
      (if (and permalink file (file-directory-p local))
          (progn
            (call-process "chmod" nil nil nil "ugo+rX" ←

```

```

↳ "-R" local)
                                (unless (file-directory-p (file-name-directory ←
↳ remote-tramp))
                                (make-directory (file-name-directory ←
↳ remote-tramp) t))
                                (call-process "rsync" nil (get-buffer-create ←
↳ "*rsync*" nil "--chmod=ugo=rX" "-avzpe" "ssh"
                                ←
↳ local
                                ←
↳ remote)
                                (when (or (called-interactively-p 'any) ←
↳ sach-org-11ty-export-and-copy-browse)
                                (browse-url (concat ←
↳ (replace-regexp-in-string "/"$ "" sach-blog-base-url)
                                ←
↳ (error "Could not find %s" local))))
                                ((or (derived-mode-p 'html-mode)
↳ (derived-mode-p 'web-mode))
↳ (let* ((json-object-type 'alist)
↳ (permalink
↳ (alist-get 'permalink (json-read-file (concat ←
↳ (file-name-sans-extension (buffer-file-name)) ".11tydata.json"))))
↳ (local (expand-file-name (concat "." permalink) ←
↳ (expand-file-name "_local" sach-11ty-base-dir)))
↳ (remote (concat "web:/var/www/static-blog" permalink)))
↳ (call-process "rsync" nil (get-buffer-create "*rsync*") nil ←
↳ "--chmod=ugo=rX" "-avzpe" "ssh"
                                local
                                remote)
↳ (when (or (called-interactively-p 'any) ←
↳ sach-org-11ty-export-and-copy-browse)
↳ (browse-url (concat (replace-regexp-in-string "/"$ "" ←
↳ sach-blog-base-url)
↳ permalink)))))))))

```

The proper blog updates (index page, RSS/ATOM feeds, category pages, prev/next links, etc.) can happen when the publishing is finished.

So my draft workflow is:

1. Write the post.
2. Export it to the local `NODE_ENV=dev npx eleventy --serve --quiet` with `ox-11ty`.
3. Check that it looks okay locally.
4. Use `sach-org-11ty-copy-just-this-post` and confirm that it looks fine.
5. Compose a toot with `sach-mastodon-11ty-toot-post` (p. 575) and check if sending it updates the Mastodon toot.
6. Re-export the post.
7. Run my blog publishing process. `NODE_ENV=production npx eleventy --quiet` and then `rsync`.

Let's see if this works...

12.6 Storing Mastodon links in Org mode

This snippet makes it easier to store links to posts with **org-store-link** and to use them as automatic annotations in **org-capture**. (2022-12-11: Now it links to media attachments, too!)

[illegible]

```
(use-package org
  :config
  (org-link-set-parameters
   "mastodon"
   :store 'sacha-mastodon-store-link)
  (with-eval-after-load 'org-capture
    (add-to-list 'org-capture-templates

      ↪ ,sacha-org-inbox-file)

      ↪ %?\n\n#+begin_quote\n%:text\n#+end_quote\n\n%a"

      :prepend t))))
```

12.7 Collecting Emacs News from Mastodon

emacs:mastodon

[2025-03-06 Thu]: Use `sacha-org-link-url-from-string`.

One of the things I like about browsing Mastodon in Emacs using [mastodon.el](#) is that I can modify my workflow to make things easier. For example, I often come across links that I'd like to save for

Emacs News. I want to boost the post and save it to an Org file, and I can do that with a single keystroke. It uses the `sacha-mastodon-store-link` function defined elsewhere in `my config`.

```
;;###autoload
(defun sacha-mastodon-save-toot-for-emacs-news ()
  (interactive)
  ;; store a link and capture the note
  (org-capture nil "[ ]")
  ;; boost if not already boosted
  (unless (get-text-property
           (car
            (mastodon-tl--find-property-range 'byline ←
              ↪ (point)))
           'boosted-p)
    (mastodon-toot--toggle-boost-or-favourite 'boost)))
```

```
(with-eval-after-load 'org-capture
  (add-to-list
    'org-capture-templates
    '("n" "Emacs News" entry (file+headline "~/sync/orgzly/news.org" "Collect Emacs News")
      "* %a :news:

#+begin_quote
%:text
#+end_quote

"
      :prepend t :immediate-finish t)))

(use-package mastodon
  :bind (:map mastodon-mode-map ("w" . sachas-mastodon-save-toot-for-emacs-news)))
```

This puts a bunch of notes in my `~/sync/orgzly/news.org` file. Then I can use `sacha-emacs-news-summarize-mastodon` to summarize a bunch of items I've captured from Mastodon, taking the title from the first link and including a link to the toot using the author's handle. This is what it looks like:

images/output-2024-09-16-13:14:38.gif

Here's the code that makes that happen:

```
;; ##autoload  
(defun sacha-match-groups (&optional object)  
  "Return the matching groups, good for debugging regexps."  
  (seq-map-indexed (lambda (entry i)  
  
                                (list i entry) ↵  
                                ↵  
↵                                (and (car entry) ↵  
↵                                ↵  
↵                                (if object ↵  
↵                                ↵  
↵                                (substring object (car entry) ↵  
↵                                (cadr entry)) ↵  
↵                                ↵  
↵                                (buffer-substring (car entry) (cadr ↵  
↵                                entry))))))  
  
                                (seq-partition  
                                (match-data t)  
                                2)))
```

```

;;;###autoload
(defun sacha-org-link-url-from-string (s)
  "Return the link URL from S."
  (if (string-match org-link-any-re s)
      (or
       (match-string 7 s)
       (match-string 2 s))))

```

```

;;;###autoload
(defun sacha-mastodon-get-note-info ()
  "Return (:handle ... :url ... :links ... :text) for the current subtree."
  (let ((url (let ((title (org-entry-get (point) "ITEM")))
                (if (string-match org-link-any-re title)
                    (or
                     (match-string 7 title)
                     (match-string 2 title))))))
        (beg end handle)
        (save-excursion
          (org-back-to-heading)
          (org-end-of-meta-data)
          (setq beg (point))
          (setq end (org-end-of-subtree)))
        (unless url
          (goto-char beg)
          (when (re-search-forward org-any-link-re end t)
            (setq url (org-element-property :raw-link (org-element-context))))
          (cond
           ((string-match "\\[[\\[https://bsky\\.app/.+?\\]\\]\\(\\(\\.+\\)\\)\\]" url)
            (setq handle (match-string 1 url)))
           ((string-match "https://\\(\\.+?\\)\\(\\(\\.+?\\)\\)/" url)
            (setq handle (concat
                          (match-string 2 url) "@" (match-string 1 url))))
           ((string-match "https://\\(\\.+?\\)\\(\\(\\.+?\\)\\)/p/[0-9]+\\.\\[0-9\\] +" url)
            (setq handle (concat
                          (match-string 2 url) "@" (match-string 1 url))))
           (t
            (list
             :handle handle
             :url (if (string-match org-link-bracket-re url) (match-string 1 url) url)
             :links (reverse (mapcar (lambda (o) (org-element-property :raw-link o))
                                     (sacha-org-get-links-in-region beg end)))
             :text (string-trim (buffer-substring-no-properties beg end))))))
    (ert-deftest sacha-mastodon-get-note-info ()
      (should
       (equal
        (with-temp-buffer
          (insert "** SOMEDAY https://mastodon.online/@jcastp/111762105597746747"
                  :news:

```

```

:PROPERTIES:
:CREATED: [2024-01-22 Mon 05:51]
:END:

jcastp@mastodon.online - I've shared my emacs config: https://codeberg.org/jcastp/emacs.d

After years of reading other's configs, copying really useful snippets, and tinkering a little ↵
↳ bit myself, I wanted to give something back, although I'm still an amateur (and it ↵
↳ shows, but I want to improve!)

If you can find there something you can use, then I'm happy to be useful to the community.

#emacs
")
      (org-mode)
      (sacha-mastodon-get-note-info))
'(:handle "@jcastp@mastodon.online"
                                     :url
                                     ↵
↳ "https://mastodon.online/@jcastp/111762105597746747"
                                     :links
                                     ("https://codeberg.org/jcastp/emacs.d")
                                     :text
                                     "jcastp@mastodon.online - I've shared my ↵
↳ emacs config: https://codeberg.org/jcastp/emacs.d\n\nAfter years of reading other's ↵
↳ configs, copying really useful snippets, and tinkering a little bit myself, I wanted ↵
↳ to give something back, although I'm still an amateur (and it shows, but I want to ↵
↳ improve!)\n\nIf you can find there something you can use, then I'm happy to be useful ↵
↳ to the community.\n\n#emacs")))))

```

It turns headings into something like this:

- [Insert a screenshot image in latex-mode with the help of org-download \(@mmonga@mathstodon.xyz\)](#)

which I can then copy into my Emacs News Org Mode file and categorize with some keyboard shortcuts.

This works particularly well with [my combined Mastodon timelines](#) (p. 569), because then I can look through all the #emacs posts from mastodon.social, emacs.ch, and social.sachachua.com in one go.

12.8 Copy Mastodon link for Emacs News

```

;;;###autoload
(defun sacha-mastodon-copy-link-dwim (prefix)
  (interactive "P")
  (if prefix
      (mastodon-toot--copy-toot-url)
      (sacha-mastodon-copy-toot-as-author-link)))

;;;###autoload
(defun sacha-emacs-news-copy-mastodon-link ()
  (interactive)
  (let ((url (org-entry-get (point) "ITEM")))
    (when (string-match "https://\\(.+?\\)/\\(@.+?\\)/" url)
      (kill-new (org-link-make-string url (concat (match-string 2 url) "@") ↵
↳ (match-string 1 url))))))

;;;###autoload

```

```

(defun sacha-emacs-news-copy-mastodon-item (&optional name-only)
  (interactive (list current-prefix-arg))
  (let (s)
    (with-current-buffer
      (if (string-match "emacs-news/index.org" (buffer-file-name))
          (save-window-excursion
            (other-window 1)
            (current-buffer))
          (current-buffer))
      (let ((url (or (thing-at-point 'url)
                    (progn
                      ↵
                      (save-restriction
                        ↵
                        (org-back-to-heading)
                        ↵
                        (org-narrow-to-subtree)
                        ↵
                        (org-end-of-meta-data)
                        ↵
                        (if (re-search-forward org-link-any-re nil t)
                            ↵
                            (thing-at-point 'url)
                            ↵
                            (setq name-only t)
                            ↵
                            (org-entry-get (point) "ITEM")
                            ↵
                            )))))
          (toot (org-entry-get (point) "ITEM"))
          attrib)
        (when (string-match org-link-bracket-re toot)
          (setq toot (match-string 1 toot)))
        (when (string-match "https://\\(.+?\\)/\\(@.+?\\)/" toot)
          (setq attrib (org-link-make-string toot
                                              ↵
                                              ↵
                                              (concat
                                                ↵
                                                ↵
                                                (match-string 2 toot) "@" (match-string 1 toot))))
          (setq s
            (if name-only
                (format " (%s)" attrib)
                (format "- %s (%s)\n"
                        ↵
                        ↵
                        (org-link-make-string
                          ↵
                          url
                          ↵
                          (sacha-page-title url)
                          ↵
                          attrib))))))
      (when (called-interactively-p 'any)
        (if (string-match "emacs-news/index.org" (buffer-file-name))
            (insert s)
            (kill-new s)))
      s))

```

- [2025-09-15 Mon]: Removed `edited_at` attribute.
- [2025-03-24 Mon]: Updated `sacha-mastodon-follow-user` to use `alist-get`.
- [2024-11-04 Mon]: Make tag a parameter.
- [2024-10-07 Mon]: Added screenshot.
- [2024-09-16 Mon]: Read JSON arrays as lists to be compatible with the latest mastodon.el.



Screenshot of combined timeline in mastodon.el

I like checking out the #emacs hashtag when I put together Emacs News. In the past, I usually browsed the hashtag timeline on emacs.ch, which also picked up updates from other people that emacs.ch was following. Now that I've moved to @sacha@social.sachachua.com and emacs.ch is winding down, I wanted to see if there was a way for me to see a combined view using mastodon.social's API feed (paging by max_id as needed). I haven't enabled public timeline feeds on my server, so I also need to reuse the OAuth mechanics from mastodon.el.

First, let's start by making a unified timeline. By digging around in mastodon-tl.el, I found that I could easily create a timeline view by passing it a vector of toot JSONs.

```
;; ##autoload
(defun sacha-mastodon-fetch-posts-after (base-url after-date)
  "Page backwards through BASE-URL using max_id for all the posts after AFTER-DATE."
  (require 'plz)
  (require 'mastodon-http)
  (let ((results []))
    (url base-url)
    (use-mastodon-el (not (string-match "^http" base-url))))
```

```

        (json-array-type 'list)
        page filtered)
      (while url
        (setq page (if use-mastodon-el
          ↪
          (mastodon-http--get-json (mastodon-http--api url) nil :silent)
          (seq-map (lambda (o)
            ↪
            (cons (cons 'external t) o))
            ↪
            (plz 'get url :as #'json-read)))
          filtered (seq-filter (lambda (o) (string<
            ↪
            after-date (assoc-default 'created_at o)))
            ↪
            page))
          (if filtered
            (progn
              (setq results (seq-concatenate 'vector
            ↪
            filtered results)
              url (concat base-url
            ↪
            (if (string-match "\\?" base-url) "&" "?")
            ↪
            "max_id=" (assoc-default 'id (elt (last page) 0))))
              (message "%s %s" (assoc-default 'created_at
            ↪
            (elt (last page) 0)) url))
              (setq url nil)))
            results))

;;###autoload
(defun sacha-mastodon-combined-tag-timeline (later-than tag &optional servers)
  "Display items after LATER-THAN about TAG from SERVERS and the current mastodon.el
  ↪ account."
  (interactive (list
    (org-read-date nil nil nil nil
    ↪
    nil "-Mon")
    "#emacs"
    '("mastodon.social"
    ↪
    "fosstodon.org")))
  (setq servers (or servers '("mastodon.social" "fosstodon.org")))
  (require 'mastodon)
  (require 'mastodon-tl)
  (require 'mastodon-toot)
  (if (stringp later-than)
    (setq later-than (org-read-date nil nil later-than)))
  (setq tag (replace-regexp-in-string "#" "" tag))
  (let* ((limit 40)
    (sources (cons (format "timelines/tag/%s?limit=%d" tag limit)
    ↪
    (mapcar (lambda (s)
    ↪
    (format
    ↪
    "https://%s/api/v1/timelines/tag/%s?limit=%d" s tag limit))
    ↪
    servers)))
    (combined
      (seq-map
        ;; remove edited_at
        (lambda (o) (assoc-delete-all 'edited_at o))
        (sort

```



```

                                (end (next-single-property-change (or pos (point)) prop)))
                                (put-text-property (or start (point-min))
                                                    (or ←
                                                    prop ←
↪ end (point-max))
↪ value)))

```

```

;;;###autoload
(defun sacha-mastodon-update-external-item-id (&rest _)
  (when (mastodon-tl--field 'external (mastodon-tl--property 'item-json))
    ;; ask the server to resolve it
    (let* ((response (mastodon-http--get-json (format "%s/api/v2/search" ←
↪ mastodon-instance-url)
↪
↪ `(("q" . ,(mastodon-toot--toot-url))
↪
↪ ("resolve" . "t"))))
          (id (alist-get 'id (seq-first (assoc-default ←
↪ 'statuses response)))))
      (inhibit-read-only t)
      (json (get-text-property (point) 'item-json))
      (when (and id json)
        (sacha-text-property-update-at-point (point) 'base-item-id id)
        (sacha-text-property-update-at-point (point) 'item-json
↪ (progn
↪
↪ (setf (alist-get 'id json) id)
↪
↪ (setf (alist-get 'external json) nil)
↪
↪ json))))))

```

So now all I need to do is make sure that this is called before the relevant mastodon.el functions if I'm looking at an external toot.

```

(with-eval-after-load 'mastodon-tl
  (advice-add #'mastodon-toot--action :before #'sacha-mastodon-update-external-item-id)
  (advice-add #'mastodon-toot--reply :before #'sacha-mastodon-update-external-item-id)
  (advice-add #'mastodon-tl--thread :before #'sacha-mastodon-update-external-item-id))

```

The only thing is that I need to press RET after loading a thread with T (`mastodon-tl--thread`) for some reason, but that's okay. Now I can boost and save posts with my usual Emacs News shortcut, and I can reply easily too.

I'm curious: how many toots would I be missing if I looked at only one instance's hashtag? Let's look at the #emacs hashtag toots on 2024-09-12:

```

;;;###autoload
(defun sacha-three-way-comparison (seq1 seq2 seq3 &optional test-fn)
  `(("1" ,@ (seq-difference seq1 (seq-union seq2 seq3 test-fn) test-fn))
    ("2" ,@ (seq-difference seq2 (seq-union seq1 seq3 test-fn) test-fn))

```



```

("3" ,@ (seq-difference seq3 (seq-union seq1 seq2 test-fn) test-fn))
("1&2" ,@ (seq-difference (seq-intersection seq1 seq2 test-fn) seq3 test-fn))
("1&3" ,@ (seq-difference (seq-intersection seq1 seq3 test-fn) seq2 test-fn))
("2&3" ,@ (seq-difference (seq-intersection seq2 seq3 test-fn) seq1 test-fn))
("1&2&3" ,@ (seq-intersection (seq-intersection seq2 seq3 test-fn) seq1
  ↪ test-fn))))

;;###autoload
(defun sacha-three-way-comparison-report (label1 seq1 label2 seq2 label3 seq3 &optional test-fn)
  (let ((list (sacha-three-way-comparison seq1 seq2 seq3)))
    `((, (format "%s only" label1) ,@ (assoc-default "1" list #'string=))
      (, (format "%s only" label2) ,@ (assoc-default "2" list #'string=))
      (, (format "%s only" label3) ,@ (assoc-default "3" list #'string=))
      (, (format "%s & %s" label1 label2) ,@ (assoc-default "1&2" list
  ↪ #'string=))
      (, (format "%s & %s" label1 label3) ,@ (assoc-default "1&3" list
  ↪ #'string=))
      (, (format "%s & %s" label2 label3) ,@ (assoc-default "2&3" list
  ↪ #'string=))
      ("all" ,@ (assoc-default "1&2&3" list #'string=)))))

```

```
(assert (equal (sacha-three-way-comparison '("A" "A&B" "A&C" "A&B&C" "A1")
      ↪                                     ↪ ("B" "A&B" ↪
      ↪ "A&B&C" "B&C"))
      ↪                                     ↪
      ↪                                     ↪ ("C" "A&C" ↪
      ↪ "A&B&C" "B&C")))
      ↪                                     ↪
      ↪                                     ↪ (('("1" "A" "A1")
      ↪                                     ↪ ("2" "B")
      ↪                                     ↪ ("3" "C")
      ↪                                     ↪ ("1&2" "A&B")
      ↪                                     ↪ ("1&3" "A&C")
      ↪                                     ↪ ("2&3" "B&C")
      ↪                                     ↪ ("1&2&3" "A&B&C"))))

(let* ((later-than "2024-09-12")
      ↪
      ↪ (earlier-than "2024-09-13")
      ↪ (results
      ↪       ↪ (mapcar (lambda (o)
      ↪                               ↪ (cons (car o)
      ↪                                       ↪
      ↪ (seq-map (lambda (o) (assoc-default 'uri o))
      ↪
      ↪ (seq-filter (lambda (toot)
      ↪
      ↪ (string< (assoc-default 'created_at toot)
      ↪
      ↪
      ↪ earlier-than)))
      ↪
      ↪ (sacha-mastodon-fetch-posts-after
      ↪
      ↪ (format "%stimelines/tag/emacs?count=40" (cdr o)))
```

```

↵                                     ↵
↵         later-than))))))
                                     `((mastodon-social . ↵
↵ "https://mastodon.social/api/v1/")
                                     (emacs-ch . ↵
↵ "https://emacs.ch/api/v1/")
                                     (sacha-instance . "))))
                                     (intersections
                                     (let-alist results
                                     (sacha-three-way-comparison-report
                                     "mastodon.social"
                                     .mastodon-social
                                     "emacs.ch"
                                     .emacs-ch
                                     "my instance"
                                     .sacha-instance
                                     #'string=))))
↵ (mapcar
↵   (lambda (row)
↵     (list (elt row 0) (length (cdr row))
↵           (string-join
↵             (seq-map-indexed (lambda (o i)
↵                               ↵
↵                               (org-link-make-string o (number-to-string (1+ i))))
↵             (cdr row))
↵           " ")))
↵   intersections))

```

Here's an Euler diagram visualizing it.

I love that I can tinker with `mastodon.el` to get it to combine the timelines. (I'm crossing the streams!) Yay Emacs!

12.10 Following people

I want to be able to follow people if I specify their ID.

```

;;;###autoload
(defun sacha-mastodon-follow-user (user-handle)
  "Follow HANDLE."
  (interactive "MHandle: ")
  (require 'mastodon-profile)
  (when (string-match "https?://\\(.+?\\)\\/@(.+\\)" user-handle)
    (setq user-handle (concat (match-string user-handle) "@" (match-string 1 ↵
↵ user-handle))))
  (let* ((account (mastodon-profile--search-account-by-handle
                    user-handle))
         (user-id (alist-get 'id account))
         (name (if (not (string-empty-p (alist-get 'display_name ↵
↵ account)))
                   (alist-get 'display_name account)
                   (alist-get 'username account)))
         (url (mastodon-http--api (format "accounts/%s/%s" user-id ↵
↵ "follow"))))
    (if account
        (mastodon-tl--do-user-action-function url name user-handle ↵
        "follow")

```

#emacs posts on 2024-09-12

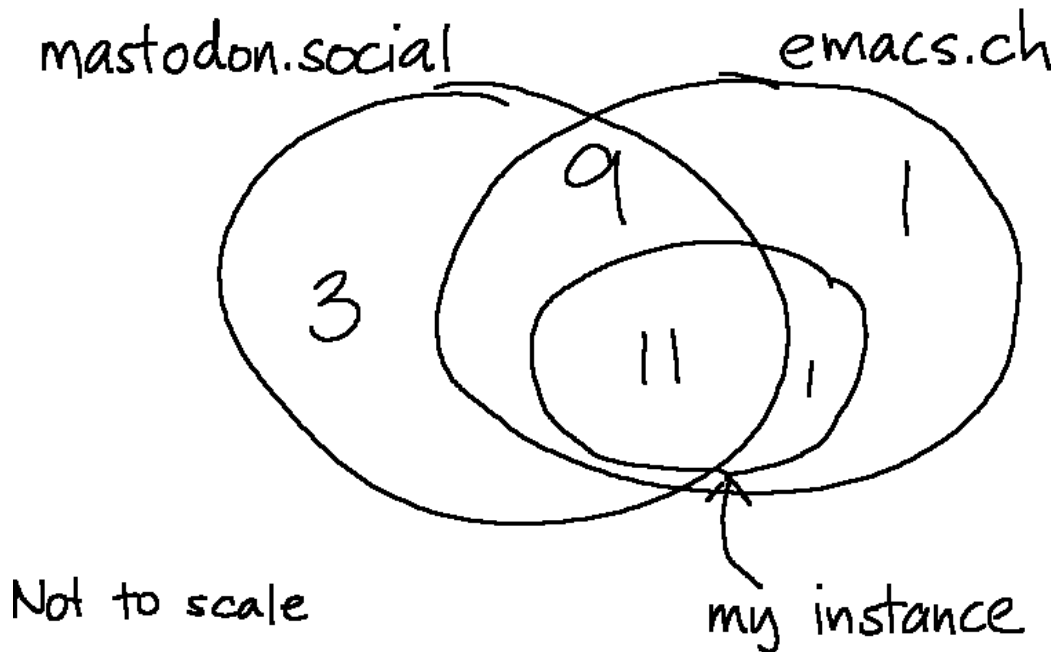


Figure 19: #emacs posts on 2024-09-12 - an Euler diagram showing the table above

```
(message "Cannot find a user with handle %S" user-handle)))))
```

12.11 Tooting a link to the current post

```
;;;###autoload
(defun sacha-11ty-post-plist ()
  (cond
    ((derived-mode-p 'org-mode)
     (let* ((subtreep (not (org-before-first-heading-p)))
            (combined (org-combine-plists
                       ↵
                       ↵
                       ↵
                       (org-export--get-export-attributes '11ty subtreep nil)
                       (org-export--get-buffer-attributes)
                       (org-export--get-environment '11ty subtreep))))
      (when (listp (plist-get combined :title))
        (plist-put combined :title (car (plist-get combined :title))))
        (plist-put combined :tags (org-get-tags))
        combined))
     ((or (derived-mode-p 'js-mode)
          (derived-mode-p 'jsonian-mode)) ; probably looking at ↵
      ↵ .11tydata.json
      (json-parse-string (buffer-string)
                          ↵
                          :object-type 'plist
```

```

↪ :array-type 'list))
  ((and (derived-mode-p 'html-mode)
        (file-exists-p (concat (file-name-base (buffer-file-name)) ↪
↪ ".11tydata.json"))))
    (let ((json-object-type 'plist)
          (json-array-type 'list))
      (json-read-file
       (concat (file-name-base (buffer-file-name)) ".11tydata.json"))))
    (t (error "Could not find info."))))

;;;###autoload
(defun sachal11ty-permalink ()
  "Get permalink for current post."
  (plist-get (sachal11ty-post-plist) :permalink))

;;;###autoload
(defun sachal11ty-tags ()
  "Get tags for current post."
  (plist-get (sachal11ty-post-plist) :tags))

;;;###autoload
(defun sachal11ty-post-url ()
  (when (org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK")
    (concat (replace-regexp-in-string "/"$ " " sachal-blog-base-url)
            (org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK"))))

;;;###autoload
(defun sachal11ty-post-text ()
  (save-excursion
    (goto-char
     (cond
      ((org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK")
       (org-find-property "EXPORT_ELEVENTY_PERMALINK"
↪
↪ (org-entry-get-with-inheritance "EXPORT_ELEVENTY_PERMALINK"))))
      (t (org-back-to-heading-or-point-min))))
    (if (org-before-first-heading-p)
        (buffer-string)
        (org-end-of-meta-data)
        (buffer-substring (point) (org-end-of-subtree))))))

```

```

;;;###autoload
(defun sachal-mastodon-11ty-toot-post ()
  "Compose a toot sharing this blog post on Mastodon."
  (interactive)
  (require 'mastodon)
  (require 'mastodon-toot)
  (let* ((info (sachal11ty-post-plist))
         (url (concat "https://sachachua.com" (plist-get info ↪
↪ :permalink)))
         (blog-text (sachal11ty-post-text))
         (title (plist-get info :title)))
    (mastodon-toot--compose-buffer
     nil nil nil
     (concat "[" (plist-get info :title) "]"
              "(" url ") "
              (mapconcat (lambda (tag) (concat "#" tag))

```

```

↪      (seq-remove (lambda (tag) (string-match "^_" tag))
↪
↪      (plist-get info :tags))
↪
↪      " "))
↪      (unless (string-match "[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9] Emacs News" ↪
↪ title)
↪      (sacha-mastodon-insert-interested-handles (concat title "\n" ↪
↪ blog-text))))))

```

12.12 Compose a Mastodon toot with the current Org subtree

I want to make it easier to microblog the current Org subtree.

```

;;;###autoload
(defun sacha-mastodon-toot-subtree ()
  (interactive)
  (let ((body (org-export-as 'md t nil t))
        (link (org-entry-get (point) "EXPORT_ELEVENTY_PERMALINK")))
    (mastodon-toot)
    (save-excursion
      (insert body)
      (when link (insert "\n\nBlog post: <" sacha-blog-base-url link ">\n")))))

```

The automatic link to my configuration is handled by a function that I add to `org-export-filter-body-functions`: [config-footer](#) (p. 295)

12.13 Posting the latest screenshot with mastodon.el

I want to make it easier to microblog the latest screenshot, or a recent screenshot if I need to pick a different one. It might also be a good time to add some text to the filename to make it easier to find later on. I can use that text as alt-text for the image, too.

```

;;;###autoload
(defun sacha-mastodon-toot-screenshot (&optional filename description)
  "Compose a buffer and attach the latest screenshot.
Prompt for a description and add that to the filename as well.
When called with a prefix argument, prompt for the file.
Use consult to provide a preview."
  (interactive
   (let ((filename
          (if current-prefix-arg
              (consult--read
               (sacha-combined-screenshots)
               :sort nil
               :require-match t
               :category 'file
               :state (lambda (candidate state)
                        ↪
↪      (when candidate
↪
↪      (with-current-buffer (find-file-noselect
↪
↪
↪      candidate)

```

```

↪                                     (display-buffer (current-buffer))))))
                                     (sacha-latest-screenshot))))

      (list
        filename
        (when (string-match ↪
↪ "^[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]_[0-9][0-9]-[0-9][0-9]-[0-9][0-9]" ↪
↪ (file-name-base filename))
          (display-buffer (find-file-noselect filename))
          (read-string "Description: "))))
      (let ((new-filename (if (string= (or description "") "")
↪
↪                                     nil
↪                                     (expand-file-name
↪                                     (concat (file-name-base filename) " " description
↪                                     (file-name-extension filename))
↪                                     (file-name-directory filename))))
        (if new-filename
          (rename-file filename new-filename))
        (unless (string-match "new toot" (buffer-name)) ; can't match off major mode yet
          (mastodon-toot))
        (mastodon-toot--attach-media
          (or new-filename filename) "image/png"
          (or description
            (when (string-match ↪
↪ "^[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]_[0-9][0-9]-[0-9][0-9]-[0-9][0-9]" ↪
↪ "\\(\\.+\\)" (save-match-data (file-name-base filename)))
              (match-string 1 (save-match-data (file-name-base ↪
↪ filename))))))))))

```

12.14 Mastodon keyboard shortcuts via Hydra

Based on <https://github.com/holgerschurig/emacs-doom-config/blob/master/config.el#L2397>

```

;; Not in the following hydra, but mentioned in "M-x describe-mode". Also, the README.org
;; contains several functions that aren't in my hydra.
;;
;; TAB                mastodon-tl--next-tab-item
;; D                  mastodon-toot--delete-and-redraft-toot
;; C-S-b              mastodon-tl--unblock-user
;; S-TAB              mastodon-tl--previous-tab-item
;; S-RET              mastodon-tl--unmute-user
;; C-S-w              mastodon-tl--unfollow-user
;; S-SPC              scroll-down-command
;; <backtab>           mastodon-tl--previous-tab-item
;; C-M-i              mastodon-tl--previous-tab-item
;; M-n                mastodon-tl--next-tab-item
;; M-p                mastodon-tl--previous-tab-item

(defhydra sacha-mastodon-help (:color blue :hint nil)
  "
Timelines^^  Toots^^^^  Own Toots^^  Profiles^^  Users/Follows^^  Misc^^
^^-----^^-----^^-----^^-----^^-----^^-----

```

```

_h_ome      _n_ext _p_rev      _r_epl_y      _A_uthors      follo_W_      _X_ lists
_l_ocal      _T_hread of toot^^ wri_t_e      user _P_rofile _N_otifications f_I_lter
_Federated    (un) _b_oost^^      _e_dit      ^^      _R_equests      _C_opy URL
fa_V_ories    (un) _f_avorite^^  _d_elete      _O_wn      su_G_estions      _S_earch
_#_ tagged    (un) p_i_n^^      ^^      _U_pdate own      _M_ute user      _H_elp
_@_ mentions  (un) boo_k_mark^^  show _E_dits  ^^      _B_lock user
boo_K_marks   _v_ote^^
trendin_g_
_u_pdate      _w_rite Emacs news _o_rg _s_screenshot
"

    ;; my custom stuff
    ("s" sachamastodon-toot-screenshot)
    ("w" sachamastodon-save-toot-for-emacs-news)
    ("o" (org-capture nil "m"))
    ;; more general things
    ("h" (progn (require 'mastodon) mastodon-tl--get-home-timeline))

    ("l" mastodon-tl--get-local-timeline)
    ("F" mastodon-tl--get-federated-timeline)
    ("V" mastodon-profile--view-favourites)
    ("#" mastodon-tl--get-tag-timeline)
    ("@" (progn (require 'mastodon) (mastodon-notifications-get-mentions)))
    ("K" mastodon-profile--view-bookmarks)
    ("g" mastodon-search--trending-tags)
    ("u" mastodon-tl--update :exit nil)

    ("n" mastodon-tl--goto-next-toot)
    ("p" mastodon-tl--goto-prev-toot)
    ("T" mastodon-tl--thread)
    ("b" mastodon-toot--toggle-boost :exit nil)
    ("f" mastodon-toot--toggle-favourite :exit nil)
    ("i" mastodon-toot--pin-toot-toggle :exit nil)
    ("k" mastodon-toot--bookmark-toot-toggle :exit nil)
    ("c" mastodon-tl--toggle-spoiler-text-in-toot)
    ("v" mastodon-tl--poll-vote)

    ("A" mastodon-profile--get-toot-author)
    ("P" mastodon-profile--show-user)
    ("O" mastodon-profile-my-profile)
    ("U" mastodon-profile--update-user-profile-note)

    ("W" mastodon-tl--follow-user)
    ("N" mastodon-notifications-get)
    ("R" mastodon-profile--view-follow-requests)
    ("G" mastodon-tl--get-follow-suggestions)
    ("M" mastodon-tl--mute-user)
    ("B" mastodon-tl--block-user)

    ("r" mastodon-toot--reply)
    ("t" mastodon-toot)
    ("e" mastodon-toot--edit-toot-at-point)
    ("d" mastodon-toot--delete-toot)
    ("E" mastodon-toot--view-toot-edits)

    ("I" mastodon-tl--view-filters)
    ("X" mastodon-tl--view-lists)
    ("C" mastodon-toot--copy-toot-url)
    ("S" mastodon-search--search-query)
    ("H" describe-mode)

```

```

    ("q" nil :exit t)
  )
  (use-package mastodon
    :bind ("s-m" . sachamastodon-help/body))

```

12.15 Making it easier to toot my config

The following snippet composes a toot buffer with a link to the relevant section of my configuration file, or to the relevant blog post if specified.

```

;;;###autoload
(defun sachamastodon-toot-config (&optional include-screenshot)
  "Toot this part of my config."
  (interactive (list current-prefix-arg))
  (let ((link (if (org-entry-get (point) "EXPORT_ELEVENTY_PERMALINK")
                  (concat ←
                        "https://sachachua.com" (org-entry-get (point) "EXPORT_ELEVENTY_PERMALINK"))
                  (concat ←
                        "https://sachachua.com/dotemacs/#" (org-entry-get (point) "CUSTOM_ID"))))
        text)
    (save-excursion
      (org-back-to-heading)
      (org-end-of-meta-data)
      (setq text (buffer-substring (point) (org-end-of-subtree))))
    (mastodon-toot)
    (insert text "\n\nLink: " link)))

```

12.16 Org contacts

12.16.1 Capture

```

;;;###autoload
(defun sachamastodon-org-contact-add ()
  "Add current toot author as a contact."
  (interactive)
  (let-alist (get-text-property (point) 'item-json)
    (with-current-buffer (find-file-noselect (car org-contacts-files))
      (if (org-find-property "MASTODON" .account.acct)
          (message "Already exists.")
          (org-insert-heading)
          (insert (format "%s\n:PROPERTIES:\n:NAME: %s\n:MASTODON: ←
↵ %s\n:ALIAS: %s\n:END:\n"
                        .account.display_name ←
                        .account.display_name ←
                        .account.acct ←
                        .account.username)))
          (message "Added %s" .account.acct)))))

```

12.16.2 Completion


```

;;;###autoload
(defun sacha-org-contacts-complete-mastodon (string)
  (let* ((completion-ignore-case org-contacts-completion-ignore-case)
        (completion-list
         (cl-loop for contact in (org-contacts-filter)
                   ;; The contact name ←
                   ;; returned by ←
                   for contact-name = ←
                   ;; Build the list of ←
                   for ignore-list = ←
                   (or (cdr (assoc-string ←
                                         ←
                                         (nth 2 ←
                                         ;; Build the list of ←
                                         for handle-list = ←
                                         ignore-list ←
                                         ←
                                         (or (cdr ←
                                         ←
                                         (nth 2 ←
                                         nconc (cl-loop for ←
                                         ←
                                         collect (format "%s (%s)" contact-name ←
                                         (completion-list (org-contacts-all-completions-prefix
                                         string
                                         (org-uniquify completion-list))))
                                         (when completion-list
                                           (org-contacts-make-collection-prefix completion-list))))))
    (when completion-list
      (org-contacts-make-collection-prefix completion-list))))))
  (when completion-list
    (org-contacts-make-collection-prefix completion-list))))

;;;###autoload
(defun sacha-mastodon-complete-contact ()
  "Suitable for adding to `completion-at-point-functions'."
  (interactive)
  (let ((beg
         (save-excursion
           (re-search-backward "\\(\\`\\|\\[\\n:,]\\|\\)[ \\t]*"

```

```

                                (goto-char (match-end 0))
      (point)))
      (end (point)))
    (list beg
      end
      (completion-table-dynamic
        (lambda (string)
          (sacha-org-contacts-complete-mastodon string))))))

(with-eval-after-load 'mastodon-toot
  (with-eval-after-load 'org-contacts
    (add-hook 'mastodon-toot-mode-hook
              (lambda ()
                (add-hook ↵
                  ↵
                  'completion-at-point-functions
                  ↵
                  #'sacha-mastodon-complete-contact nil t))))))

```

12.17 Copy Mastodon toot URL as author link

```

;;;###autoload
(defun sacha-mastodon-copy-toot-as-author-link ()
  (interactive)
  (let* ((url (mastodon-toot--toot-url))
         (handle (concat "@"))
         ↵
         (let-alist (or (mastodon-tl--property 'base-toot)
                        ↵
                        ↵
                        (mastodon-tl--property 'item-json)
                        ↵
                        .account.acct))))
    ;; figure out how to properly add to org-stored-links someday
    (kill-new (org-link-make-string url handle))
    (message "Link stored (%s, %s)." handle url)))

```

12.18 Collect my recent toots in an Org file so that I can refile them

I want to use my microblog posts on Mastodon as building blocks for longer posts on my blog. Getting them into an Org file makes it easier to link to them or refile them to other parts of my Org files so that I can build up my notes.

```

(use-package pandoc :defer t)
(advice-add #'org-feed-add-items :after #'sacha-org-feed-sort)
(setq org-feed-alist '(("Mastodon" "https://emacs.ch/@sachac/with_replies.rss"
  ↵
  "~/sync/orgzly/toots.org" "Toots"
  ↵
  :formatter sacha-mastodon-org-feed-formatter))))

;;;###autoload
(defun sacha-mastodon-org-feed-formatter (entry)
  (concat "* " (pandoc-convert-stdio

```

```
(dom-text (dom-by-tag  
↪  
  
↪      (with-temp-buffer  
↪  
↪          (insert "<item>"  
↪  
↪              (plist-get entry :item-full-text)  
↪              "<>/item>")  
↪              (xml-parse-region (point-min) (point-max)))  
↪              'description))  
↪              "html" "org")  
↪              "\n\n[" (format-time-string (cdr org-time-stamp-formats)  
↪              (date-to-time ↪  
↪              (plist-get entry :pubDate))))  
↪              "]\\n" (plist-get entry :link)))
```

```
; ; ; ##autoload  
(defun sachas-feed-sort (pos entries)  
  (save-excursion  
    (goto-char pos)  
    (when (looking-at org-complex-heading-regexp)  
      (org-sort-entries nil ?T))))
```

Now I can use `org-feed-update-all` (`C-c C-x g`) to pull things into my toots.org file.

12.19 Archive toots on my blog

I want to compile my public microblog posts into weekly posts so that they're archived on my blog. It might make sense to make them list items or subtrees so that I can move them around easily.

[illegible]

```

;; (format ←
↳ "#+begin_quote\n#+begin_export html\n%s\n#+end_export\n#+end_quote\n\n%s\n\n"
;;
↳ (assoc-default 'content o)
;;
↳ (org-link-make-string (assoc-default 'url o) (assoc-default 'created_at o)))
)
(seq-filter
  (lambda (o)
    (string= (assoc-default 'visibility ←
↳ o) "public")))
(sacha-mastodon-fetch-posts-after
  (format ←
↳ "accounts/%s/statuses?count=40&exclude_reblogs=t" (mastodon-auth--get-account-id))
  date))
""))

;;;###autoload
(defun sacha-mastodon-insert-sacha-toots-since (date)
  (interactive (list (org-read-date nil nil nil "Since date: ")))
  (insert (sacha-mastodon-format-sacha-toots-since date)))

;;;###autoload
(defun sacha-mastodon-roundup (date)
  (interactive (list (org-read-date nil nil nil "Date of post: ")))
  (org-insert-heading)
  (let ((start (org-read-date nil nil "--wed" nil (date-to-time date))))
    (insert "Wednesday weblog: Toots ending " start " :review:weblog:\n\n")
    (sacha-mastodon-insert-sacha-toots-since start)))

```

13 Web

```
(setq browse-url-firefox-program "firefox")
```

13.1 Emacs: Open URLs or search the web, plus browse-url-handlers emacs.org

- [2025-07-14 Mon]: Naturally, [do this only with text you trust](#). =)
- [2025-07-12 Sat]: Use `cl-pushnew` instead of `add-to-list`, correct `browse-url-browser-browser-function` to `browse-url-browser-function`, and add an example for `eww`.

On IRC, someone asked for help configuring Emacs to have a keyboard shortcut that would either open the URL at point or search the web for the region or the word at point. I thought this was a great idea that I would find pretty handy too.

Let's write the interactive function that I'll call from my keyboard shortcut.

- First, let's check if there's an active region. If there isn't, let's assume we're looking at the thing at point (could be a URL, an e-mail address, a filename, or a word).
- If there are links, open them.
- Otherwise, if there are e-mail addresses, compose a message with all those email addresses in the "To" header.

- Are we at a filename? Let's open that.
- Otherwise, do a web search. Let's make that configurable. Most people will want to use a web browser to search their favorite search engine, such as DuckDuckGo or Google, so we'll make that the default.

```
(defcustom sacha-search-web-handler "https://duckduckgo.com/html?q="
  "How to search. Could be a string that accepts the search query at the end (URL-encoded)
  or a function that accepts the text (unencoded)."
  :type '(choice (string :tag "Prefix URL to search engine.")
               (function :tag "Handler"
                           ↵ function.)))))

;;;###autoload
(defun sacha-open-url-or-search-web (&optional text-or-url)
  (interactive (list (if (region-active-p)
                        ↵ (buffer-substring (region-beginning) (region-end))
                        ↵ (or
                           ↵ (and (derived-mode-p 'org-mode)
                                ↵ (let ((elem (org-element-context)))
                                ↵ (and (eq (org-element-type elem) 'link)
                                ↵ (buffer-substring-no-properties
                                ↵ (org-element-begin elem)
                                ↵ (org-element-end elem))))))
                        ↵ (thing-at-point 'url)
                        ↵ (thing-at-point 'email)
                        ↵ (thing-at-point 'filename)
                        ↵ (thing-at-point 'word))))))
  (catch 'done
    (let (links)
      (with-temp-buffer
        (insert text-or-url)
        (org-mode)
        (goto-char (point-min))
        ;; We add all the links to a list first because ↵
        ↵ following them may change the point
        (while (re-search-forward org-any-link-re nil t)
          (cl-pushnew (match-string-no-properties 0) ↵
            links))
        (when links
          (dolist (link links)
            (org-link-open-from-string link))
          (throw 'done links))
        ;; Try emails
        (while (re-search-forward thing-at-point-email-regexp ↵
          (cl-pushnew (match-string-no-properties 0) ↵
            links))
          nil t)
        links)))
```

```

                (when links
                  (compose-mail (string-join links ", "))
                  (throw 'done links)))
;; Open filename if specified, or do a web search
(cond
 ((ffap-guesser) (find-file-at-point))
 ((functionp sacha-search-web-handler)
  (funcall sacha-search-web-handler text-or-url))
 ((stringp sacha-search-web-handler)
  (browse-url (concat sacha-search-web-handler ←
    ↪ (url-hexify-string text-or-url))))))

```

I've been really liking how [consult-omni](#) lets me do quick searches as I type from within Emacs, which is actually really cool. I've even extended it to search my bookmarks as well, so that I can find things using my words for them and not trust the internet's words for them. So if I wanted to search using consult-omni, this is how I would do it instead.

```
(setopt sacha-search-web-handler #'consult-omni)
```

Now I can bind that to `C-c o` in my config with this bit of Emacs Lisp.

```
(keymap-global-set "C-c o" #'sacha-open-url-or-search-web)
```

Here's a quick demo:

https://sachachua.com/blog/2025/07/emacs-open-urls-or-search-the-web-plus-browse-url-handlers/2025-07-11_7.07.10.webm?caption=Screencastshowingitinuse

nil

Depending on the kind of URL, I might want to look at it in different browsers. For example, some websites like <https://emacswiki.org> work perfectly fine without JavaScript, so opening them in EWW (the Emacs Web Wowser) is great. Then it's right there within Emacs for easy copying, searching, etc. Some websites are a little buggy when run in anything other than Chromium. For example, [MailChimp](#) and [BigBlueButton](#) (which is the [webconference server we use for EmacsConf](#)) both behave a bit better under [Google Chrome](#). There are some URLs I want to ignore because they don't work for me or they tend to be too paywalled, like [permalink.gmane.org](#) and [medium.com](#). I want to open Mastodon URLs in [mastodon.el](#). I want to open the rest of the URLs in Firefox, which is my current default browser.

To change the way Emacs opens URLs, you can customize `browse-url-browser-function` and `browse-url-handlers`. For example, to set up the behaviour I described, I can use:

```

(setq browse-url-handlers
  (seq-union
   browse-url-handlers
   '(("https?://?medium\\.com" . ignore)
     ("https?:[/^]/+/@[/^]/+/.*" . mastodon-url-lookup)
     ("https?://mailchimp\\.com" . browse-url-chrome)
     ("https?://bbb\\.emacsverse\\.org" . browse-url-chrome)
     ("https?://emacswiki.org" . eww))))
(setopt browse-url-browser-function 'browse-url-firefox)

```

If you wanted to use EWW as your default web browser, you could use `(setopt browse-url-browser-function 'eww)` instead.

Could be a fun tweak. I wonder if something like this might be handy for other people too!

13.2 Checking URLs

Out of this discussion: <https://fosstodon.org/@fullyabstract/116092736652067422>

```
(defvar sach-check-dead-links-skip
  (regexp-opt
    '("news.ycombinator"
      "meejah.ca"
      "reddit"))
  "Some sites block automated link checking.")
(defvar sach-check-dead-links-user-agent "Emacs link checker"
  "Some sites may let certain user agents through. Change this as desired.")

;;;###autoload
(defun sach-check-dead-links ()
  "Look for dead links."
  (interactive)
  (let (pos)
    (setq pos
      (catch 'done
        (while (re-search-forward ffap-url-regexp nil t)
          (let* ((start (match-beginning 0))
                 (end (match-end 0))
                 (url (or (and (derived-mode-p 'org-mode)
                               (org-element-property :raw-link (org-element-context)))
                          (ffap-url-at-point)))
                 (browse-url-handlers nil)
                 (url-user-agent sach-check-dead-links-user-agent))
            (message "Checking %s" url)
            (unless (or (string-match sach-check-dead-links-skip url)
                       (save-match-data
                         (condition-case nil
                           (url-file-exists-p url)
                           (error nil))))
              (goto-char start)
              (browse-url url)
              (if (y-or-n-p "Continue?")
                  (goto-char end)
                  (throw 'done (cons url (point)))))))
          (goto-char (cdr pos))))))
```

13.3 Search

```
(use-package engine-mode
  :defer t
  :config
  (defengine sach-blog "https://www.google.ca/search?q=site:sachachua.com+%s" :keybinding "b")
  (defengine mail "https://mail.google.com/mail/u/0/#search/%s" :keybinding "m")
  (defengine google "https://google.com/search?q=%s" :keybinding "g")
  (defengine emacs wiki "https://google.com/search?q=site:emacs wiki.org+%s" :keybinding "e")
  (engine-mode)
  :hydra
  (sach-engine-mode-hydra
   (:color blue)
   "Engine mode"
   ("b" engine/search-sach-blog "blog")
   ("m" engine/search-mail "mail")
```

```
( "g" engine/search-google "google")
( "e" engine/search-emacswiki "emacswiki"))
```

13.4 Parsing RSS and Atom feeds

Related: [Reading more blogs](#); Emacs Lisp: [Listing blogs based on an OPML file](#)

```
; ;; ###autoload
(defun sacha-rss-get-entries (url)
  "Return a list of the form ((:title ... :url ... :date ...) ...)."
  (with-current-buffer (url-retrieve-synchronously url)
    (set-buffer-multibyte t)
    (goto-char (point-min))
    (when (re-search-forward "<\\?xml\\|<rss" nil t)
      (goto-char (match-beginning 0))
      (sort
        (let* ((feed (xml-parse-region (point) (point-max)))
               (is-rss (> (length (xml-get-children ←
                                   (car feed) 'entry)) 0)))
          (if is-rss
              (mapcar
                (lambda (entry)
                  (list
                    :url
                    (or
                     (xml-get-attribute
                      (car
                        (seq-filter ←
                          (λ (x) (string= (xml-get-attribute x 'rel) "alternate"))
                            (xml-get-children entry 'link))
                        'href)
                     (dom-text (dom-by-tag ←
                                  entry 'guid))))
                    :title
                    (elt (car (xml-get-children ←
                              entry 'title)) 2)
                    :date
                    (date-to-time (elt (car ←
                                         (xml-get-children entry 'updated)) 2))))
                (xml-get-children (car feed) 'entry))
              (mapcar (lambda (entry)
                         (list
                           :url
                           (or ←
                             (caddr (car (xml-get-children entry 'link)))
                             (dom-text (dom-by-tag entry 'guid)))
                           :title
                           (caddr ←
                             (car (xml-get-children entry 'title)))
                           :date
                           (←
                             (date-to-time (elt (car (xml-get-children entry 'pubDate)) 2))))
```



```

      (xml-get-children (car feed) 'channel)) 'item))))
      :key (lambda (o) (plist-get o :date))
      :lessp #'time-less-p
      :reverse t))))

```

Making an OPML table:

```

;; ##autoload
(defun sacha-opml-table (xml)
  (sort
   (mapcar
    (lambda (o)
      (let ((latest (car (condition-case nil (sacha-rss-get-entries
        (dom-attr o 'xmlUrl))
        (error nil))))))
        (list
         (if latest
          (format-time-string "%Y-%m-%d" (plist-get
            latest :date))
          ""))
          (org-link-make-string
           (or (dom-attr o 'htmlUrl)
              (dom-attr o 'xmlUrl))
           (replace-regexp-in-string " *|" "" (dom-attr o 'text)))
           (if latest
            (org-link-make-string
             (plist-get latest :url)
             (or (plist-get latest :title)
                  ""))))
          ""))))
    (dom-search
     xml
     (lambda (o)
       (and
        (eq (dom-tag o) 'outline)
        (dom-attr o 'xmlUrl)
        (dom-attr o 'text))))
    :key #'car
    :reverse t))

```

13.5 Spookfox

13.5.1 Link to current webpage from Spookfox

toblog

```

;; ##autoload
(defun sacha-org-spookfox-complete ()
  (spookfox-js-injection-eval-in-active-tab "window.location.href" t))

```

```

(with-eval-after-load 'org
  (org-link-set-parameters
   "spookfox"
   :complete #'sacha-org-spookfox-complete
   :insert-description #'sacha-org-link-insert-description))

```

13.5.2 STARTED Link to currently-selected text using Spookfox

```
;;;###autoload
(defun sacha-spookfox-link-to-fragment ()
  (interactive)
  (let ((url
        (spookfox-js-injection-eval-in-active-tab ↵
         ↵ "window.location.href + '#::~text=' + ↵
         ↵ encodeURIComponent(window.getSelection().toString())" t)))
    (when (called-interactively-p 'any)
      (insert url))
    url))
```

13.5.3 Running the current Org Mode Babel Javascript block from Emacs using Spookfox em-bark:emacs:org:spookfox

I often want to send Javascript from Emacs to the web browser. It's handy for testing code snippets or working with data on pages that require Javascript or authentication. I could start Google Chrome or Mozilla Firefox with their remote debugging protocols, copy the websocket URLs, and talk to the browser through something like Puppeteer, but it's so much easier to use the [Spookfox](#) extension for Mozilla to execute code in the active tab. `spookfox-js-injection-eval-in-active-tab` lets you evaluate Javascript and get the results back in Emacs Lisp.

I wanted to be able to execute code even more easily. This code lets me add a `:spookfox t` parameter to Org Babel Javascript blocks so that I can run the block in my Firefox active tab. For example, if I have `(spookfox-init)` set up, Spookfox connected, and <https://planet.emacslife.com> in my active tab, I can use it with the following code:

```
#+begin_src js :eval never-export :spookfox t :exports results
[ ... document.querySelectorAll('.post > h2') ].slice(0,5).map((o) => '- ' + ↵
  ↵ o.textContent.trim().replace(/[ \n]+/g, ' ') + '\n').join('')
#+end_src
```

[https://sachachua.com/blog/2024/01/running-the-current-org-mode-babel-javascript-block-in-spookfox/2024-01-1908.16.31.webm?caption = Evaluating Javascript block with : spookfox](https://sachachua.com/blog/2024/01/running-the-current-org-mode-babel-javascript-block-in-spookfox/2024-01-1908.16.31.webm?caption=Evaluating%20Javascript%20block%20with%20spookfox)

To do this, we wrap some advice around the `org-babel-execute:js` function that's called by `org-babel-execute-src-block`.

```
;;;###autoload
(defun sacha-org-babel-execute:js-spookfox (old-fn body params)
  "Maybe execute Spookfox."
  (if (assq :spookfox params)
      (spookfox-js-injection-eval-in-active-tab
        body t)
      (funcall old-fn body params)))
```

```
(with-eval-after-load 'ob-js
  (advice-add 'org-babel-execute:js :around #'sacha-org-babel-execute:js-spookfox))
```

I can also run the block in Spookfox without adding the parameter if I make an interactive function:

```
;;;###autoload
(defun sacha-spookfox-eval-org-block ()
  (interactive))
```

```

(let ((block (org-element-context)))
  (when (and (eq (org-element-type block) 'src-block)
              (string= (org-element-property ↵
                        ↵ :language block) "js")))
    (spookfox-js-injection-eval-in-active-tab
      (nth 2 (org-src--contents-area block))
      t))))

```

I can add that as an Embark context action:

```

(with-eval-after-load 'embark-org
  (define-key embark-org-src-block-map "f" #'sacha-spookfox-eval-org-block))

```

In Javascript buffers, I want the ability to send the current line, region, or buffer too, just like [nodejs-repl](#) does.

```

;;;###autoload
(defun sacha-spookfox-send-region (start end)
  (interactive "r")
  (spookfox-js-injection-eval-in-active-tab (buffer-substring start end) t))

;;;###autoload
(defun sacha-spookfox-send-buffer ()
  (interactive)
  (sacha-spookfox-send-region (point-min) (point-max)))

;;;###autoload
(defun sacha-spookfox-send-line ()
  (interactive)
  (sacha-spookfox-send-region (line-beginning-position) (line-end-position)))

;;;###autoload
(defun sacha-spookfox-send-last-expression ()
  (interactive)
  (sacha-spookfox-send-region (save-excursion (nodejs-repl--beginning-of-expression)) ↵
    ↵ (point)))

(defvar-keymap sacha-js-spookfox-minor-mode-map
  :doc "Send parts of the buffer to Spookfox."
  "C-x C-e" 'sacha-spookfox-send-last-expression
  "C-c C-j" 'sacha-spookfox-send-line
  "C-c C-r" 'sacha-spookfox-send-region
  "C-c C-c" 'sacha-spookfox-send-buffer)

(define-minor-mode sacha-js-spookfox-minor-mode "Send code to Spookfox.")

```

I usually edit Javascript files with [js2-mode](#), so I can use `sacha-js-spookfox-minor-mode` in addition to that.

I can turn the minor mode on automatically for `:spookfox t` source blocks. There's no `org-babel-edit-prep:js` yet, I think, so we need to define it instead of advising it.

```

;;;###autoload
(defun org-babel-edit-prep:js (info)
  (when (assq :spookfox (nth 2 info))
    (sacha-js-spookfox-minor-mode 1)))

```

Let's try it out by sending the last line repeatedly:

https://sachachua.com/blog/2024/01/running-the-current-org-mode-babel-javascript-block-in-spookfox/2024-01-19_8.26.35.webm?caption=Sendingthecurrentline

I used to do this kind of interaction with [Skewer](#), which also has some extra stuff for evaluating CSS and HTML. Skewer hasn't been updated in a while, but maybe I should also check that out again to see if I can get it working.

Anyway, now it's just a little bit easier to tinker with Javascript!

13.5.4 Using Spookfox to scroll Firefox up and down from Emacs

web:emacs

I open lots of pages in the process of making Emacs News. I like to open the pages in Mozilla Firefox, but I want the keyboard focus to stay with Emacs so that I can quickly categorize the links. I also sometimes want to scroll the page up or down. While reading the [Reading, and not forgetting](#) post, I came across [Spookfox](#), which bridges Emacs and Firefox using an Firefox add-on and websockets. After I started spookfox and connected to it by clicking on the extension in Firefox, I was able to interact with it from Emacs Lisp. I feel a little nervous about it security-wise, but at least it's only listening on the local port. There might be another way to do it with the Marionette support in Firefox, but I haven't looked into it yet.

```
(use-package spookfox
  :files ("lisp/*.el" "lisp/apps/*.el"))
  :load-path ("~/vendor/spookfox/lisp" "~/vendor/spookfox/lisp/apps")
  :when sacha-laptop-p
  :config
  (require 'spookfox-tabs)
  ;(require 'spookfox-org-tabs)
  (require 'spookfox-js-injection)
  (add-to-list 'spookfox-enabled-apps 'spookfox-tabs)
  (with-eval-after-load 'spookfox-org-tabs (add-to-list 'spookfox-enabled-apps ↵
    ↵ 'spookfox-org-tabs))
  (add-to-list 'spookfox-enabled-apps 'spookfox-js-injection)
  ;; (spookfox-init) ; don't automatically enable it; run (spookfox-init) to manually ↵
  ↵ enable
  )
```

Anyway, this code seems to do the job of scrolling my Firefox window:

```
;; ##autoload
(defun sacha-spookfox-scroll-down ()
  (interactive)
  (spookfox-js-injection-eval-in-active-tab "window.scrollTo(0, ↵
    ↵ document.documentElement.clientHeight);" t))

;; ##autoload
(defun sacha-spookfox-scroll-up ()
  (interactive)
  (spookfox-js-injection-eval-in-active-tab "window.scrollTo(0, ↵
    ↵ -document.documentElement.clientHeight);"))
```

```
(keymap-global-set "C-s-v" 'sacha-spookfox-scroll-down)
(keymap-global-set "S-s-v" 'sacha-spookfox-scroll-up)
```

This code opens a tab without switching keyboard focus away from Emacs:

```

;;;###autoload
(defun sacha-spookfox-background-tab (url &rest args)
  "Open URL as a background tab."
  (if spookfox--connected-clients
      (spookfox-tabs--request (cl-first spookfox--connected-clients) ←
        ↪ "OPEN_TAB" `(:url ,url))
      (browse-url url)))

```

My Emacs News code for processing my upvoted Reddit posts can automatically grab the links from Reddit link posts, but sometimes people post Reddit text or image posts and then include the link to the actual project in the post body or a comment instead.

```

;;https://emacs.stackexchange.com/questions/41801/how-to-stop-completing-read-ivy-completing-read-from-sorting
;;;###autoload
(defun sacha-presorted-completion-table (completions)
  (lambda (string pred action)
    (if (eq action 'metadata)
        '(metadata
          (cycle-sort-function . identity)
          (display-sort-function . identity))
        (complete-with-action action completions string pred))))

```

```

;;;###autoload
(defun sacha-spookfox-get-links ()
  (seq-uniq
   (spookfox-eval-js-in-active-tab ←
    ↪ "[ ... (document.querySelector('[data-testid=post-container'])?parentElement || ↪
    ↪ document).querySelectorAll('a')].map(a => a.href).filter(a => a && ↪
    ↪ (!window.location.host.match(/reddit/) || !a.match(/redd\?.?it/)) && ↪
    ↪ !a.match(window.location.host))" t)))

;;;###autoload
(defun sacha-spookfox-complete-link (&optional prompt)
  (completing-read
   (or prompt "Link: ")
   (sacha-presorted-completion-table
    (sacha-spookfox-get-links))))

;;;###autoload
(defun sacha-spookfox-insert-link-from-page (link)
  (interactive (list (sacha-spookfox-complete-link)))
  (insert (org-link-make-string link (sacha-page-title link))))

;;;###autoload
(defun sacha-spookfox-open-link-from-page (link)
  (interactive (list (sacha-spookfox-complete-link)))
  (sacha-spookfox-background-tab link))

;;;###autoload
(defun sacha-spookfox-insert-link-to-tab ()
  (interactive)
  (let ((tab (spookfox-request-active-tab)))
    (insert (org-link-make-string
              (plist-get tab :url)
              (plist-get tab :title)))))

```

13.5.5 Emacs and Spookfox: org-capture the current tab from Firefox or a link from the page

I want to quickly capture notes based on the current tab in Firefox or a link from the page's main body. I have the [Org Capture Firefox extension](#) and `Ctrl-Shift-L` seems to be the keyboard shortcut for capturing with it, so I probably just have to get the hang of using it.

I also want to make it easier to add notes even when I've already switched back to Emacs. I could use `s-2` to shift to Firefox (I have some Autokey shortcuts for focusing specific applications; `s-1` is Emacs), but sometimes I just want to add a link at point.

```
;;###autoload
(defun sacha-spookfox-insert-url ()
  (interactive)
  (insert (spookfox-js-injection-eval-in-active-tab "window.location.href" t)))
;;###autoload
(defun sacha-spookfox-insert-org-link ()
  (interactive)
  (insert (apply #'org-link-make-string
                 (append ↵
                    ↵ (spookfox-js-injection-eval-in-active-tab "[window.location.href, document.title]" t) ↵
                    ↵ nil))))
```

```
(with-eval-after-load 'org-capture
  (cl-pushnew
    `("f" "Firefox" entry
      (file ,sacha-org-inbox-file)
      "* %^{Note}\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n%(apply ↵
        ↵ #'org-link-make-string
        ↵ (append ↵
            ↵ (spookfox-js-injection-eval-in-active-tab "[window.location.href, document.title]" ↵
            ↵ t) nil)))"
      org-capture-templates)
    (cl-pushnew
      `("F" "Firefox link" entry
        (file ,sacha-org-inbox-file)
        "* %^{Note}\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n%(org-link-make-string
          (sacha-spookfox-complete-link)))"
        org-capture-templates))
```

This uses `sacha-spookfox-complete-link` from [spookfox](#) (p. 589).

14 Self-tracking, statistics, and other data transformations

14.1 Quantified Awesome

```
(require 'quantified nil t)

(defmacro sacha-org-with-current-task (&rest body)
  "Execute BODY with the point at the subtree of the current task."
  (declare (debug t))
  `(if (derived-mode-p 'org-agenda-mode)
      (save-window-excursion
        (org-agenda-switch-to)
        ,@body)
      ,@body))
```

```

;;;###autoload
(defun sacha-org-clock-in-and-track ()
  "Start the clock running. Clock into Quantified Awesome."
  (interactive)
  (sacha-org-with-current-task
   (org-clock-in)
   (call-interactively 'sacha-org-quantified-track)
   (when (websocket-openp obs-websocket) (sacha-stream-message (org-get-heading t t t t)))
   (cond
    ((org-entry-get (point) "AUTO")
     (org-link-open-from-string (org-entry-get (point) "AUTO"))))
    (t
     (save-restriction
      (org-narrow-to-subtree)
      (org-next-link)
      (when (looking-at org-link-any-re)
       (org-open-at-point)))))))

(defmacro sacha-with-org-task (&rest body)
  "Run BODY within the current agenda task, clocked task, or cursor task."
  `(cond
   ((derived-mode-p 'org-agenda-mode)
    (let* ((marker (org-get-at-bol 'org-marker))
           (buffer (marker-buffer marker))
           (pos (marker-position marker)))
      (with-current-buffer buffer
        (save-excursion
         (save-restriction
          (widen)
          (goto-char pos)
          ,@body))))))
   ((and (derived-mode-p 'org-mode) (org-at-heading-p)) (save-excursion ,@body))
   ((org-clocking-p) (save-excursion (org-clock-goto) ,@body))
   ((derived-mode-p 'org-mode) ,@body)))

(defvar sacha-org-quantified-regexps
  '(("emacsconf" . "Emacs | Emacsconf")
    ("emacs" . "Emacs")
    ("consulting" . "E1 Gen"))
  "Alist of regexp . category.")

;;;###autoload
(defun sacha-org-quantified-track (&optional category note)
  "Create a tracking record using CATEGORY and NOTE.
   Default to the current task in the agenda, the currently-clocked
   entry, or the current subtree in Org."
  (interactive (list nil nil))
  (unless (and category note)
    (sacha-with-org-task
     (setq category (or category
                       (org-entry-get-with-inheritance "QUANTIFIED"))))
    (cond
     ((null category)
      (let* ((heading (org-get-heading))
              (guess (seq-find (lambda (entry)
                                (string-match (car entry)
                                              (org-get-heading))))
              (string-match (car entry)
                            (org-get-heading))))
        (org-entry-get-with-inheritance "QUANTIFIED")))))

```

```

↩↩
↩↩      heading))
↩↩
↩↩      sach-org-quantified-regexps)))
      (setq category (or (cdr guess) (read-string "Category: ")))
      (org-set-property "QUANTIFIED" category))
      ((string= category ' "ask")
       (setq category (read-string "Category: ")))
      (setq note
        (concat
          (if (string= (or (org-entry-get-with-inheritance "QUANTIFIEDQUIET") "") "t")
              "!private "
              "")
          (or note (elt (org-heading-components) 4) (read-string "Note: "))))
      (quantified-track (concat category " | " note)))

;;;###autoload
(defun sach-org-quick-clock-in-task (location jump)
  "Track and clock in on the specified task.
   If JUMP is non-nil or the function is called with the prefix argument, jump to that ↩↩
   ↩↩ location afterwards."
  (interactive (list (save-excursion (sach-org-refile-get-location "Location")) ↩↩
    ↩↩ current-prefix-arg))
  (when location
    (if jump
        (progn (org-refile 4 nil location) (sach-org-clock-in-and-track))
        (save-window-excursion
          (org-refile 4 nil location)
          (sach-org-clock-in-and-track)))))

(keymap-global-set "C-c q" 'sach-org-quick-clock-in-task)
(with-eval-after-load 'org
  (keymap-set org-agenda-mode-map "!" 'sach-org-clock-in-and-track))

```

14.1.1 Childtime!

```

;;;###autoload
(defun sach-childcare ()
  (interactive)
  (unwind-protect
    (when (org-clocking-p)
      (org-clock-out)))
  (quantified-track "Childcare"))

```

14.1.2 Make a tablist of my time entries

```

(define-derived-mode sach-quantified-list-mode tablist-mode "Time"
  "Major mode for time entries"
  (setq tabulated-list-format [("id" 5)
    ↩↩
    ↩↩      ("timestamp" 25)
    ↩↩
    ↩↩      ("duration" 5)

```



```

↪          ("full_name" 60)
↪
↪          ("note" 20)])
    (tabulated-list-init-header)
    (tabulated-list-print t))

;;###autoload
(defun sacha-quantified-list (start end filter)
  (interactive (list (org-read-date nil nil nil "Start: ") (org-read-date nil nil nil
↪ "End: ")
↪                                     (read-string
↪ "Filter: ")))
  (switch-to-buffer (get-buffer-create "*quantified*"))
  (setq filter (and filter
↪                                     (not (string=
↪                                     (split-string
↪ filter " ")))
  (filter " ")))
  (let ((json-array-type 'list)
        (json-object-type 'alist))
    (setq tabulated-list-entries
          (seq-keep
            (lambda (o)
              (let-alist o
                (when (or (not filter)
↪
↪ (not (seq-find
↪
↪ (lambda (term)
↪
↪ (not
↪
↪ (or
↪
↪ (string-match term .full_name)
↪
↪ (string-match term (or .data.note ""))))))
↪ filter))))
          (list
            .id
            (vector
              (number-to-string .id)
              (format-time-string
↪ "%a %b %d %l:%M%p" (parse-iso8601-time-string .timestamp))
              (propertize (if
↪ .duration (format-secs "%h:%.2m" .duration) "")
↪ 'duration .duration)
              .full_name
              (or .data.note ""))))))
    (quantified-parse-json
      (quantified-request
        (format
↪
↪ "/records.json?start=%s&end=%s&auth_token=%s"
↪ (or start "")
↪ (or end ""))

```

```

                                (quantified-token))
                                nil "GET")))))
    (sacha-quantified-list-mode))
;;###autoload
(defun sacha-quantified-list-sum-marked-duration ()
  (interactive)
  (let ((seconds (apply '+
    ↪                                (mapcar
    ↪                                (lambda (o)
    ↪                                (get-text-property 0 'duration
    ↪                                ↪
    ↪                                (aref (cdr o) 2)))
    ↪                                (tablist-get-marked-items))))))
    (message "%s (%.1f)"
              (format-seconds "%d:%z%.2h:%.2m" seconds)
              (/ seconds 3600.0))))
;; (sacha-quantified-list "2024-09-30" nil "E1")

```

14.2 Compare times and effort estimates

This is for comparing times in column view and in tables.

```

;;###autoload
(defun sacha-compare-times (clocked estimated)
  (if (and (> (length clocked) 0) estimated)
      (format "%.2f"
        (/ (* 1.0 (org-hh:mm-string-to-minutes clocked))
           (org-hh:mm-string-to-minutes estimated)))
      ""))

```

Use with `#+COLUMNS: %40ITEM %17Effort(Estimated){:} %CLOCKSUM, #+BEGIN: columnview :hlines 1 ... #+END:`, and

```
#+TBLFM: $4='(sacha-compare-times $3 $2)
```

14.2.1 Using the `calendar-date-echo-text` variable to help plot a heatmap on a year-long calendar in Emacs

[images/output-2023-01-06-10:26:49.gif](#)

Building on [Display a calendar heat map using Emacs Lisp](#), I figured out how to use `calendar-date-echo-text` to store the date so that I can pick it up when plotting the heatmap:

```

;; This seems to be the only way we can hack the date in for now
(setq calendar-date-echo-text '(apply #'format (list "%04d-%02d-%02d" year month day)))

```

```

(defvar sacha-calendar-count-scaled)
;;###autoload
(defun sacha-calendar-heat-map-using-echo-text (&rest _)
  (when sacha-calendar-count-scaled
    (save-excursion

```

```

(goto-char (point-min))
(while (not (eobp))
  (let* ((help (get-text-property (point) 'help-echo))
         (next-change
          (or (next-single-property-change (point) 'help-echo)
              (point-max))))
    (inhibit-read-only t)
    (count-scaled (and help
                        (assoc-default help
                                        sachal-calendar-count-scaled))))
    (when (and help
                (string-match "[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]" help))
      (put-text-property (point) (+ 2 (point))
                          'face (intern (format "calendar-scale-%d"
                                                  count-scaled)))
      (goto-char next-change))))

```

```

(advice-add #'calendar :after #'sachal-calendar-heat-map-using-echo-text)
(advice-add #'calendar-redraw :after #'sachal-calendar-heat-map-using-echo-text)
(advice-add #'year-calendar :after #'sachal-calendar-heat-map-using-echo-text)

```

So now I don't need the advice around `calendar-generate-month`, just the code that sets up the faces, loads the values, and figures out the data.

nil

Now I can have it display the last year of data or so.

```

(defvar sachal-calendar-count-scaled)
;;;###autoload
(defun sachal-calendar-visualize (values)
  (setq sachal-calendar-count-scaled values)
  (let* ((date (calendar-current-date))
         (month (calendar-extract-month date))
         (year (calendar-extract-year date)))
    (year-calendar month (1- year))))

```

The code to load the data stays the same.

nil

Here's the code from lawlist's StackOverflow answer that [displays the Emacs calendar for a year](#):

nil

It might be fun to scroll by year:

```

;;;###autoload
(defun sachal-scroll-year-calendar-forward-year (&optional arg event)
  "Scroll the yearly calendar by year in a forward direction."
  (interactive (list (prefix-numeric-value current-prefix-arg)

```

```

                                last-nonmenu-event))
(unless arg (setq arg 1))
(save-selected-window
 (if (setq event (event-start event)) (select-window (posn-window event)))
 (unless (zerop arg)
  (setq displayed-year (+ (or arg 1) displayed-year))
  (year-calendar displayed-month displayed-year))
 (goto-char (point-min))
 (run-hooks 'calendar-move-hook)))

;;;###autoload
(defun sacha-scroll-year-calendar-backward-year (&optional arg event)
  "Scroll the yearly calendar by month in a backward direction."
  (interactive (list (prefix-numeric-value current-prefix-arg)
                     last-nonmenu-event))
  (sacha-scroll-year-calendar-forward-year (- (or arg 1)) event))
(eval-after-load "calendar" '(progn
  (define-key calendar-mode-map "{" 'sacha-scroll-year-calendar-backward-year)
  (define-key calendar-mode-map "}" 'sacha-scroll-year-calendar-forward-year)))

```

I used M-x gif-screencast to make the animated GIF. Yay Emacs!

14.3 Workrave

```

(defun sacha-workrave-file (expand-file-name ".\\Workrave\\historystats" (getenv "AppData")))

;;;###autoload
(defun sacha-workrave-transform-statistics (&optional file)
  (interactive (list sacha-workrave-file))
  (with-current-buffer (find-file-noselect file)
    ;; D day month-1 year hour min day month-1 year hour min
    (let ((result "Date\\tStart\\tEnd\\tClicks\\tKeystrokes\\n"))
      (goto-char (point-min))
      (while (re-search-forward "^D \\(.*\\)" nil t)
        (let ((dates (split-string (match-string 1))))
          (if (re-search-forward "^m \\(.*\\)" nil t)
              (let ((info (split-string (match-string 1))))
                (setq result
                  (concat result
                    (format "%d-%d-%s\\t%s:%02d\\t%s:%02d\\t%s\\t%s\\n"
                      (+ 1900 (string-to-number (elt dates 2))) ; year
                      (1+ (string-to-number (elt dates 1))) ; month
                      (elt dates 0) ; day
                      (elt dates 3) ; start hour
                      (string-to-number (elt dates 4)) ; start min
                      (elt dates 8) ; end hour
                      (string-to-number (elt dates 9)) ; end min
                      (elt info 5) ; clicks
                      (elt info 6) ; keystrokes
                    ))))))
          (if (interactive-p)
              (kill-new result)
              result))))

```

14.4 Blog

```

;;;###autoload
(defun sacha-strip-blog-share ()
  (interactive)
  (let (base)
    (save-excursion
      (goto-char (point-min))
      (while (re-search-forward
        "<div class=\"sharedaddy sd-sharing-enabled\">.*?<div ↵
        ↵ class=\"sharing-clear\"></div></div></div></div>" nil t)
        (replace-match ""))))

```

14.5 Time tracking, previous weekly review

```

(defun sacha-org-quantified-categories
  '(("Business"
    ("Earn" . "Business - Earn")
    ("E1" . "Business - Earn - Consulting - E1")
    ("Connect" . "Business - Connect")
    ("Build" . "Business - Build"))
    ("Discretionary"
    ("Social" . "Discretionary - Social")
    ("Productive" . "Discretionary - Productive")
    ("Sewing" . "Discretionary - Productive - Sewing")
    ("Writing" . "Discretionary - Productive - Writing")
    ("Emacs" . "Discretionary - Productive - Emacs")
    ("Play" . "Discretionary - Play"))
    ("Personal" ;("Biking" . "Personal - Bike")
    ("Routines" . "Personal - Routines"))
    ("Sleep" nil)
    ("Unpaid work"
    ("Commuting" . "Unpaid work - Subway")
    ("Cook" . "Unpaid work - Cook")
    ("Tidy" . "Unpaid work - Tidy up"))))
  "Categories for time summary.")

;;;###autoload
(defun sacha-org-summarize-time-use (&optional start end)
  (interactive (list (org-read-date) (org-read-date)))
  (let ((time-summary (quantified-summarize-time start end))
        (categories sacha-org-quantified-categories)
        result)
    (setq result
      (mapconcat
        (lambda (a)
          (if (assoc (car a) time-summary)
              (concat
                (format "- %s: %.1f hours" (car a) (/ (cdr (assoc (car a) time-summary)) ↵
                ↵ 3600.0))
              (if (cdr a)
                  (let ((detail
                        (delq nil
                          (mapcar (lambda (b)
                                    (if (assoc (cdr b) time-summary)
                                        (format "%s: %.1f"
                                          (car b)

```

```

                                (/ (cdr (assoc (cdr b) time-summary)) ←
← 3600.0))
                                nil))
                                (cdr a))))
    (if detail
      (concat " (" (mapconcat 'identity detail ", ") ")")
      "")
    (if (string-equal (car a) "Sleep")
      (format " - average of %.1f hours per day" (/ (cdr (assoc (car a) ←
← time-summary)) 3600.0 7.0))
      "")
    "\n"))
  categories "")
(if (called-interactively-p 'any)
  (insert result)
  result)))

```

14.5.1 List upcoming tasks so that I can see if I'm overloaded

```

;;;###autoload
(defun sacha-org-summarize-upcoming-week ()
  "Summarize upcoming tasks as a list."
  (interactive)
  (org-agenda nil "w")
  (let ((string (buffer-string))
        business relationships life)
    (with-temp-buffer
      (insert string)
      (goto-char (point-min))
      (while (re-search-forward sacha-weekly-review-line-regexp nil t)
        (cond
         ((string= (match-string 1) "routines") nil) ; skip routine tasks
         ((string= (match-string 1) "business")
          (add-to-list 'business (concat " - [ ] " (match-string 3))))
         ((string= (match-string 1) "people")
          (add-to-list 'relationships (concat " - [ ] " (match-string 3))))
         (t (add-to-list 'life (concat " - [ ] " (match-string 3))))))
      (setq string
            (concat
             "*Plans for next week*\n"
             "- Business\n"
             (mapconcat 'identity business "\n")
             "\n- Relationships\n"
             (mapconcat 'identity relationships "\n")
             "\n- Life\n"
             (mapconcat 'identity life "\n"))))
      (if (called-interactively-p 'any)
          (kill-new string)
          string)))

```

This uses Org Agenda's log mode to summarize the tasks that I checked off. I still need to match it up with the plans for the previous week to see which items I'd planned ahead, and which ones were new tasks. (Hmm, is it important to track those separately? I might just skip it.)

```

;;;###autoload
(defun sacha-org-summarize-previous-week ()

```

```

"Summarize previously-completed tasks as a list."
(interactive)
(save-window-excursion
  (org-agenda nil "w")
  (org-agenda-later -1)
  (org-agenda-log-mode 16)
  (let ((string (buffer-string))
        business relationships life)
    (with-temp-buffer
      (insert string)
      (goto-char (point-min))
      (while (re-search-forward sachu-weekly-review-line-regexp nil t)
        (cond
          ((string= (match-string 1) "routines") nil) ; skip routine tasks
          ((string= (match-string 1) "business")
           (add-to-list 'business (concat " - " (match-string 2))))
          ((string= (match-string 1) "people")
           (add-to-list 'relationships (concat " - " (match-string 2))))
          (t (add-to-list 'life (concat " - " (match-string 2)))))))
    (setq string
      (concat
        "*Accomplished this week*\n\n"
        "- Business\n"
        (mapconcat 'identity business "\n")
        "\n- Relationships\n"
        (mapconcat 'identity relationships "\n")
        "\n- Life\n"
        (mapconcat 'identity life "\n"))))
    (if (called-interactively-p 'any)
        (kill-new string)
        string))))

```

14.5.2 Compare time use

```

;; ##autoload
(defun sachu-quantified-compare (start1 end1 start2 end2 &optional categories label1 label2)
  "Return a table comparing the times for START1 - END1 and START2 - END2."
  (interactive (list
    (org-read-date "Start of ↵
↵ period 1")
    (org-read-date "End of period 1")
    (org-read-date "Start of ↵
↵ period 2")
    (org-read-date "End of period 2")
    '("Business" "Discretionary - ↵
↵ Play" "Unpaid work"
↵ Family" "Discretionary - Social" "Sleep"
↵ Productive" "Personal"))))
  (let* ((start2 (org-read-date nil nil (or start2 "-sat")))
        (end2 (org-read-date nil nil (or end2 "+1")))
        (start1 (org-read-date nil nil (or start1 "-4sat")))
        (end1 (org-read-date nil nil (or end1 "-sat")))
        (time2 (quantified-summarize-time start2 end2))
        (time1 (quantified-summarize-time start1 end1))
        (label1 (or label1 "Period 1 %")))

```



```

                                (scroll-up 1))))
(use-package org
  :config
  (when sacha-phone-p
    (add-to-list 'org-file-apps '("\\.png\\") . default))
    (add-to-list 'org-file-apps '("\\.jpg\\") . default))
    (add-to-list 'org-file-apps '("\\.jpeg\\") . default)))
)

```

```

;;###autoload
(defun sacha-format-intent (intent &optional params)
  "Return a command string for sending INTENT with PARAMS.
  PARAMS is an alist of (\"key\" . \"value\") pairs."
  (format "am broadcast --user 0 -a %s %s"
    intent
    (mapconcat
      (lambda (o)
        (format
          "-e %s %s"
          (shell-quote-argument (car o))
          (shell-quote-argument (cdr o))))
      params
      " ")))

;;###autoload
(defun sacha-send-intent (intent &optional params)
  "Send broadcast INTENT to my phone.
  PARAMS is a plist of :key value pairs."
  (let ((command (sacha-format-intent intent params)))
    (if sacha-phone-p
      (shell-command command)
      (shell-command (format "ssh phone %s" (shell-quote-argument command))))))

```

15.1 Syncthing

From https://www.reddit.com/r/emacs/comments/bqqqra/quickly_find_syncthing_conflicts_and_resolve_them/ In termux, you also need to `pkg install diffutils`.

```

(setq ediff-toggle-skip-similar t
  ediff-diff-options "-w"
  ediff-window-setup-function 'ediff-setup-windows-plain
  ediff-split-window-function 'split-window-horizontally)

```

```

;;###autoload
(defun sacha-resolve-orgzly-syncthing ()
  (interactive)
  (ibizaman/syncthing-resolve-conflicts "~/sync/orgzly"))

;;###autoload
(defun sacha-resolve-orgzly-ipad-syncthing ()
  (interactive)
  (ibizaman/syncthing-resolve-conflicts "~/sync/orgzly-ipad"))

;; ↩
↪ https://www.reddit.com/r/emacs/comments/bqqqra/quickly\_find\_syncthing\_conflicts\_and\_resolve\_them/

```

```

;;;###autoload
(defun ibizaman/syncthing-resolve-conflicts (directory)
  "Resolve all conflicts under given DIRECTORY."
  (interactive "D")
  (let* ((all (ibizaman/syncthing--get-sync-conflicts directory))
         (chosen (ibizaman/syncthing--pick-a-conflict all)))
    (ibizaman/syncthing-resolve-conflict chosen)))

;;;###autoload
(defun ibizaman/syncthing-show-conflicts-dired (directory)
  "Open dired buffer at DIRECTORY showing all syncthing conflicts."
  (interactive "D")
  (find-name-dired directory "*.sync-conflict-*org"))

;;;###autoload
(defun ibizaman/syncthing-resolve-conflict-dired (&optional arg)
  "Resolve conflict of first marked file in dired or close to point with ARG."
  (interactive "P")
  (let ((chosen (car (dired-get-marked-files nil arg))))
    (ibizaman/syncthing-resolve-conflict chosen)))

;;;###autoload
(defun ibizaman/syncthing-resolve-conflict (conflict)
  "Resolve CONFLICT file using ediff."
  (let* ((normal (ibizaman/syncthing--get-normal-filename conflict))
         (ibizaman/ediff-files
          (list conflict normal)
          `(lambda ()
              (when (y-or-n-p "Delete conflict file? ")
                (kill-buffer (get-file-buffer ,conflict))
                (delete-file ,conflict))))))
    ))

;;;###autoload
(defun ibizaman/syncthing--get-sync-conflicts (directory)
  "Return a list of all sync conflict files in a DIRECTORY."
  (seq-filter (lambda (o) (not (string-match "\\..stversions" o))) (directory-files-recursively
    ↪ directory "\\..sync-conflict-*org$")))

(defvar ibizaman/syncthing--conflict-history nil
  "Completion conflict history")

;;;###autoload
(defun ibizaman/syncthing--pick-a-conflict (conflicts)
  "Let user choose the next conflict from CONFLICTS to investigate."
  (completing-read "Choose the conflict to investigate: " conflicts
    nil t nil ibizaman/syncthing--conflict-history))

;;;###autoload
(defun ibizaman/syncthing--get-normal-filename (conflict)
  "Get non-conflict filename matching the given CONFLICT."
  (replace-regexp-in-string "\\..sync-conflict-*\\(\\..*\\)$" "\\1" conflict))

```

```

;;;###autoload
(defun ibizaman/ediff-files (&optional files quit-hook)
  (interactive)
  (lexical-let ((files (or files (dired-get-marked-files)))
                (quit-hook quit-hook)
                (wnd (current-window-configuration))))
    (if (<= (length files) 2)
        (let ((file1 (car files))
              (file2 (if (cdr files)
                        (cadr files)
                        (read-file-name
                         "file: "
                         (dired-dwim-target-directory))))))
          (if (file-newer-than-file-p file1 file2)
              (ediff-files file2 file1)
              (ediff-files file1 file2))
          (add-hook 'ediff-after-quit-hook-internal
                    (lambda ()
                      (setq ediff-after-quit-hook-internal nil)
                      (when quit-hook (funcall quit-hook))
                      (set-window-configuration wnd))))
        (error "no more than 2 files should be marked"))))

```

16 Clipboard

```

(use-package clipmon
  :disabled t
  :init (progn (setq clipmon-action 'kill-new clipmon-timeout nil clipmon-sound nil
                    ↪ clipmon-cursor-color nil clipmon-suffix nil) (clipmon-mode)))

```

On my phone:

```

(use-package xclip :if sacha-phone-p) ; Turn on with xclip-mode

```

17 Mail and news

17.1 TOBLOG Send mail asynchronously

Based on [smtpmail-async](#), but with the list of variables tweaked because `mail-extr-all-top-level-domains` was an `#<obarray n=344>` that couldn't get passed.

```

;;;###autoload
(defun sacha-async-smtpmail-send-it ()
  (let ((to (message-field-value "To"))
        (buf-content (buffer-substring-no-properties
                      (point-min) (point-max))))
    (message "Delivering message to %s ..." to)
    (async-start
     `(lambda ()
         (require 'smtpmail)
         (with-temp-buffer
          (insert ,buf-content)
          (set-buffer-multibyte nil)

```

```
;; Pass in the variable environment for smtpmail
,(async-inject-variables
  "\\(smtpmail\\|async-smtpmail\\|user-mail\\)-\\|auth-sources\\|epg\\|nsm"
  nil ←
  ↪ "\\(mail-header-format-function\\|smtpmail-address-buffer\\|mail-mode-abbrev-table\\)"
    (smtpmail-send-it)))
  `(lambda (&optional _ignore)
    (message "Delivering message to %s ... done" ,to))))
```

```
(setq send-mail-function 'sacha-async-smtpmail-send-it
  message-send-mail-function 'sacha-async-smtpmail-send-it)
```

17.2 Notmuch

I use Notmuch with [Lier](#) to fetch my mail from Gmail.

```
(setq notmuch-message-headers '("Subject" "To" "Cc" "Date" "Reply-To"))
(use-package notmuch
  :if sacha-laptop-p
  :config (setq-default notmuch-search-oldest-first nil)
  (setq notmuch-fcc-dirs nil)
  (setq notmuch-archive-tags '("-inbox" "-flagged" "-unread" "-new")))
(use-package ol-notmuch
  :if sacha-laptop-p)
```

```
;; ###autoload
(defun sacha-notmuch-flagged ()
  (interactive)
  (notmuch-search "tag:flagged and not tag:trash"))
;; ###autoload
(defun sacha-notmuch-inbox ()
  (interactive)
  (notmuch-search "tag:inbox and not tag:trash"))
;; ###autoload
(defun sacha-notmuch-important-inbox ()
  (interactive)
  (notmuch-search "tag:primary and tag:inbox and not tag:trash"))
;; ###autoload
(defun sacha-notmuch-search-this-author ()
  (interactive)
  (notmuch-search (format "from: \"%s\""
    (plist-get (get-text-property (point) 'notmuch-search-result) ←
    ↪ :authors))))
```

17.2.1 Act on current message with Embark

embark

```
;; ###autoload
(defun sacha-embark-mail-finder ()
  "Identify when we're in a notmuch message."
  (cond ((derived-mode-p 'notmuch-show-mode)
    ↪ (mail . ,(plist-get (plist-get ←
    ↪ (notmuch-show-get-message-properties) :headers) :From))))
```

```
(with-eval-after-load 'embark
  (add-to-list 'embark-target-finders 'sacha-embark-mail-finder)
)
```

17.2.2 Add comment to blog post

```
;;;###autoload
(defun sacha-message-add-blog-comment (url)
  (interactive (list (sacha-complete-blog-post-url)))
  (save-excursion
    (goto-char (point-min))
    (let* ((author (when (re-search-forward "Name you want.+?: \\(\\.+\\)" nil t)
                        (match-string 1)))
           (message (when (re-search-forward "Message: *\\n?" nil t)
                        (match-string 1)
                        (read-string "Message: "
                                     (buffer-substring (match-end 0)
                                                         (point-max))))))
           (comment (if (re-search-forward "Can I share your comment" nil t)
                        (match-string 1)
                        (read-string "Comment: "
                                     (buffer-substring (match-end 0)
                                                         (point-max))))))
           (date (format-time-string "%FT%T%z" (date-to-time (time-current-time))))
           (new-comment `((author . ,author)
                          (date . ,date)
                          (message . ,(format "<div class=\"email-body\">%s</div>"
                                              (org-export-string-as message 'html t))))))
    (find-file (sacha-11ty-add-blog-comment new-comment url))))
```

17.3 Gnus

I still use Gnus so that I can use [Gmane](#) to read mailing lists.

I used to have my config in in `~/.gnus`, but people might find it handy, so I've added it to my public [Emacs configuration](#).

```
(setq mml-secure-openpgp-encrypt-to-self t)
(setq gnus-select-method '(nnnil ""))
(setq gnus-secondary-select-methods
      '((nnntp "news.gmane.io")
        ;; (nnmaildir "mail"
        ;;       (directory "~/Maildir/account.gmail"))
```

```

;; (directory-files nnheader-directory-files-safe)
;; (get-new-mail nil))
;; (nnimap "imap.googlemail.com"
;; (nnimap-address "imap.googlemail.com")
;; (nnimap-server-port 993)
;; (nnimap-stream ssl)
;; (nnimap-authenticator login))
(nnimap "localhost"
  (nnimap-address "localhost")
  (nnimap-stream network)
  (nnimap-user "sacha")
  (nnimap-authenticator login)
  (nnimap-authinfo-file "~/authinfo.gpg"))
))

(setq smtpmail-smtp-server "smtp.googlemail.com"
smtpmail-smtp-service 587
smtpmail-auth-credentials "~/authinfo.gpg"
send-mail-function 'smtpmail-send-it
message-send-mail-function 'smtpmail-send-it
gnus-check-new-newsgroups nil
gnus-activate-level 2
gnus-ignored-newsgroups "^to\\\\.\\\\|^\\\\[0-9. ]+\\\\( \\\\|\\\\$\\\\)\\\\|^\\\\[\"']\\\\[\"'#()]" )

```

I now use Dovecot with OfflineIMAP for local IMAP access to my mail and synchronization with Gmail, but you can see the commented-out information for Gmail in case you prefer that. I have two-factor authentication enabled for Gmail, so I set up an app-specific password for Gnus. I have GPG set up for encryption, and an `~/.authinfo.gpg` file set up with something like:

```
machine imap.gmail.com login sachas@sachachua.com password mysecretapppassword
machine imap.gmail.com login sachas@sachachua.com password mysecretapppassword port 993
machine smtp.gmail.com login sachas@sachachua.com password mysecretapppassword port 587
machine localhost login sachas password mysecretlocalpassword port 993
machine localhost login sachas password mysecretlocalpassword port 143
```

If you don't have GPG set up and you don't mind saving your passwords in the clear, you can set up an `~/.authinfo` file instead.

```
(use-package gnus
  :defer t
  :commands gnus

:config
(require 'mm-decode)
(setq mm-discouraged-alternatives
  '("text/html" "text/richtext")
  mm-automatic-display
  (-difference mm-automatic-display '("text/html" "text/enriched" "text/richtext"))))
```

Hide quoted text.

```
(setq gnus-treat-hide-citation t)
```

Get smarter about filtering depending on what I read or mark. I use ! (tick) for marking threads as something that interests me.

```
(setq gnus-use-adaptive-scoring t)
(setq gnus-default-adaptive-score-alist
      '((gnus-unread-mark)
```

```
(gnus-ticked-mark (subject 10))
(gnus-killed-mark (subject -5))
(gnus-catchup-mark (subject -1)))
```

17.3.1 Windows

I don't use Windows much any more, but this is here in case it's helpful for people who still do.

Sending e-mail on Windows was a bit of a pain. Fortunately, I eventually found something that works. I've configured [emailrelay](#) to accept the mail and forward it to Gmail. The server starts with this batch file:

```
start "emailrelay" "C:\Program Files (x86)\emailrelay\emailrelay.exe" --as-proxy smtp.gmail.com:2
```

Sending queued mail works with this batch file:

```
"c:\Program Files (x86)\emailrelay\emailrelay.exe" --as-client smtp.gmail.com:587 --client-auth c
```

I should probably get around to using `--as-proxy` properly, since it still seems to hold mail until I explicitly send it.

On Linux, it's simply a matter of setting up a mail server such as [Postfix](#).

Hide HTML mail. I need to fiddle with this some more, since Gnus still tries to display them. Sometimes my Gnus crashes when it tries to display HTML mail.

17.4 Approve or discard Mailman messages

The mailing lists for [emacsconf-org](#), [emacsconf-org-private](#), [emacsconf-submit](#), and [emacs-tangents](#) are all handled by the Mailman program. We usually set mailing lists to moderated so that

```

;;###autoload
(defun sach-mailman-approve ()
  "Approve this mailing list message."
  (interactive)
  (goto-char (point-min))
  (when (re-search-forward "From: \\(\\(\\(\\.+\\)-request@.*?\\)\\)\\nSubject: \\(confirm ↵
    ↵ [0-9a-f]+\\)" nil t)
    (let* ((id (match-string 2)))
      (compose-mail (match-string 1) (match-string 3)
                    `(("Approved" ↵
    ↵
    ↵
    (concat "pass " (match-string 2))))))
      (message-send-and-exit))))

;;###autoload
(defun sach-mailman-discard ()
  "Discard the current message."
  (interactive)
  (goto-char (point-min))
  (when (re-search-forward "From: \\(\\(\\(\\.+\\)-request@.*?\\)\\)\\nSubject: \\(confirm ↵
    ↵ [0-9a-f]+\\)" nil t)
    (compose-mail (match-string 1) (match-string 3))
    (message-send-and-exit)))

```

```

;;;###autoload
(defun sachamailman-web (&optional list-id)
  "Open the web admin interface."
  (interactive
   (list
    (if (and (derived-mode-p 'notmuch-show-mode)
              (re-search-forward "\<https://.+?/mailman/admindb/\<(.+\\)\>" nil t))
        (match-string 2)
        (completing-read "List: " '("emacsconf-org" "emacsconf-org-private"
                                     "emacs-tangents"
                                     "emacsconf-submit" "emacsconf-discuss"
                                     "info-gnu-emacs")))))
    (goto-char (point-min))
    (browse-url (concat "https://lists.gnu.org/mailman/admindb/" list-id "?adminpw="
                        (url-hexify-string (string-trim (shell-command-to-string
                                                         (concat "pass " list-id)))))))

```

18 Collaboration

```

(use-package crdt
  :vc (:url "https://github.com/zaeph/crdt.el")
  :commands (crdt-share-buffer crdt-connect)
  :load-path "~/vendor/crdt.el"
  :if sachalaptop-p)

```

18.1 Bike Brigade

18.1.1 Transforming HTML clipboard contents with Emacs to smooth out Mailchimp annoyances: dates, images, comments, colours emacs

[2025-09-17 Wed]: Minor tweaks to get it to work better with [Getting a Google Docs draft ready for Mailchimp via Emacs](#) (p. 628).

I've recently started handling the [Bike Brigade](#) newsletter, so now I'm itching to solve the little bits of friction that get in my way when I work with the rich-text [Mailchimp](#) block editor.

I'm not quite ready to generate everything with [Org Mode](#). Sometimes other people go in and edit the newsletter through the web interface, so I shouldn't just dump a bunch of HTML in. (We don't have the more expensive plan that would allow me to make editable templates.) I draft the newsletter as a Slack canvas so more people can weigh in with their suggestions:

And then I redo it in Mailchimp:

My process is roughly:

1. Duplicate blocks.
2. Copy the text for each item and paste it in. Adjust formatting.



newsletter block history

- communications folder
- code for delivery milestones

Bike Brigade



Join a market ride

Dufferin was huge this week! Join us at other farmers market rides:

- Dufferin Grove on Thursdays at 7pm
- Sorauren Park on Mondays starting June 30 at 7pm
- Trinity Bellwoods on Tuesdays starting July 1 at 7pm

[Check dispatch]



Canada Day schedule changes

- FYFB will be closed on July 1 (Tue) for Canada Day. Deliveries will move to June 30 (Mon).

on break this week.

Figure 20: Screenshot of Slack canvas

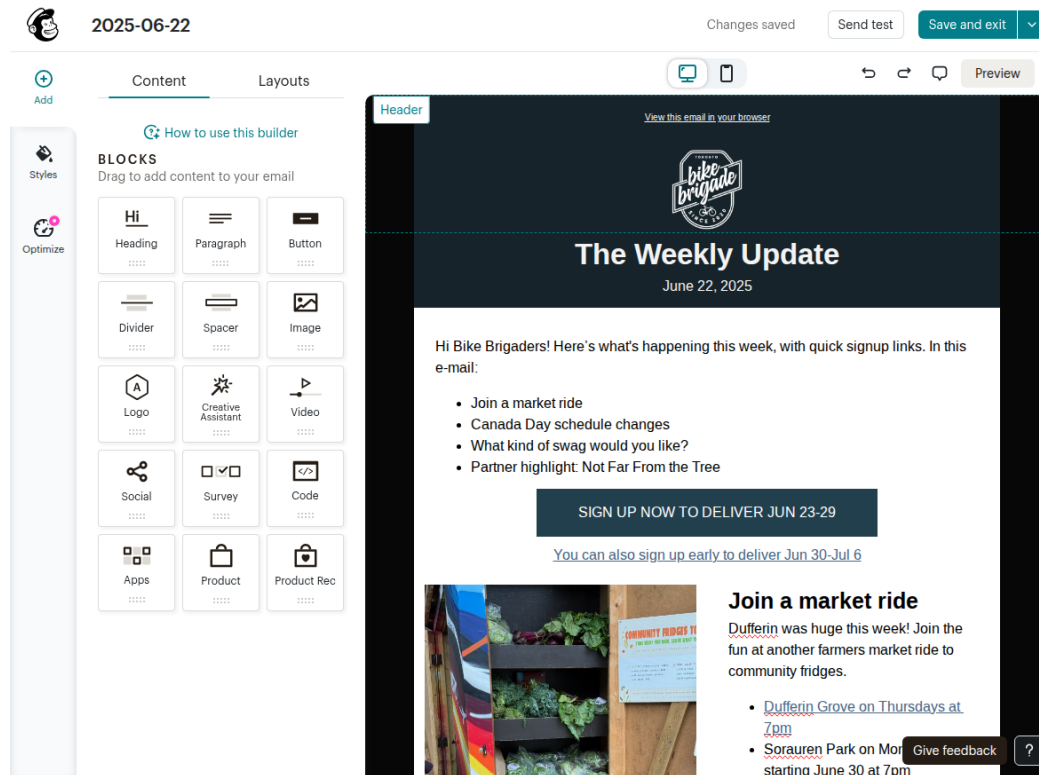


Figure 21: Screenshot of Mailchimp design

3. Update the dates and links. Flip back and forth between the dispatch webpage and Mailchimp, getting the links and the dates just right.
4. Download images one by one.
5. Replace the images by uploading the saved images. Hunt through lots of files named image (3).png, image (4).png, and so on. Update their attributes and links.
6. Change text and link colours as needed by manually selecting the text, clicking on the colour button in the toolbar, and selecting the correct colour.
7. Change the text on each button. Switch to Slack, copy the link, switch back to Mailchimp, and update the link.

I think I can get Emacs to make things easier.

Automating buttons	615
Transforming HTML	616
Saving images	617
Cleaning up	621
Removing sections	621
Removing unneeded styles	622
Formatting calls to action	623
Changing link colours	624
Just the headings	626

Wrapping it up	626
Next steps	627

Automating buttons The newsletter includes a button to make it easier to volunteer for deliveries. In case people want to plan ahead, I also include a link to the following week’s signups.

Dates are fiddly and error-prone, so I want to automate them. I can use a Mailchimp code block to paste in some HTML directly, since I don’t think other people will need to edit this button. Here I take advantage of `org-read-date`’s clever date parsing so that I can specify dates like `+2Sun` to mean two Sundays from now. That way, I don’t have to do any date calculations myself.

This code generates something like this:

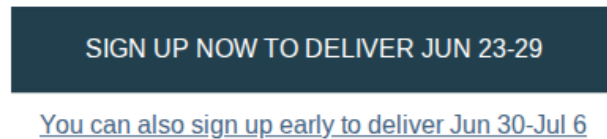


Figure 22: Screenshot of buttons

nil

Here’s the code. It calculates the dates, formats the HTML code. I use `format-time-string` to format just the month part of the dates and compare them to tell if I can skip the month part of the end date. After the HTML is formatted, the code uses `xdotool` (a Linux command-line tool) to switch to Google Chrome so that I can paste it in.

```
;;###autoload
(defun sacha-brigade-copy-signup-block (date)
  (interactive (list (if current-prefix-arg (org-read-date nil t nil "Date: ")
    ↵
    (org-read-date nil t "+Sun"))))
  (when (stringp date) (setq date (date-to-time date)))
  (let* ((newsletter-date (format-time-string "%Y-%m-%d" date))
    (current-week (org-read-date nil t "++Mon" nil date))
    (current-week-end (org-read-date nil t "++Sun" nil date))
    (next-week (org-read-date nil t "+2Mon" nil date))
    (next-week-end (org-read-date nil t "+3Sun" nil date))
    result)
    (setq result (format
      "<table class=\"sign-up\" style=\"background-color: #223f4d; ↵
    ↵ text-align: center; margin: auto; margin-top: 24px; margin-bottom: ↵
    ↵ 12px;\><tbody><tr><td><a ↵
    ↵ href=\"https://dispatch.bikebrigade.ca/campaigns/signup?current_week=%s\" ↵
    ↵ target=\"_blank\" class=\"sign-up mceButtonLink\" ↵
    ↵ style=\"background-color:#223f4d;border-radius:0;border:2px solid ↵
    ↵ #223f4d;color:#ffffff;display:block;font-family:'Helvetica Neue', Helvetica, Arial, ↵
    ↵ Verdana, sans-serif;font-size:16px;font-weight:normal;font-style:normal;padding:16px ↵
    ↵ 28px;text-decoration:none;text-align:center;direction:ltr;letter-spacing:0px\" ↵
    ↵ rel=\"noreferrer\">SIGN UP NOW TO DELIVER %s-%s</a></td></tr></table>
    <p style=\"text-align: center; font-family: 'Helvetica Neue', Helvetica, Arial, Verdana\"><a ↵
    ↵ href=\"https://dispatch.bikebrigade.ca/campaigns/signup?current_week=%s\" ↵
    ↵ style=\"color: #476584; margin-top: 12px; margin-bottom: 12px;\>You ↵
    ↵ can also sign up early to deliver %s-%s</a></p>\"
      (format-time-string "%Y-%m-%d" current-week)
      (upcase (format-time-string "%b %-e" current-week))
    ))))
```



```

    (or functions ←
      (buffer-string))))
      (dom))

(defvar sachatransformhtmlclipboardfunctions nil "List of functions to call with the ←
  ← clipboard contents.
Each function should take a DOM node and return the resulting DOM node.")
;; Rich text can sometimes be finicky to paste, so maybe I'll default to working with plain text
;; if there's a code view I can use to paste in the HTML.
(defvar sachatransformhtmlclipboardrichtext nil
  "Non-nil means copy as rich text instead of plain HTML.")
;;;###autoload
(defun sachatransformhtmlclipboard (&optional activate-app-afterwards functions text ←
  ←
  ← as-rich-text)
  "Parse clipboard contents and transform it.
This calls FUNCTIONS, defaulting to `sachatransformhtmlclipboardfunctions'.
If ACTIVATE-APP-AFTERWARDS is non-nil, use xdotool to try to activate that app's window."
  (when (region-active-p) (setq text (buffer-substring (region-beginning) (region-end))))
  (unless text
    (setq text (shell-command-to-string "unbuffer -p xclip -o -selection clipboard ←
    ← -t text/html 2>& /dev/null")))
  (when (string= text "") (error "Clipboard does not contain HTML.))
  (with-temp-buffer
    (insert (sachatransformhtml functions text))
    (if (or as-rich-text sachatransformhtmlclipboardrichtext)
      (shell-command-on-region
        (point-min) (point-max)
        "xclip -i -selection clipboard -t text/html -filter ←
        ← 2>& /dev/null")
      (kill-new (buffer-substring-no-properties (point-min) (point-max)))))
  (when activate-app-afterwards
    (call-process "xdotool" nil nil nil "search" "--onlyvisible" "--all" ←
    ← activate-app-afterwards "windowactivate" "windowfocus"))

```

Saving images Images from Slack don't transfer cleanly to Mailchimp. I can download images from Slack one at a time, but Slack saves them with generic filenames like `image (2).png`. Each main newsletter item has one image, so I'd like to automatically save the image using the item title.

When I copy HTML from the Slack canvas, images are included as data URIs. The markup looks like this: `<img src='data:image/png;base64,iVBORw0KGgo ...` With the way I do the draft in Slack, images are always followed by the item title as an `h2` heading. If there isn't a heading, the image just doesn't get saved. If there's no image in a section, the code clears the variable, so that's fine too. I can parse and save the images like this:

```

;; Hmm, now I need to modify it to handle emojis in the text.
;; Emojis have :text: in the alt. I need to upload them.

;;;###autoload
(defun sachatransformhtmlsaveimages (dom dir &optional file-prefix transform-fn)
  "Returns a list of (section list).
list is a list of alists with the following keys:
- type (main, extra, emoji)
- alt
- filename"

```

```

    (let (last-image last-image-filename last-image-alt results last-image-node)
      (dom-search
dom
        (lambda (node)
          (pcase (dom-tag node)
            ('img
              (let* ((data (dom-attr node 'src))
                    (jpg (concat (file-name-sans-extension data)
                                  ".jpg"))
                    (when (file-exists-p jpg)
                      (dom-set-attribute node 'src jpg)
                      (setq data jpg))
                    (when (or last-image last-image-filename)
                      (cond
                        ;; this image is after another image, so the previous image must be an extra one
                        (last-image
                          (setq last-image-filename
                                (expand-file-name
                                  (format "%s%s-extra.%s"
                                         (or file-prefix "")
                                         (if transform-fn
                                             (funcall transform-fn (caar results))
                                             (caar results))
                                         (car last-image)
                                         dir))
                                (with-temp-file
                                  last-image-filename
                                  (set-buffer-file-coding-system 'binary)
                                  (insert
                                    (base64-decode-string (cdr last-image))))
                                ;; this image is after another image,
                                so the previous image must be an extra one
                                (last-image-filename
                                  (let ((new-filename
                                         (expand-file-name
                                           (format "%s%s-extra.%s"
                                                  (or file-prefix "")
                                                  (if transform-fn
                                                      (funcall transform-fn (caar
results))
                                                      (caar results))
                                                  ".jpg")
                                           dir))))

```

```

↪ "convert" nil nil nil last-image-filename                                (call-process ↪
                                                                                               ↪
↪                               new-filename)                                (setq ↪
↪ last-image-filename new-filename))))
    (push
      (list
        (cons 'type 'extra)
        (cons 'filename last-image-filename)
        (cons 'alt last-image-alt))
      (cdr (car results)))
    (dom-remove-node dom node)
    (setq last-image-node nil)
    (setq last-image nil last-image-filename nil))
    (cond
      ((string-match "^:.*?:" (or (dom-attr node 'alt) ""))
       (setq last-image-node nil)
       (push
         (list
           (cons 'type 'emoji)
           (cons 'filename (dom-attr node 'src))
           (cons 'alt (dom-attr node 'alt)))
         (cdar results)))
       ((string-match "^images/" data)
        (setq last-image nil
              ↪
↪ last-image-filename data
                                                                                               last-image-alt ↪
↪ (dom-attr node 'alt)
    last-image-node node))
       ((string-match "^data:image/" data)
        (with-temp-buffer
          (insert data)
          (goto-char (point-min))
          (when (looking-at ↪
↪ "data:image/\\([^\n;]+?\\);base64,")
                                                                                               (setq last-image (cons ↪
↪ (match-string 1)
                                                                                               ↪
↪                               ↪
↪ (buffer-substring (match-end 0) (point-max)))
                                                                                               ↪
↪ last-image-filename nil
                                                                                               ↪
↪ last-image-alt (dom-attr node 'alt)
    last-image-node node))))))
    ('h2
      (when (not (string= (string-trim (dom-texts node)) ""))
        (cond
          (last-image
            (setq last-image-filename
                  (expand-file-name
                    (format ↪
↪ "%s%s.%s"
                                                                                               ↪
↪                               (or file-prefix ""))
                                                                                               ↪
↪                               (if transform-fn

```

```

↪                                     (funcall transform-fn (dom-texts node)) ↪
↪                                     (dom-texts node)) ↪
↪                                     (car last-image)) ↪
↪                                     dir))
↪                                     (with-temp-file last-image-filename
↪                                     (set-buffer-file-coding-system ↪
↪                                     'binary)
↪                                     (insert (base64-decode-string ↪
↪                                     (cdr last-image))))))
↪                                     (last-image-filename
↪                                     (let ((new-filename
↪                                     ↪
↪                                     (expand-file-name
↪                                     ↪
↪                                     (format "%s%s.%s"
↪                                     ↪
↪                                     (or file-prefix "")
↪                                     ↪
↪                                     (if transform-fn
↪                                     ↪
↪                                     (funcall transform-fn (dom-texts node))
↪                                     ↪
↪                                     (dom-texts node))
↪                                     ↪
↪                                     (file-name-extension last-image-filename)
↪                                     dir)))
↪                                     (copy-file last-image-filename new-filename t)
↪                                     (setq last-image-filename ↪
↪                                     new-filename))))
↪                                     (push (cons (string-trim (dom-texts node))
↪                                     ↪
↪                                     `(((filename . ,last-image-filename)
↪                                     (alt . ,last-image-alt)
↪                                     (type . main))))
↪                                     results)
↪                                     (when last-image-node
↪                                     (dom-remove-node dom last-image-node))
↪                                     (setq last-image nil
↪                                     last-image-node nil
↪                                     last-image-filename ↪
↪                                     nil))))))
↪                                     (nreverse results)))

```

I wrapped this in a small function for newsletter-specific processing:

```

;;;###autoload
(defun sachatransformhtmlslugify (s)
  (downcase
   (replace-regexp-in-string
    "^-+\\|-$" ""
    (replace-regexp-in-string
     "[^A-Za-z0-9]+" "-"
     (string-trim s)))))
(defvar sachabrigadenewsletterimagesdirectory "~/proj/bike-brigade/newsletter/images")
;;;###autoload

```



```
(defun sachabrigade-newsletter-heading-to-image-file-name (heading)
  (replace-regexp-in-string
    "[^a-z0-9]" " "
    (replace-regexp-in-string
      "+"
      "-"
      (string-trim (downcase heading)))))
;;###autoload
(defun sachabrigade-save-newsletter-images (dom)
  (sachatransform-html-save-images
    dom
    sachabrigade-newsletter-images-directory
    (concat (substring (org-read-date nil nil "+Sun") 0 10)
            "-news-")
    #'sachatransform-html-slugify))
```

For easier testing, I used `xclip -o -selection clipboard -t text/html > ~/Downloads/test.html` to save the clipboard. To run the code with the saved clipboard, I can call it like this:

```
(sachabrigade-save-newsletter-images
  (with-temp-buffer (insert-file-contents "~/Downloads/test.html") (libxml-parse-html-region))
)
```

Cleaning up Now that I've saved the images, I can remove them:

```
;;###autoload
(defun sachatransform-html-remove-images (dom)
  (dolist (img (dom-by-tag dom 'img))
    (dom-remove-node dom img))
  dom)
```

I can also remove the italics that I use for comments.

```
;;###autoload
(defun sachatransform-html-remove-italics (dom)
  (dolist (node (dom-by-tag dom 'i))
    (dom-remove-node dom node))
  dom)
```

Here's how I can test it:

```
(sachatransform-html-clipboard
  nil
  '(sachatransform-html-remove-images
    sachatransform-html-remove-italics)
  (with-temp-buffer (insert-file-contents "~/Downloads/test.html") (buffer-string)))
```

Removing sections I put longer comments and instructions under "Meta" headings, which I can automatically remove.

```
(defvar sachabrigade-section nil)
;;###autoload
(defun sachabrigade-remove-meta-recursively (node &optional recursing)
  "Remove <h1>Meta</h1> headings in NODE and the elements that follow them."
```



```

                                (seq-keep (lambda (class-name)
↪                                     (let ((prop (gethash (concat "." class-name) rules)))
↪                                     (when (and prop
↪                                     (string-match
↪ "font-weight:700\\|font-style:italic" prop))
↪                                     prop)))
↪
↪ (split-string (dom-attr node 'class))
↪                                     ";"))))
                                (unless (or (null styles) (string= styles ""))
                                (dom-set-attribute node 'style styles)
                                (dom-remove-attribute node 'class)
                                node))))
;;###autoload
(defun sacha-brigade-simplify-html (dom)
  (let ((css-rules (sacha-html-extract-css-rules dom)))
    (dolist (tag '(li b ul span p a h2 div))
      (dolist (node (dom-by-tag dom tag))
        (or (sacha-brigade-convert-span-style node css-rules)
            (dolist (attr '(style class id))
              (when (dom-attr node attr)
                (dom-remove-attribute node
↪ attr))))))
    ;; remove comments
    (mapc (lambda (node) (dom-remove-node (dom-parent dom node) node))
          (dom-by-tag dom 'sup))
    ;; remove blank paragraphs
    (dom-search
     dom
     (lambda (node)
       (when (eq (dom-tag node) 'p)
         (when (and (string= (string-trim (dom-texts node)) "")
                     (not (dom-by-tag node
↪ 'img)))
           (dom-remove-node (dom-parent dom node) node))))
    ;; fix links
    (dolist (node (dom-by-tag dom 'a))
      (when (string-match "https://www\\.google\\.com\\/url" (dom-attr node
↪ 'href))
        (let ((args (url-parse-query-string
↪ (cdr
↪ (url-path-and-query (url-generic-parse-url (dom-attr node 'href))))))
          (dom-set-attribute node 'href (car (assoc-default "q"
↪ args 'string=))))))
      dom))

```

Formatting calls to action Mailchimp recommends using buttons for calls to action so that they're larger and easier to click than links. In my Slack canvas draft, I use [link text] to indicate those calls to action. Wouldn't it be nice if my code automatically transformed those into centered buttons?

```
;;###autoload
```

```

(defun sacha-brigade-format-buttons (dom)
  (dolist (node (dom-by-tag dom 'a))
    (let ((text (dom-texts node)))
      (if (string-match "\\[ *\\(\\.+?\\) *\\]" text)
          ;; button, wrap in a table
          (with-temp-buffer
            (insert
              (format "<table><tbody><tr><td <
→ style=\"padding: 12px 0 12px 0\"><div style=\"margin-top: 12px\"><table <
→ align=\"center\" border=\"0\" cellpadding=\"0\" cellspacing=\"0\" <
→ role=\"presentation\" class=\"mceButtonContainer\" style=\"padding-top: 24px; margin: <
→ auto; margin-top: 12px; text-align: center\"><tbody><tr <
→ class=\"mceStandardButton\"><td <
→ style=\"background-color:#000000;border-radius:0;margin-top:12px;text-align:center\" <
→ valign=\"top\" class=\"mceButton\"><a href=\"%s\" target=\"_blank\" <
→ class=\"mceButtonLink\" style=\"background-color:#000000;border-radius:0;border:2px <
→ solid #000000;color:#ffffff;display:block;font-family:'Helvetica Neue', Helvetica, <
→ Arial, Verdana, <
→ sans-serif;font-size:16px;font-weight:normal;font-style:normal;padding:16px <
→ 28px;text-decoration:none;text-align:center;direction:ltr;letter-spacing:0px\" <
→ rel=\"noreferrer\">%s</a></td></tr></tbody></table></td></tr></table>"
              (dom-attr <
→ node 'href)
              (match-string <
→ 1 text)))
          (let ((parent-paragraph (sacha-dom-closest dom <
→ node 'p)))
            (dom-add-child-before
              (dom-parent dom parent-paragraph)
              (car (dom-by-tag <
→ (libxml-parse-html-region (point-min) (point-max)) 'div))
              parent-paragraph)
            (dom-remove-node dom <
→ parent-paragraph))))))
    dom)

```

Now I can test those functions in combination:

```

(sacha-transform-html-clipboard
  nil
  '(sacha-transform-html-remove-images
    sacha-transform-html-remove-italics
    sacha-brigade-remove-meta-recursively
    sacha-brigade-format-buttons)
  (with-temp-buffer (insert-file-contents "~/Downloads/test.html") (buffer-string)))

```

Changing link colours I also want to change the link colours to match the colour scheme. The newsletter has two parts distinguished by background colours. Bike Brigade updates use black text on a white background, and community updates use white text on a dark blue background so that they're visually distinct. For contrast, I like to use light blue links in the community section, which doesn't match the colour of the links when I paste them in from Slack. This meant manually recolouring the text and each of the links in Mailchimp, which was tedious.

This code changes the colours of the links. It also changes the colours of text by wrapping spans around them. It skips the links we turned into buttons.

```

(defvar sacha-brigade-community-text-style "color: #ffffff")

```



- Starting June 30 at 7pm
- Trinity Bellwoods on Tuesdays starting July 1 at 7pm

Check dispatch

Figure 23: Screenshot of button

COMMUNITY UPDATES



Partner highlight: Not Far From the Tree

Did you know that over 1.5 million pounds of fresh fruit grow in Toronto? [Not Far From the Tree](#) coordinates tree owners, volunteers, and community agency partners for a shared harvest.

Figure 24: Screenshot of community update colours

```

(defvar sachabrigadecommunitylinkstyle "color: #aed9ef")
;;;###autoload
(defun sachabrigaderecolorrecursively (node)
  "Change the colors of links and text in NODE.
Ignore links with the class mceButtonLink.
Uses `sachabrigadecommunitytextstyle' and `sachabrigadecommunitylinkstyle'."
  (pcase (dom-tag node)
    ('table node) ; pass through, don't recurse further
    ('a ; change the colour
      (unless (string= (or (dom-attr node 'class) "") "mceButtonLink")
        (dom-set-attribute node 'style sachabrigadecommunitylinkstyle))
      node)
    (_
      (let ((processed
              (seq-map
               (lambda (child)
                 (if (stringp child)
                     (dom-node 'span
                               (style . ,sachabrigadecommunitytextstyle) child)
                     (sachabrigaderecolorrecursively child))))
              (dom-children node))))
        `((, (dom-tag node) , (dom-attributes node) ,@processed))))))

```

I can add that to the sequence:

```

(sachabrigadetransformhtmlclipboard
 nil
 '(sachabrigadetransformhtmlremoveimages
    sachabrigadetransformhtmlremoveitalics
    sachabrigaderecolorrecursively
    sachabrigadeformatbuttons
    sachabrigaderecolorrecursively)
 (with-temp-buffer (insert-file-contents "~/Downloads/test.html") (buffer-string)))

```

Just the headings

```

;;;###autoload
(defun sachabrigadegustheadings (dom)
  (let ((entries
        (dom-node 'ul)))
    (dolist (tag (dom-by-tag dom 'h2))
      (let ((text (string-trim (dom-texts tag)))
            (unless (string= text "")
              (dom-append-child entries (dom-node 'li nil text)))))
      entries))

```

Wrapping it up Now that I've made all those little pieces, I can put them together in two interactive functions. The first function will be for the regular colour scheme, and the second function will be for the light-on-dark colour scheme. For convenience, I'll have it activate Google Chrome afterwards so that I can paste the results into the right block.

```

;;;###autoload
(defun sachabrigadetransformhtml (&optional recolor file as-rich-text)
  (interactive (list nil (when current-prefix-arg (read-file-name "File: "))))
  (sachabrigadetransformhtmlclipboard
   "Chrome")

```

```

      (append
        '(sacha-transform-html-remove-images
          sacha-transform-html-remove-italics
          sacha-brigade-remove-meta-recursively
          sacha-brigade-remove-styles
          sacha-brigade-format-buttons)
        (if recolor '(sacha-brigade-recolor-recursively)))
      (when file
        (with-temp-buffer (insert-file-contents file) (buffer-string)))
      as-rich-text))

;; ###autoload
(defun sacha-brigade-transform-community-html (&optional file as-rich-text)
  (interactive (list (when current-prefix-arg (read-file-name "File: "))))
  (sacha-brigade-transform-html t file as-rich-text))

;; ###autoload
(defun sacha-brigade-transform-just-headings (&optional file as-rich-text)
  (interactive (list (when current-prefix-arg (read-file-name "File: "))))
  (sacha-transform-html-clipboard
   "Chrome"
   '(sacha-brigade-just-headings)
   (when file
     (with-temp-buffer (insert-file-contents file) (buffer-string)))
   as-rich-text))

```

And then I can use links like this for quick shortcuts:

- `[[elisp:(sacha-brigade-transform-html nil "~/Downloads/test.html")]]`
- `[[elisp:(sacha-brigade-transform-community-html "~/Downloads/test.html")]]`
- `[[elisp:(sacha-brigade-transform-html)]]`

Since this pastes the results as formatted text, it's editable using the usual Mailchimp workflow. That way, other people can make last-minute updates.

With embedded images, the saved HTML is about 8 MB. The code makes quick work of it. This saves about 10-15 minutes per newsletter, so the time investment probably won't [directly pay off](#). But it also reduces annoyance, which is even more important than raw time savings. I enjoyed figuring all this out. I think this technique of transforming HTML in the clipboard will come in handy. By writing the functions as small, composable parts, I can change how I want to transform the clipboard.

Next steps It would be interesting to someday automate the campaign blocks while still making them mostly editable, as in the following examples:

- [tan-yong-sheng/mailchimp_auto](#): A command line application to automate the email campaign creation process on MailChimp based on the google spreadsheet input
- [Automating My Newsletter Generation with MailChimp, Google Sheets, and AWS Lambda - DEV Community](#)

Maybe someday!

(Also, hat tip to this [Reddit post](#) that helped me get xclip to work more reliably from within Emacs by adding `-filter 2>& /dev/null` to the end of my xclip call so it didn't hang.)

[2025-09-18 Thu]: Got it to include the dates in the TOC as well

I've been volunteering to help with the [Bike Brigade](#) newsletter. I like that there are people who are out there helping improve food security by delivering food bank hampers to recipients. Collecting information for the newsletter also helps me feel more appreciation for the lively Toronto biking scene, even though I still can't make it out to most events. The general workflow is:

1. collect info
2. draft the newsletter somewhere other volunteers can give feedback on
3. convert the newsletter to Mailchimp
4. send a test message
5. make any edits requested
6. schedule the email campaign

We have the [Mailchimp](#) Essentials plan, so I can't just export HTML for the whole newsletter. Someday I should experiment with services that might let me generate the whole newsletter from Emacs. That would be neat. Anyway, with Mailchimp's block-based editor, at least I can paste in HTML code for the text/buttons. That way, I don't have to change colours or define links by hand.

The logistics volunteers coordinate via [Slack](#), so a Slack Canvas seemed like a good way to draft the newsletter. I've previously written about my workflow for [copying blocks from a Slack Canvas and then using Emacs to transform the rich text](#), including recolouring the links in the section with light text on a dark background. However, copying rich text from a Slack Canvas turned out to be unreliable. Sometimes it would copy what I wanted, and sometimes nothing would get copied. There was no way to export HTML from the Slack Canvas, either.

I switched to using [Google Docs](#) for the drafts. It was a little less convenient to add items from Slack messages and I couldn't easily right-click to download the images that I pasted in. It was more reliable in terms of copying, but only if I used [xclip](#) to save the clipboard into a file instead of trying to do the whole thing in memory.

I finally got to spend a little time automating a new workflow. This time I exported the Google Doc as a zip that had the HTML file and all the images in a subdirectory. The HTML source is not very pleasant to work with. It has lots of extra markup I don't need. Here's what an entry looks like:

```
<span class="c1">Spreading the word at the East Lynn Farmers Market</span></h2><p class="c2"><span>Thanks to Iain and Sarah for representing Bike Brigade and Community Fridges Toronto at the East Lynn Farmers Market last Thursday! He said, "There were some really excited folks to learn about how we work and what we do, and for some, enthused to hear how they can get involved with our group!"</span> Check out the rest of </span><span class="c4"><a class="c7" href="https://www.google.com/url?q=https://www.facebook.com/groups/bikebrigadetoronto/posts/4207395666250243/&sa=D&source=editors&ust=1757808178718651&usg=A0vVaw2Ioajqz70Klm8js1BEC2Lu">his notes</a></span><span>&nbsp;</span>in this Facebook post. Want to join them in the tent on Oct 9 or maybe even on a regular basis? </span><span class="c4"><a class="c7" href="https://www.google.com/url?q=https://www.facebook.com/groups/bikebrigadetoronto/posts/4207395666250243/&sa=D&source=editors&ust=1757808178718937&usg=A0vVaw2qC13Q3Lb0JfJL9l2uTmdw">Comment</a></span><span>&nbsp;</span>or </span><span class="c4"><a class="c7" href="mailto:info@bikebrigade.ca">e-mail</a></span><span class="c0">!</span></p><p class="c2 c5"><span class="c0"></span><span></span></p><p class="c2"><span class="c4"><a class="c7" href="https://www.google.com/url?q=https://www.facebook.com/groups/bikebrigadetoronto/posts/4207395666250243/&sa=D&source=editors&ust=1757808178719468&usg=A0vVaw2IahBakMixvSD3plwv5c3q">[ Comment on Facebook ]</a></span></p><h2 class="c8" id=">
```

Figure 25: Exported HTML for an entry

Things I wanted to do with the HTML:

- Remove the google.com/url redirection for the links. Mailchimp will add its own redirection for click-tracking, but at least the links can look simpler when I paste them in.

- Remove all the extra classes and styles.
- Turn [call to action] into fancier Mailchimp buttons.

Also, instead of transforming one block at a time, I decided to make an Org Mode document with all the different blocks I needed. That way, I could copy and paste things in quick succession.

Here's what the result looks like. It makes a table of contents, adds the sign-up block, and adds the different links and blocks I need to paste into Mailchimp.

```
September 14, 2025
* In this e-mail
#+begin_src html
<p>Hi Bike Brigaders! Here's what's happening this week, with quick signup links. In this e-mail:</p><ul><li>Help wanted</li><li>Spreading the word at the East Lynn Farmers Market</li><li>Your safety is our top priority</li><li>August milestones</li><li>Ghost Bike for Samuel Willett</li><li>Pedal ride to Biddisage, community picnic</li><li>Scarborough community ride and picnic</li><li>World Car Free Day Ride into Fall</li><li>Grants for resident-led climate action projects, activities & events</li></ul>
#+end_src

* Sign-up block
#+begin_src html
<div style="background-color: #223f4d; text-align: center; max-width: 384px; margin: auto; margin-bottom: 12px;"><a href="https://dispatch.bikebrigade.ca/campaigns/signup?current_week=2025-09-15" target="_blank" class="mcebuttonlink" style="background-color: #223f4d;border-radius:8px;border:2px solid #223f4d;color:#ffffff;display:block;font-family:'Helvetica Neue', Helvetica, Arial, Verdana, sans-serif;font-size:16px;font-weight:normal;font-style:normal;padding:16px 28px;text-decoration:none;text-align:center;direction:ltr;letter-spacing:0px" rel="nofollow">SIGN UP NOW TO DELIVER SEP 15-28</a>
</div>
<p style="text-align: center; font-family: 'Helvetica Neue', Helvetica, Arial, Verdana"><a href="https://dispatch.bikebrigade.ca/campaigns/signup?current_week=2025-09-29" style="color: #478584; margin-top: 12px; margin-bottom: 12px;" target="_blank">You can also sign up early to deliver Sep 29-Oct 5</a></p>
#+end_src

* Bike Brigade
* Help wanted...
* Spreading the word at the East Lynn Farmers Market
copy:/home/sacha/proj/bike-brigade/newsletter/images/2025-09-21-news-spreading-the-word-at-the-east-lynn-farmers-market.jpg
https://www.facebook.com/groups/bikebrigadetoronto/posts/420739566258243/

#+begin_src html
<div><h2>Spreading the word at the East Lynn Farmers Market</h2><p>Thanks to Iain and Sarah for representing Bike Brigade and Community Fridges Toronto at the East Lynn Farmers Market last Thursday! He said, "There were some really excited folks to learn about how we work and what we do, and for some, enthused to hear how they can get involved with our group!" Check out the rest of <a href="https://www.facebook.com/groups/bikebrigadetoronto/posts/420739566258243/">his notes</a> in this Facebook post. Want to join them in the tent on Oct 9 or maybe even on a regular basis? <a href="https://www.facebook.com/groups/bikebrigadetoronto/posts/420739566258243/">Comment</a> or <a href="mailto:info@bikebrigade.ca">e-mail</a>.</p><p><table align="center" border="0" cellpadding="0" cellspacing="0" role="presentation" data-block-id="609" class="mcebuttoncontainer" style="margin: auto; text-align: center"><tbody><tr><td class="mcebutton"><div style="background-color: #808080;border-radius:8px;text-align:center" valign="top" class="mcebutton"><a href="https://www.facebook.com/groups/bikebrigadetoronto/posts/420739566258243/" target="_blank" class="mcebuttonlink" style="background-color:#000000;border-radius:8px;border:2px solid #000000;color:#ffffff;display:block;font-family:'Helvetica Neue', Helvetica, Arial, Verdana, sans-serif;font-size:16px;font-weight:normal;font-style:normal;padding:16px 28px;text-decoration:none;text-align:center;direction:ltr;letter-spacing:0px" rel="nofollow">Comment on Facebook</a></td></tr></tbody></table></div>
#+end_src

* Your safety is our top priority
copy:/home/sacha/proj/bike-brigade/newsletter/images/2025-09-21-news-your-safety-is-our-top-priority.jpg
https://www.bikebrigade.ca/code-of-conduct
```

Figure 26: Screenshot of newsletter Org file with blocks for easy copying

I need to copy and paste the image filenames into the upload dialog on Mailchimp, so I use my [custom Org Mode link type for copying to the clipboard](#). For the HTML code, I use `#+begin_src html ... #+end_src` instead of `#+begin_export html ... #+end_export` so that I can use [Embark](#) and [embark-org](#) to quickly copy the contents of the source block. (That doesn't work for export blocks yet.) I have `C-` bound to `embark-act`, the source block is detected by the functions that `embark-org.el` added to `embark-target-finders`, and the `c` binding in `embark-org-src-block-map` calls `embark-org-copy-block-contents`. So all I need to do is `C-. c` in a block to copy its contents.

```
;; ##autoload
(defun sachabrigade-process-latest-newsletter-draft (date)
  "Create an Org file with the HTML for different blocks."
  (interactive (list (if current-prefix-arg (org-read-date nil t nil "Date: ")
    ↵
    (org-read-date nil t "+Sun")))))
  (when (stringp date) (setq date (date-to-time date)))
  (let ((default-directory "~/Downloads/newsletter")
        file
        dom
        sections)
    (call-process "unzip" nil nil nil "-o" (sachalatest-file "~/Downloads" ↵
    ↵ "\\*.zip$"))
    (setq file (sachalatest-file default-directory))
    (with-temp-buffer
      (insert-file-contents-literally file)
      (goto-char (point-min))
      (setq dom (sachabrigade-simplify-html (libxml-parse-html-region ↵
    ↵ (point-min) (point-max))))
    (sachabrigade-save-newsletter-images dom)
```

```

(setq sections
  (sacha-html-group-by-tag
    'h1
    (dom-children
      (dom-by-tag
        dom 'body))))))
(with-current-buffer (get-buffer-create "*newsletter*")
  (erase-buffer)
  (org-mode)
  (insert
    (format-time-string "%B %-e, %Y" date) "\n"
    "* In this e-mail\n#+begin_src html\n"
    "<p>Hi Bike Brigaders! 'Heres what's happening this week, with quick
→ signup links. In this e-mail:</p>"
    (replace-regexp-in-string
      "<li>" "\n<li>"
      (with-temp-buffer
        (svg-print
          (apply 'dom-node
            'ul nil
            (append
              ←
              (sacha-brigade-toc-items (assoc-default "Bike Brigade" sections 'string=))
              ←
              (sacha-brigade-toc-items (assoc-default "In our community" sections 'string=))))))
            (buffer-string)))
      "\n<br />\n"
      (sacha-brigade-copy-signup-block date)
      "\n#+end_src\n\n")
    (dolist (sec '("Bike Brigade" "In our community"))
      (insert "*" sec "\n"
        (mapconcat
          (lambda (group)
            (let* ((item (apply ←
              ←
              (append
                ←
                (list ←
                  ←
                  (dom-node 'h2 nil (car group))))
                  ←
                  (cdr group))))
              (format "** ←
→ %s\n\n%s\n\n%s\n\n#+begin_src html\n\n#+end_src\n\n"
              ←
              (car group)
              ←
              (if image (org-link-make-string (concat "copy:" image)) "")
              ←
              (or (sacha-html-last-link-href item) ""))
              ←
              (sacha-transform-html
                ←
                (delq nil
                  ←
                  (list

```

[illegible]

```

                                (progn
                                  (when current-section
                                    (push (cons section-name (nreverse ↵
↵ current-section)) results)
                                    (setq current-section nil))
                                  (setq section-name (string-trim (dom-texts ↵
↵ node))))))
                                (when section-name
                                  (push node current-section)))
    (when current-section
      (push (cons section-name (reverse current-section)) results)
      (setq current-section nil))
    (nreverse results)))

;; ###autoload
(defun sach-html-last-link-href (node)
  "Return the last link HREF in NODE."
  (dom-attr (car (last (dom-by-tag node 'a))) 'href))

;; ###autoload
(defun sach-brigade-image (heading)
  "Find the latest image related to HEADING."
  (car
    (nreverse
      (directory-files sach-brigade-newsletter-images-directory
↵ ↵
↵ t (regexp-quote (sach-brigade-newsletter-heading-to-image-file-name heading))))))

```

Some of the functions it uses are in my config, particularly the section on [Transforming HTML clipboard contents with Emacs to smooth out Mailchimp annoyances: dates, images, comments, colours](#) (p. 612).

Along the way, I learned that `svg-print` is a good way to [turn document object models back into HTML](#).

When I saw two more events and one additional link that I wanted to include, I was glad I already had this code sorted out. It made it easy to paste the images and details into the Google Doc, reformat it slightly, and get the info through the process so that it ended up in the newsletter with a usefully-named image and correctly-coloured links.

I think this is a good combination of Google Docs for getting other people's feedback and letting them edit, and Org Mode for keeping myself sane as I turn it into whatever Mailchimp wants.

My next step for improving this workflow might be to check out other e-mail providers in case I can get Emacs to make the whole template. That way, I don't have to keep switching between applications and using the mouse to duplicate blocks and edit the code.

18.1.3 Bike Brigade: working with Mailchimp images

All right, let's try uploading the files ourselves. This uses <https://github.com/sachac/mailchimp-el>.

```

(use-package mailchimp :load-path "~/proj/mailchimp-el" :vc (:url ↵
↵ "https://github.com/sachac/mailchimp-el"))

```

```

;; ###autoload
(defun sach-brigade-reuse-or-upload-images (images)
  "Return an updated list of (section . images)."
```

```

(let* ((base-regexp "^.+?-news-\\|\\(\\.[0-9][0-9]\\)?\\.\\(jpg\\|png\\)$")
(recent-files
(mapcar
(lambda (o)
(cons
(cons 'base
(replace-regexp-in-string
base-regexp ""
(file-name-base (assoc-default 'name o))))
o))
(assoc-default 'files (mailchimp-recent-files 100))))
results)
(mapcar
(lambda (section)
(setcdr
section
(mapcar
(lambda (image)
(let* ((base
(if (eq (alist-get 'type image) 'emoji)
(replace-regexp-in-string "^:\\|:$" "" (alist-get 'alt image))
(and (assoc-default 'filename image)
(replace-regexp-in-string
base-regexp ""
(file-name-base (alist-get 'filename image))))))
(existing (and base (seq-find (lambda (o)
(string= base (alist-get 'base o)))
recent-files))))
(when base
(let ((url
(if existing
(assoc-default 'full_size_url existing)
(alist-get
'full_size_url
(mailchimp-upload-file
(alist-get 'filename image)
(if (eq (alist-get 'type image) 'emoji)
(concat
(replace-regexp-in-string "^:\\|:$" "" (alist-get 'alt image))
".")
(file-name-extension (alist-get 'filename image)))))))
(push (cons 'url url) image)
(push (cons (file-name-base (alist-get 'filename image)) url) results)))
(cdr section)))
section)
images)))

;; ###autoload
(defun sacha-brigade-toc (sections)
(replace-regexp-in-string
"<li>" "\n<li>"
(with-temp-buffer
(svg-print
(apply 'dom-node
'ul nil
(append
(sacha-brigade-toc-items ←
↪ (assoc-default "Bike Brigade" sections 'string=))

```



```
(list
```

```

    ↪ src=\"%s\" style=\"width: 100%%; max-width: 100%%\" alt=\"%s\"></td></tr>\" ↪
    ↪ (assoc-default 'url extra-image) (assoc-default 'alt extra-image nil ""))
    ↪ "")
    ↪ ;; No images
    ↪ (format "<table width=\"%100%%\" border=\"%0\" cellpadding=\"%0\" ↪
    ↪ align=\"%center\" style=\"%margin-top: 12px; margin-bottom: ↪
    ↪ 12px;\"><tbody>%s</tbody></table>\"
    ↪ (format "<tr><td colspan=\"%2\" style=\"%padding-top: 12px\">%s</td></tr>\"
    ↪ (sacha-transform-html
    ↪
    ↪ (delq nil
    ↪
    ↪
    ↪
    ↪ (list
    ↪
    ↪
    ↪ 'sacha-transform-html-remove-italics
    ↪
    ↪
    ↪ 'sacha-brigade-format-buttons
    ↪
    ↪
    ↪ (when recolor
    ↪
    ↪
    ↪ 'sacha-brigade-recolor-recursively)))
    ↪
    ↪ item))
    ↪
    ↪ ;; extra images?
    ↪ (if extra-image
    ↪ (format "<tr><td colspan=\"%2\" style=\"%padding-top: 12px\"><img ↪
    ↪ src=\"%s\" alt=\"%s\" style=\"%width: 100%%; max-width: 100%%\"></td></tr>\" ↪
    ↪ (assoc-default 'url extra-image) (assoc-default 'alt extra-image nil ""))
    ↪
    ↪ "")
    ↪
    ↪ ))
    ↪
    ↪ (sacha-html-group-by-tag 'h2 section)
    ↪
    ↪ "")
    ↪
    ↪
    ↪ ;; ##autoload
    ↪ (cl-defun sacha-brigade-block (text &key (bg "#223f4d")
    ↪
    ↪
    ↪ (style ↪
    ↪ "padding-left:24px;padding-right:24px;padding-top:12px;padding-bottom:12px"))
    ↪ (format
    ↪ " <table border=\"%0\" cellpadding=\"%0\" cellspacing=\"%0\" width=\"%100%%\" ↪
    ↪ style=\"%border-collapse:collapse\" role=\"%presentation\"><tbody><tr><td style= ↪
    ↪ \"%padding-top:0;padding-bottom:0;padding-right:0;padding-left:0;border:0;border-radius:0\" ↪
    ↪ valign=\"%top\"><table width=\"%100%%\" style= ↪
    ↪ \"%border:0;background-color:%s;border-radius:0\"><tbody><tr><td style=\"%s\" ↪
    ↪ class=\"%mceTextBlockContainer\"><div data-block-id=\"%738\" class=\"%mceText\" style= ↪
    ↪ \"%width:100%%\">%s</div></td></tr></tbody></table></td></tr></tbody></table>\"
    ↪
    ↪ bg
    ↪
    ↪ style
    ↪
    ↪ text))
    ↪
    ↪
    ↪ ;; ##autoload
    ↪ (defun sacha-brigade-update-images (dom images)
    ↪ "Replace image references.
    ↪ IMAGES is an alist of (filename . URL).\"
    ↪ (let ((all-images (mapcan 'cdr images))) ; todo: hash or alist
    ↪ (mapcar
    ↪ (lambda (node)
    ↪ (when-let* ((base (file-name-base (dom-attr node 'src)))
    ↪ (url
    ↪ (alist-get 'url

```



```

        (seq-find
        (lambda (o)
          (string= base
            (file-name-base (alist-get 'filename o))))
        all-images))))
      (dom-set-attribute node 'src url)))
    (dom-by-tag dom 'img)))
  dom)

;;;###autoload
(defun sacha-brigade-process-latest-newsletter-draft-with-images (date)
  "Create an Org file with the HTML for different blocks."
  (interactive (list (if current-prefix-arg (org-read-date nil t nil "Date: ")
    ↪
    ↪ (org-read-date nil t "+Sun"))))
    (when (stringp date) (setq date (date-to-time date)))
    (let ((default-directory "~/proj/bike-brigade/temp_newsletter/")
          file
          images
          dom
          sections
          html)
      ↪ (call-process "unzip" nil nil nil "-o" ↪
      ↪ (sacha-latest-file "~/Downloads" "\\*.zip$"))
      (setq file (sacha-latest-file default-directory ".html"))
      (with-temp-buffer
        (insert-file-contents-literally file)
        (goto-char (point-min))
        (setq dom (libxml-parse-html-region (point-min) (point-max)))
        (setq images (sacha-brigade-reuse-or-upload-images
          (sacha-brigade-save-newsletter-images dom)))
        (setq dom (sacha-brigade-simplify-html dom))
        (setq dom (sacha-brigade-update-images dom images))
        (setq sections
          (sacha-html-group-by-tag
            'h1
            (dom-children
              (dom-by-tag
                dom 'body)))))
        (setq html
          (replace-regexp-in-string "<p><span></span></p>" ""
            (concat
              "table ↪
↪ class=\"newsletter\" margin=0 cellpadding=0 cellspacing=0 ↪
↪ style=\"border-collapse:collapse\"><tbody><tr><td>"
              (sacha-brigade-block ↪
                ↪ (format "<table style=\"margin: auto\"><tbody><tr><td style=\"text-align: center; ↪
↪ color: #f3f3f3\"><div style=\"text-align: center; color: ↪
↪ #f3f3f3\">%s</div></td></tr></table>"
                ↪
                ↪
                ↪ (format-time-string "%B %-e, %Y" date))
                ↪
                ↪ :bg "#16232a"
                ↪
                ↪ :style "padding: 0px 24px 12px 24px")
                "<base ↪
↪ href=\"\"><style>table { border-collapse: collapse !important } table.newsletter { ↪
↪ border-collapse: collapse} .mceStandardButton a, table.sign-up a { text-decoration: ↪

```

```

↪ none }</style><table><tbody><tr><td style=\"padding: 12px 24px 12px 24px\"><p>Hi Bike ↪
↪ Brigaders! 'Heres what's happening this week, with quick signup links. In this ↪
↪ e-mail:</p>\"
                                                                    (sacha-brigade-toc sections)
                                                                    \"
                                                                    ↪
↪ (sacha-brigade-copy-signup-block date)
                                                                    ↪
↪ (sacha-brigade-format-section (assoc-default \"Bike Brigade\" sections #'string=) images)
                                                                    ↪
↪ \"</td></tr></tbody></table><table style=\"background-color:#223f4d;\"><tbody><tr><td ↪
↪ style=\"padding-left: 24px; padding-right: 24px\">\"
                                                                    (sacha-brigade-block \"<h1 ↪
↪ style=\"text-align: center;\"><span style= \"color:#ffffff;\">In our ↪
↪ community</span></h1>\" )
                                                                    ↪
↪ (sacha-brigade-format-section (assoc-default \"In our community\" sections #'string=) ↪
↪ images t)
                                                                    \"</td></tr></tbody></table>\"
                                                                    (if (assoc-default \"Other ↪
↪ updates\" sections #'string=)
                                                                    (format ↪
↪ \"<table><tbody><tr><td style=\"padding: 12px 24px 12px 24px\"><h2>Other ↪
↪ updates</h2>%s</td></tr></tbody></table>\"
                                                                    ↪
↪
                                                                    (sacha-transform-html
                                                                    ↪
↪
                                                                    nil
                                                                    ↪
↪
                                                                    (car (assoc-default \"Other updates\" sections ↪
↪ #'string=))))
                                                                    \" )
                                                                    ↪
↪ \"</td></tr></tbody></table>\")))
    (when (called-interactively-p 'any)
      (kill-new html))
    html))

```

18.1.4 Emacs: Updating a Mailchimp campaign using a template, sending test e-mails, and scheduling it

I'm helping other volunteers get on board with doing the [Blue Brigade](#) newsletter. Since not everyone has access to (or the patience for) [MailChimp](#), we've been using Google Docs to draft the newsletter and share it with other people behind the scenes. I've previously written about [getting a Google Docs draft ready for Mailchimp via Emacs and Org Mode](#), which built on my code for [transforming HTML clipboard contents to smooth out Mailchimp annoyances: dates, images, comments, colours](#). Now I've figured out how to update, test, and schedule the MailChimp campaign directly from Emacs so that I don't even have to go into the MailChimp web interface at all. I added those functions to [sachac/mailchimp-el](#).

I used to manually download a ZIP of the Google Docs newsletter draft. I didn't feel like figuring out authentication and Google APIs from Emacs, so I did that in a NodeJS script instead. [convert-newsletter.js](#) can either create or download the latest newsletter doc from our Google Shared Drive. ([google-api](#) might be helpful if I want to do this in Emacs, not sure.) If I call `convert-newsletter.js` with the `download` argument, it unpacks the zip into `~/proj/bike-brigade/temp_newsletter`, where my Emacs Lisp function for processing the latest newsletter draft with images can turn it into the HTML to insert into the HTML template I've previously created. I've been thinking about whether

I want to move my HTML transformation code to NodeJS as well so that I could run the whole thing from the command-line and possibly have other people run this in the future, or if I should just leave it in Emacs for my convenience.

Updating the campaign through the Mailchimp API means that I don't have to log in, replicate the campaign, click on the code block, and paste in the code. Very nice, no clicks needed. I also use TRAMP to write the HTML to a file on my server (`sacha-bike-brigade-output-file` is of the form `/ssh:hostname:/path/to/file`) so that other volunteers can get a web preview without waiting for the test email.

```
;; ##autoload
(defun sacha-brigade-next-campaign (&optional date)
  (setq date (or date (org-read-date nil nil "+Sun")))
  (seq-find
   (lambda (o)
     (string-match (concat "^" date)
                   (alist-get 'title (alist-get 'settings o))))
   (alist-get 'campaigns (mailchimp-campaigns 5))))

(defvar sacha-bike-brigade-output-file nil)

;; ##autoload
(defun sacha-brigade-download-newsletter-from-google-docs ()
  "Download the newsletter from Google Docs and puts it in ~/proj/bike-brigade/temp_newsletter/."
  (interactive)
  (let ((default-directory "~/proj/bike-brigade"))
    (delete-directory "~/proj/bike-brigade/temp_newsletter" t)
    (with-current-buffer (get-buffer-create "*Newsletter*")
      (erase-buffer)
      (display-buffer (current-buffer))
      (call-process "node" nil t t "convert-newsletter.js" "download"))))

;; ##autoload
(defun sacha-brigade-create-or-update-campaign (&optional use-local)
  (interactive (list current-prefix-arg))
  (let* ((date (org-read-date nil nil "+Sun"))
         (template-name "Bike Brigade weekly update")
         (list-name "Bike Brigade")
         (template-id
          (alist-get
           'id
           (seq-find
            (lambda (o)
              (string= template-name (alist-get 'name o)))
            (alist-get 'templates (mailchimp--request-json "templates")))))
         (list-id (seq-find
                  (lambda (o)
                    (string= list-name
                              (alist-get 'name o)))
                  (alist-get 'lists (mailchimp--request-json "lists")))))
         (campaign (sacha-brigade-next-campaign date))
         (body `((type . "regular")
                  (recipients (list_id . ,(alist-get 'id list-id))
                              (settings
                               (title . ,date)
                               (subject_line . "Bike Brigade: Weekly update")
                               (from_name . "Bike Brigade")
                               (reply_to . "info@bikebrigade.ca")))))
```

```

        (tracking
        (opens . t)
        (html_clicks . t))))))
(unless campaign
  (setq campaign (mailchimp--request-json
    "/campaigns"
    :method "POST"
    :body
    body)))
;; Download the HTML
(if use-local
  (progn
    (make-directory "~/proj/bike-brigade/temp_newsletter/" t)
    (let ((default-directory "~/proj/bike-brigade/temp_newsletter/"))
      (call-process "unzip" nil (get-buffer-create "*Newsletter*") nil
        (sacha-latest-file sacha-download-dir ".zip")))
    (let ((default-directory "~/proj/bike-brigade/"))
      (call-process "node" nil (get-buffer-create "*Newsletter*") nil
        "convert-newsletter.js"
        "resize")))
    (sacha-brigade-download-newsletter-from-google-docs))
;; Upload to Mailchimp
(mailchimp-campaign-update-from-template
  (alist-get 'id campaign)
  template-id
  (list
    (cons "main_content_area"
      (sacha-brigade-process-latest-newsletter-draft-with-images
        date))))
(when sacha-bike-brigade-output-file
  (with-temp-file sacha-bike-brigade-output-file
    (insert (alist-get 'html (mailchimp--request-json (format "/campaigns/%s/content" ↵
      ↵ (alist-get 'id campaign))))))
  (browse-url (concat "https://sachachua.com/bike-brigade/" (file-name-nondirectory ↵
    ↵ sacha-bike-brigade-output-file)))
  (message "%s" "Done!"))))

```

Now to send the test e-mails...

```

(defvar sacha-brigade-test-emails nil "Set to a list of e-mail addresses.")
;;;###autoload
(defun sacha-brigade-send-test-to-me ()
  (interactive)
  (mailchimp-campaign-send-test-email (sacha-brigade-next-campaign) user-mail-address))

;;;###autoload
(defun sacha-brigade-send-test ()
  (interactive)
  (if sacha-brigade-test-emails
    (mailchimp-campaign-send-test-email (sacha-brigade-next-campaign) ↵
      ↵ sacha-brigade-test-emails)
    (error "Set `sacha-brigade-test-emails'.")))

```

And schedule it:

```

;;;###autoload
(defun sacha-brigade-schedule ()
  (interactive)

```

```
(let* ((campaign (sacha-brigade-next-campaign))
      (sched (format-time-string "%FT%TZ" (org-read-date t t "+Sun 11:00") t)))
      (mailchimp-campaign-schedule campaign sched)
      (message "Scheduled %s" (alist-get 'title (alist-get 'settings campaign)))))
```

Progress, bit by bit! Here's a screenshot showing the Google Docs draft on one side and my web preview in the other:

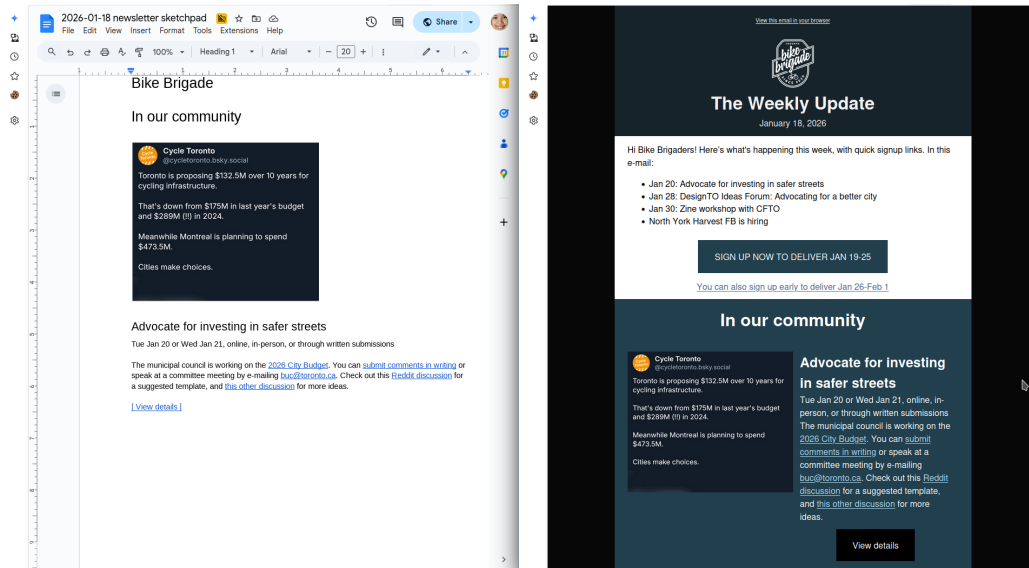


Figure 27: Google Docs and Mailchimp campaign preview

It'll be even cooler if I can get some of this working via systemd persistent tasks so that they happen automatically, or have some kind of way for the other newsletter volunteers to trigger a rebuild. Anyway, here's <https://github.com/sachac/mailchimp-el> in case the code is useful for anyone else.

19 Streaming

19.1 Mode for streaming

```
(setq sacha-stream-inbox-file "~/sync/topics/live.org")
(with-eval-after-load 'org-capture
  (add-to-list 'org-capture-templates
    `("u" "Update" item ; Update for the livestream
      (file+headline ,sacha-stream-inbox-file "Updates")
      "- %U %?"))
  (add-to-list 'org-capture-templates
    `("l" "Timestamp" item
      (file+headline ,sacha-stream-inbox-file "Timestamps")
      "- %U %i?"))))
```

```
(defvar sacha-stream-old-variable-values nil "Alist of values to save.")
(defvar sacha-stream-inbox-file nil "File to save new items to.")
(defvar sacha-stream-url nil "*URL of current livestream.")
(defvar sacha-stream-inbox-target
  `(file+headline ,sacha-stream-inbox-file "Current / notes for next time"))
```

```

(defvar sacha-stream-variables-to-override
  `((custom-enabled-themes . (ef-trio-dark))
    (org-agenda-files . ("/sync/stream/index.org" "~/sync/topics/live.org" ↵
      ↵ "~/sync/stream/inbox.org"))
    (org-refile-targets
      .
      (((("~/sync/stream/index.org"
          "~/sync/stream/inbox.org"
          "~/sync/topics/live.org"
          "~/sync/orgzly/posts.org"
          "~/sync/emacs/Sacha.org"
          "~/sync/orgzly/news.org") . t)))
    (sacha-org-inbox-file . ,sacha-stream-inbox-file)
    (sacha-file-shortcuts .
      ((("C" "~/proj/emacs-calendar/README.org" "Emacs calendar")
        ("e" "~/sync/emacs/Sacha.org" "Config")
        ("E" "~/sync/emacs-news/index.org" "Emacs News")
        ("f" "~/sync/orgzly/journal-fr.org" "French journal")
        ("F" "~/sync/orgzly/french.org" "French")
        ("i" "~/sync/topics/live.org" "Live")
        ("s" "~/proj/stream/index.org" "Yay Emacs")
        ("p" "~/sync/orgzly/posts.org" "Posts")
        ("n" "~/sync/topics/now.org" "Now")
        ("w" "~/sync/topics/workflows.org" "Workflows"))))
    (org-capture-templates
      .
      ((("r" "Note" entry
        ,sacha-stream-inbox-target
        "* %?\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n%i\n\n- %a\n%U"
        :prepend t)
        ("u" "Update" item
        (file+headline ,sacha-stream-inbox-file "Updates")
        "- %U %?"
        :prepend t)
        ("F" "Firefox link" entry
        ,sacha-stream-inbox-target
        "* %^{Note}\n:PROPERTIES:\n:CREATED: ↵
        ↵ %U\n:END:\n\n%(org-link-make-string\n(sacha-spookfox-complete-link))"
        ("f" "Firefox" entry
        ,sacha-stream-inbox-target
        "* %^{Note}\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n%(apply ↵
        ↵ #'org-link-make-string\n
        ↵ (append (spookfox-js-injection-eval-in-active-tab \"[window.location.href, ↵
        ↵ document.title]\" t) nil))"
        ("📰" "Emacs News" entry
        (file+headline "~/sync/orgzly/news.org"
          "Collect Emacs News")
        "* %a :news:\n\n#+begin_quote\n%:text\n#+end_quote\n\n"
        :prepend t :immediate-finish t)
        ("m" "Mastodon" entry
        ,sacha-stream-inbox-target
        "* %?\n\n#+begin_quote\n%:text\n#+end_quote\n\n%a"
        :prepend t)
        ("t" "Task with annotation" entry
        ,sacha-stream-inbox-target
        "* TODO %?\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n%a\n"
        :prepend t)
        ("i" "Interrupting task" entry

```

```

, sacha-stream-inbox-target
"* STARTED %^{Task}\n:PROPERTIES:\n:CREATED: %U\n:END:\n%a\n"
:clock-in :clock-resume :prepend t)
("T" "Task without annotation" entry
, sacha-stream-inbox-target
"* TODO %^{Task}\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n"
:prepend t)
("c" "Contents to current clocked task" plain
(clock) "%i%\n%a" :empty-lines 1)
("." "Today" entry
, sacha-stream-inbox-target
"* TODO %^{Task}\nSCHEDULED: %t\n:PROPERTIES:\n:CREATED: %U\n:END:\n"
:immediate-finish t)
("v" "Video" entry
, sacha-stream-inbox-target
"* TODO %^{Task} :video:\nSCHEDULED: %t\n"
:immediate-finish t)
("e" "Errand" entry
, sacha-stream-inbox-target
"* TODO %^{Task} :errands:\n:PROPERTIES:\n:CREATED: %U\n:END:\n"
:immediate-finish t)
("n" "Note" entry
, sacha-stream-inbox-target
"* %^{Note}\n:PROPERTIES:\n:CREATED: %U\n:END:\n"
:immediate-finish t)
("N" "Note" entry
, sacha-stream-inbox-target
"* %^{Note}\n:PROPERTIES:\n:CREATED: %U\n:END:\n"
:prepend t)
("s" "Selection from browser" entry
, sacha-stream-inbox-target
"* %a :website:\n:PROPERTIES:\n:CREATED: ↵
↳ %U\n:END:\n#+begin_quote\n%i\n#+end_quote\n\n%?\n"
:prepend t)
("S" "Screenshot" item
(file+headline ,sacha-stream-inbox-file "Timestamps")
"- %U [[file:%(sacha-latest-screenshot)]]%i%?" )
("l" "Timestamp" item
(file+headline ,sacha-stream-inbox-file "Timestamps")
"- %U %i%?" )
("q" "Quick note" item
, sacha-stream-inbox-target)
("w" "Web" entry (file ,sacha-stream-inbox-file)
"* %a\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n%i\n")
("W" "Web bookmark" entry
(file "~/sync/orgzly/resources.org")
"* %a\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n%i\n"
:prepend t)
("y" "Yay Emacs" entry
(file+headline "~/proj/yayemacs/index.org"
"Notes for this session")
"* %?\n:PROPERTIES:\n:CREATED: %U\n:END:\n\n%i\n\n%a\n")))))

;; ##autoload
(defun sach-stream-override-variables ()
  "Save and override variables for streaming."
  (interactive)
  (if sach-stream-old-variable-values
      (message "Values already saved? Not overriding.")

```

```

    (setq sach-stream-old-variable-values
      (mapcar
        (lambda (var)
          (progn1 (cons (car var) (symbol-value (car var)))
            (setopt--set
              (car var)
              (cdr var))))
          sach-stream-variables-to-override)))
    (when (featurep 'org)
      (org-refile-cache-clear)))

;;;###autoload
(defun sach-stream-restore-variables ()
  "Restore values saved in `sach-stream-old-variable-values'."
  (interactive)
  (mapc
    (lambda (var) (setopt--set (car var) (cdr var)))
    sach-stream-old-variable-values)
  (setq sach-stream-old-variable-values nil)
  (when (featurep 'org)
    (org-refile-cache-clear)))

;;;###autoload
(defun sach-stream-refresh-variables ()
  "Restore and override `sach-stream-variables-to-override'."
  (interactive)
  (sach-stream-restore-variables)
  (sach-stream-override-variables))

(defvar sach-stream-sensitive-files-regexps
  '("\\.gpg"
    "Inbox.org"
    "organizer.org"
    "business.org")
  "List of regexps matching files that would ideally not end up on stream.")

;;;###autoload
(defun sach-stream-clean-up-buffers ()
  "Clean up buffers that might be sensitive."
  (interactive)
  (condition-case nil
    (progn
      (org-save-all-org-buffers)
      (let (remaining cleaned)
        (mapc
          (lambda (buf)
            (with-current-buffer buf
              (when (and (buffer-file-name)
                ↵
                (seq-find (lambda (o) (string-match o (buffer-file-name)))
                ↵
                sach-stream-sensitive-files-regexps))
              (if (buffer-modified-p buf)
                (push ↵
                  (buffer-name buf) remaining)
                (push (buffer-name ↵
                  buf) cleaned)
                (kill-buffer buf))))))

```



```

                                (buffer-list))
                                (message "%s remaining, %s cleaned" remaining cleaned)))
    (error nil)))

(defvar sacha-stream-or-video-global-mode-done-hook nil "Functions to run after switching out ↔
↔ of the mode.")
(defvar-keymap sacha-stream-or-video-global-mode-map)
;; ###autoload
(define-minor-mode sacha-stream-or-video-global-mode
  "On air or doing a video."
  :init-val nil
  :keymap sacha-stream-or-video-global-mode-map
  :lighter "⌂"
  (if sacha-stream-or-video-global-mode
      (progn
        (sacha-stream-clean-up-buffers)
        (global-display-line-numbers-mode 1)
        (fontaine-set-preset 'presentation)
        (keycast-header-line-mode 1)
        (sacha-stream-override-variables)
        (cl-pushnew
         'sacha-marginalia-annotate-variable
         (alist-get 'variable marginalia-annotators)))
        (global-display-line-numbers-mode -1)
        (fontaine-set-preset 'regular)
        (keycast-header-line-mode -1)
        (setf
         (alist-get 'variable marginalia-annotators)
         (remove
          'sacha-marginalia-annotate-variable
          (alist-get 'variable marginalia-annotators)))
        (sacha-stream-restore-variables)
        (run-hooks 'sacha-stream-or-video-global-mode-done-hook))
      (sacha-navigate-set-up-file-shortcuts))

```

```

(use-package fontaine
  :config
  (setq fontaine-presets
    '((regular :default-height 100)
      (presentation :default-height 140))))

```

19.2 Display large text (and maybe QR code?)

I want to display a short string for emphasis and to make it easier for people to copy.

```

(defun sacha-stream-calculate-fit-to-width-height (str &optional face)
  (let* ((window-width (window-body-width nil t))
        (current-height 100)
        (margin 10)
        (current-windows (current-window-configuration))
        (string-width
         (with-temp-buffer
          (insert (propertize str 'face face))
          (switch-to-buffer (current-buffer))
          (car (window-text-pixel-size nil (point-min) (point-max))))))
    (progn

```

```

    (if (> string-width 0)
        (round (* current-height (/ (float (- window-width margin)) string-width)))
        current-height)
    (set-window-configuration current-windows)))

(defvar sacha-stream-show-string-hook nil "Additional functions to call with the text.")

(declare-function 'qrencode-format "qrencode")
(declare-function 'qrencode "qrencode")

;;###autoload
(defun sacha-stream-show-string (text &optional qr)
  "Display large TEXT in a buffer, centered.
If QR is a string, display a QR code for it below.
Otherwise, if QR is non-nil, display a QR code for TEXT."
  (interactive (list (read-string "Text: ") t))
  (with-current-buffer (get-buffer-create "*stream*")
    (erase-buffer)
    (let ((char-height (sacha-stream-calculate-fit-to-width-height text))
          win-height
          above-space)
      (switch-to-buffer (current-buffer))
      (delete-other-windows)
      (insert (propertize (concat text "\n") 'face `(:height ,char-height) 'display-text t))
      (setq-local cursor-type nil)
      (when qr
        (insert
         (propertize
          (qrencode-format
           (qrencode (if (stringp qr) qr
                        text)
                      nil nil
                      'return-raw))
          'face 'qrencode-face)))
        (run-hook-with-args 'sacha-stream-show-string-hook text)
        ;;(setq win-height (window-body-height nil t))
        ;;(setq above-space (max 0 (/ (- win-height char-height) 2)))
        ;;(goto-char (point-min))
        ;; (put-text-property
        ;;  (line-beginning-position)
        ;;  (1+ (line-end-position))
        ;;  'line-height (+ above-space (/ char-height 2)))
        )))

;; Let's add a show: org link type

(defun sacha-stream-org-export (path desc format _)
  "Export stream."
  (or desc path))

(defun sacha-stream-org-insert-description (link &optional description)
  (unless description
    link))

(defun sacha-stream-org-complete ()
  "Prompt for stream."
  (interactive)
  (let* ((v-or-f (symbol-at-point))
        (found (if v-or-f (cl-some (lambda (x) (funcall (nth 1 x) v-or-f))
```

```

                                describe-symbol-backends)))
  (v-or-f (if found v-or-f (function-called-at-point)))
  (found (or found v-or-f)))
(concat "stream:"
      (completing-read (format-prompt "Describe symbol"
                                     (and found v-or-f))
                      #'help--symbol-completion-table
                      (lambda (vv)
                        (cl-some (lambda (x) (funcall (nth 1 x) vv))
                               describe-symbol-backends))
                      t nil nil
                      (if found (symbol-name v-or-f))))))

(defun sacha-stream-org-open (path)
  (sacha-stream-show-string path t))

```

```

(with-eval-after-load 'org
  (org-link-set-parameters "stream"
                          :complete #'sacha-stream-org-complete
                          ↵
                          :insert-description #'sacha-stream-org-insert-description
                          ↵
                          :export #'sacha-stream-org-export
                          ↵
                          :follow #'sacha-stream-org-open))

```

```

(with-eval-after-load 'embark
  (keymap-set embark-general-map "Z" 'sacha-stream-show-string))
(with-eval-after-load 'sacha-stream
  (keymap-set sacha-stream-or-video-global-mode-map "C-M-z" 'sacha-stream-show-string))
(setq sacha-stream-show-string-hook
      '((lambda (text)
          (sacha-socialstream-send-message
            (concat (format-time-string "%-H:%M%p: ")
                    text)))
        (lambda (text)
          (org-capture-string text "l")
          (org-capture-finalize)))))

```

19.3 OBS websockets

```

(with-eval-after-load 'transient
  (transient-define-prefix sacha-stream-transient ()
    ["Browse"
     ("bl" "Live"
      (lambda () (interactive))
      (browse-url ↵
        "https://studio.youtube.com/channel/UC1T2UAbC6j7TqOWurVhkuHQ/livestreaming/dashboard"))
     ("bu" "Upcoming"
      (lambda () (interactive))
      (browse-url ↵
        "https://studio.youtube.com/channel/UC1T2UAbC6j7TqOWurVhkuHQ/livestreaming/manage"))
     ("bd" "Dashboard"
      (lambda () (interactive))
      (browse-url "https://studio.youtube.com/channel/UC1T2UAbC6j7TqOWurVhkuHQ")))]

```

```

("bw" "Web"
 (lambda () (interactive)
  (browse-url "https://yayemacs.com")))]
["Other"
 ("o" "OBS" sachas-obs-websocket-transient)
 ("m" (lambda ()
  (propertize
   "Mode"
   'face
   (if sachas-stream-or-video-global-mode
    'success
    'error)))
 sachas-stream-or-video-global-mode)])

(transient-define-prefix sachas-obs-websocket-transient ()
 "Transient menu for controlling OBS via websocket."
 :refresh-suffixes t
 ["OBS"
  ("c" (lambda () (propertize "Connect to OBS" 'face 'error))
   obs-websocket-connect :if-not obs-websocket-connected-p)
  ("c" (lambda () (propertize "Disconnect from OBS" 'face 'success))
   obs-websocket-disconnect :if obs-websocket-connected-p)]
 ["Output"
  :if obs-websocket-connected-p
  ("s" (lambda () (propertize "Stream - begin" 'face 'error)) obs-websocket-start-streaming
   :if-not obs-websocket-streaming-p)
  ("s" (lambda () (propertize "Stream - end" 'face 'success)) obs-websocket-stop-streaming
   :if obs-websocket-streaming-p)
  ("r" (lambda () (propertize "Record - begin" 'face 'error)) obs-websocket-start-recording ↵
   ↵ :if-not obs-websocket-recording-p)
  ("r" (lambda () (propertize "Record - end" 'face 'success)) obs-websocket-stop-recording ↵
   ↵ :if obs-websocket-recording-p)]
 ["Scenes"
  :if obs-websocket-connected-p
  ("d" "Desktop"
   (lambda () (interactive)
    (obs-websocket-send "SetCurrentProgramScene" :sceneName "Desktop"))))
  ("e" "Emacs"
   (lambda () (interactive)
    (obs-websocket-send "SetCurrentProgramScene" :sceneName "Emacs"))))
  ("i" "Intermission"
   (lambda () (interactive)
    (obs-websocket-send "SetCurrentProgramScene" :sceneName "Intermission"))))
  ("P" "Preview current scene" obs-websocket-preview-current-scene)]
 ["Items"
  :if obs-websocket-connected-p
  ("w" "Toggle webcam" (lambda () (interactive) (obs-websocket-toggle-scene-item-enabled ↵
   ↵ :sourceName "Webcam")))]))

(use-package obs-websocket
 :load-path "~/proj/obs-websocket-el"
 :preface (load "~/proj/obs-websocket-el/obs-websocket-autoloads.el" t t)
 :bind
  ((("C-c v" . sachas-stream-transient)
   ("s-v" . sachas-stream-transient)
   ("C-h h" . obs-websocket-stop-streaming) ; panic button; Help help!
   ))

```

Let's have some fun.

```
(defvar sach-obs-websocket-transforms
  `((webcam-big . (:scene-item "Webcam" :positionX 0.0 :positionY 0.0 :scaleX 2.05 :scaleY 2.05))
    (webcam-small . (:scene-item "Webcam" :positionX 1313.0 :positionY 842.0 :scaleX 0.495 ↵
      ↵ :scaleY 0.495))))

;;;###autoload
(defun sach-obs-websocket-webcam-really-big ()
  (interactive)
  (let ((transform (assoc-default 'webcam-big sach-obs-websocket-transforms)))
    (obs-websocket-set-scene-item-transform
     (plist-get transform :scene-item)
     transform)))

;;;###autoload
(defun sach-obs-websocket-webcam-normal ()
  (interactive)
  (let ((transform (assoc-default 'webcam-small sach-obs-websocket-transforms)))
    (obs-websocket-set-scene-item-transform
     (plist-get transform :scene-item)
     transform)))
```

19.4 Comment overlay via socialstream.ninja

<https://socialstream.ninja>

```
(defvar sach-socialstream-session nil "Session code (not the full URL).")

;;;###autoload
(defun sach-socialstream-send-message (text)
  "Send TEXT."
  (interactive "MText: ")
  (url-retrieve
   (format "https://io.socialstream.ninja/%s/sendChat/null/%s"
           sach-socialstream-session
           (url-hexify-string text))
   #'ignore))
```

19.5 Custom Org link type for hints (and sound effects)

```
(defun sach-org-hint-export (path desc format _)
  "Export hint."
  (pcase format
    ((or 'html '11ty 'md)
     (format "<label class=\"hint\"><input type=\"checkbox\"> <span ↵
       ↵ class=\"hint-desc\">%s</span><span class=\"hint-text\">%s</span></label>" desc path))
     ('ascii
      desc)))

(defvar sach-org-hint-functions nil
  "Functions to call with the hint as the argument.")

(defface sach-org-hint-face
  '(t :inherit font-lock-keyword-face :weight bold))
```

```

"Face for hints.")

(defun sacha-org-hint-open (path)
  "Display the hint at PATH."
  (let ((overlay (car (org-find-overlays 'sacha-org-hint)))
        (text (replace-regexp-in-string "^hint:" "" path))
        elem)
    (if overlay
      (delete-overlay overlay)
      (setq elem (org-element-context))
      (setq overlay (make-overlay (org-element-begin elem) (org-element-end elem)))
      (overlay-put overlay 'display text)
      (overlay-put overlay 'evaporate t)
      (overlay-put overlay 'sacha-org-hint text)
      (overlay-put overlay 'face 'sacha-org-hint-face)
      (run-hook-with-args 'sacha-org-hint-functions text))))

(defun sacha-org-hint-reset ()
  "Remove all hint overlays"
  (interactive)
  (remove-overlays (point-min) (point-max) 'sacha-org-hint))

(defun sacha-org-hint-play-sound (text)
  "Play sound for TEXT.
Match it against `sacha-org-hint-sound-alist'."
  (when-let* ((sound (assoc-default text sacha-org-hint-sound-alist #'string=)))
    (start-process "mpv" nil "mpv" (expand-file-name sound) "--no-video" "--force-window=no"))))

```

To activate:

```

(with-eval-after-load 'org
  (setq sacha-org-hint-sound-alist
        '(("yup" . "~/proj/stream/correct.mp3")
          ("nope" . "~/proj/stream/wrong.mp3")))
  (setq sacha-org-hint-functions '(sacha-org-hint-play-sound))
  (org-link-set-parameters "hint"

    ↪      :export #'sacha-org-hint-export
    ↪      :follow #'sacha-org-hint-open))

```

19.6 Chat

```

(defun sacha-stream-chat-process nil)
(defvar sacha-stream-chat-command `("npx" "masterchat-cli" "stream" "-n" "-t" "all"))
(defvar sacha-stream-chat-buffer "*stream-chat*")
;; ##autoload
(defun sacha-stream-chat-start (&optional url)
  "Start the process if it's not already running."
  (interactive (if (process-live-p sacha-stream-chat-process)
    nil
    (list (read-string "URL: "))))
  (if url
    (unless (process-live-p sacha-stream-chat-process)
      (with-current-buffer (get-buffer-create sacha-stream-chat-buffer)
        (erase-buffer)
        (setq sacha-stream-chat-process

```

```

        (make-process
         :name "stream-chat"
         :command (append sachā-stream-chat-command (list url))
         :buffer (get-buffer-create sachā-stream-chat-buffer)
         :stderr (get-buffer-create "*stream-chat-err*"))))
      (display-buffer (current-buffer)))
    (if (string= (buffer-name) sachā-stream-chat-buffer)
        (bury-buffer)
        (switch-to-buffer sachā-stream-chat-buffer))))

;;;###autoload
(defun sachā-stream-chat-stop ()
  (interactive)
  (kill-process sachā-stream-chat-process))

```

```
(keymap-global-set "s-c" #'sachā-stream-chat-start)
```

19.7 DONE Send currently-clocked task title to file, include in stream stream

```

(defvar sachā-stream-display-file nil)

;;;###autoload
(defun sachā-stream-obs-display-text (text)
  "Display TEXT in the current task area in my OBS."
  (interactive (list (read-string "Text: ")))
  (when sachā-stream-display-file
    (with-temp-buffer
      (org-mode)
      (write-region
       (if (string= text "")
           ""
           (string-join (org-wrap text 100) "\n"))
       nil sachā-stream-display-file))))

;;;###autoload
(defun sachā-stream-obs-org-display-current-task ()
  "Display the current task on OBS."
  (interactive)
  (sachā-stream-obs-display-text
   (if (member "stream" (org-get-tags))
       (org-entry-get (point) "ITEM")
       "")))

;;;###autoload
(defun sachā-org-clear-streaming-task ()
  "Clear the text."
  (sachā-stream-obs-display-text ""))

```

```

(setq sachā-stream-display-file "~/proj/stream/current-task.txt")
(with-eval-after-load 'org-clock
  (add-hook 'org-clock-in-hook #'sachā-stream-obs-org-display-current-task)
  (add-hook 'org-clock-out-hook #'sachā-org-clear-streaming-task))

```

19.8 Stream agenda

```
;;;###autoload
(defun sacha-stream-agenda ()
  (interactive)
  (org-agenda nil "s"))
```

19.9 Simple streaming with FFmpeg

[2024-01-20 Sat]: As it turns out, my X230T can't handle streaming with FFMpeg. I get like half a frame a second. I'll try using vdo.ninja to send my screen over to another computer that might be able to handle OBS.

My X230T can't handle OBS, but maybe I can stream with FFmpeg. [Quick notes on livestreaming to YouTube with FFmpeg on a Lenovo X230T](#)

```
# From pacmd list-sources | egrep '^\\s+name'
export ↵
  ↵ LAPEL=alsa_input.usb-Jieli_Technology_USB_Composite_Device_433035383239312E-00.mono-fallback ↵
  ↵ #
export YETI=alsa_input.usb-Blue_Microphones_Yeti_Stereo_Microphone_REV8-00.analog-stereo
export SYSTEM=alsa_output.pci-0000_00_1b.0.analog-stereo.monitor
# MIC=$LAPEL
# AUDIO_WEIGHTS="1 1"
export MIC=$YETI
export AUDIO_WEIGHTS="0.5 0.5"
export OFFSET=+1920,430
export SIZE=1280x720
export SCREEN=LVDS-1 # from xrandr
xrandr --output $SCREEN --mode 1280x720
```

I switch to a larger size and a light theme. I also turn [consult](#) previews off to minimize the risk of leaking data through buffer previews.

[prepare-for-emacsconf-screenshots-or-recordings](#) (p. 497)

```
source ~/bin/setup-laptop-for-streaming
DATE=$(date "+%Y-%m-%d-%H-%M-%S")
ffmpeg -f x11grab -framerate 30 -video_size $SIZE -i :0.0$OFFSET -f pulse -i $MIC -f pulse -i ↵
  ↵ $SYSTEM -filter_complex ↵
  ↵ "amix=inputs=2:weights=$AUDIO_WEIGHTS:duration=longest:normalize=0[audio]" -c:v ↵
  ↵ libx264 -preset fast -maxrate 690k -bufsize 2000k -g 60 -vf format=yuv420p -c:a aac ↵
  ↵ -b:a 96k -y -f tee -map 0:v -map '[audio]' -flags +global_header ↵
  ↵ "/home/sacha/recordings/$DATE.flv|[f=flv]rtmp://a.rtmp.youtube.com/live2/$YOUTUBE_KEY"
```

```
source ~/bin/setup-laptop-for-streaming
DATE=$(date "+%Y-%m-%d-%H-%M-%S")
ffmpeg -f x11grab -framerate 30 -video_size $SIZE -i :0.0$OFFSET -f pulse -i $MIC -f pulse -i ↵
  ↵ $SYSTEM -filter_complex ↵
  ↵ "amix=inputs=2:weights=$AUDIO_WEIGHTS:duration=longest:normalize=0[audio]" -c:v ↵
  ↵ libx264 -preset fast -maxrate 690k -bufsize 2000k -g 60 -vf format=yuv420p -c:a aac ↵
  ↵ -b:a 96k -map 0:v -map '[audio]' -y "/home/sacha/recordings/$DATE.flv"
```

Some code to start and stop the stream:


```

(defvar sacha-stream-process nil)
(defvar sacha-stream-type nil)
(defvar sacha-stream-offset-seconds 2 "Number of seconds to offset timestamps.")
(defvar sacha-stream-start-time nil)

;;;###autoload
(defun sacha-stream-toggle ()
  (interactive)
  (if (process-live-p sacha-stream-process)
      (sacha-stream-stop)
      (sacha-stream-start)))

;;;###autoload
(defun sacha-recording-toggle ()
  (interactive)
  (if (process-live-p sacha-stream-process)
      (sacha-recording-stop)
      (sacha-recording-start)))

;;;###autoload
(defun sacha-stream-start ()
  (interactive)
  (unless (process-live-p sacha-stream-process)
    (unless (getenv "YOUTUBE_KEY")
      (setenv "YOUTUBE_KEY" (auth-info-password (auth-source-search :host ←
↪ "https://studio.youtube.com"))))
    (setq sacha-stream-type 'stream)
    (setq sacha-stream-start-time (current-time))
    (setq sacha-stream-process (start-process "ffmpeg" (get-buffer-create ←
↪ "*stream-ffmpeg*")
↪
↪
↪ "bash" (expand-file-name "~/bin/stream-laptop")))
    (message "Streaming.")))

;;;###autoload
(defun sacha-recording-start ()
  (interactive)
  (unless (process-live-p sacha-stream-process)
    (setq sacha-stream-type 'record)
    (setq sacha-stream-start-time (current-time))
    (setq sacha-stream-process (start-process "ffmpeg" (get-buffer-create ←
↪ "*stream-ffmpeg*")
↪
↪
↪ "bash" (expand-file-name "~/bin/record-laptop")))
    (message "Recording.")))

;;;###autoload
(defun sacha-stream-stop ()
  (interactive)
  (when (process-live-p sacha-stream-process)
    (setq sacha-stream-type nil)
    (setq sacha-stream-start-time nil)
    (stop-process sacha-stream-process)
    (kill-process sacha-stream-process)))

(defalias 'sacha-recording-stop #'sacha-stream-stop)

```

```
;; ##autoload
(defun sacha-recordings-dired ()
  (interactive)
  (dired sacha-recordings-dir "-lt"))
```

Let's have relative timestamps:

```
;; ##autoload
(defun sacha-stream-insert-timestamp ()
  (interactive)
  (when sacha-stream-start-time
    (let ((time (format-secs "%.2h:%z%.2m:%.2s"
                              (- (time-to-secs (current-time))
                                  (time-to-secs sacha-stream-start-time)
                                  (if (eq sacha-stream-type 'stream)
                                      sacha-stream-offset-secs 0))))
      (insert (org-link-make-string
              (concat "video:" (sacha-latest-file
                                "~/recordings" "flv")
                  time)
              " "))))))

;; ##autoload
(defun sacha-stream-set-recording-file ()
  (interactive)
  (org-entry-put (point) "RECORDING"
                 (sacha-latest-file
                  "~/recordings" "flv")))
```

Ideas for next steps:

- Add a [REC] indicator to the modeline.
- Get video links to jump to the right spot, maybe based on org-media-note or subed

19.10 Controlling my stream audio from Emacs: background music, typing sounds, and push to talk emacs

- 2021-02-11: Parsed `pacmd list-sources` so that I can mute/unmute devices by regular expression.
- 2021-02-07: Made it work with my USB microphone.

I was experimenting with streaming Emacs geeking around on [twitch.tv](https://www.twitch.tv). Someone asked me to have soft background music and typing sounds. Since I'm a little clueless about music and don't want to bother with hunting down nice royalty-free music, I figured I could just use the [Mozart dice game](#) to programmatically generate music.

I installed the [mozart-dice-game](#) NPM package and used this bit of Javascript to generate a hundred MIDI files.

```
const x = require('mozart-dice-game')
```

```
for (let i = 0; i < 100; i++) { x.saveMinuet('minuet' + String(i).padStart('3', '0') + ↵
  ↵ '.mid'); }
```

Then I wrote this Emacs Lisp function to turn it on and off.

```
(defvar sacha-background-music-process nil "Process for playing background music")
;; ###autoload
(defun sacha-stream-toggle-background-music (&optional enable)
  (interactive)
  (if (or sacha-background-music-process
        (and (numberp enable) (< enable 0)))
      (progn
        (when (process-live-p sacha-background-music-process)
          (kill-process sacha-background-music-process))
        (setq sacha-background-music-process nil))
      (let ((files (directory-files "~/proj/music" t "mid\\")))
        (setq sacha-background-music-process
              (apply
               'start-process
               "*Music*"
               nil
               (append (list "timidity" "-idl" "--volume=10") files)))))))
```

People also suggested typing sounds. I guess that's a good way to get a sense of activity. The default selectric sound was a little too loud for me, so we'll use the move sound for now. It would be nice to make this more random-sounding someday.

```
;; ###autoload
(defun sacha-selectric-type-sound ()
  "Make the sound of typing."
  ;; Someday, randomize this or something
  (selectric-make-sound (expand-file-name "selectric-move.wav" selectric-files-path)))
```

```
(use-package selectric-mode
  :if sacha-laptop-p
  :diminish ""
  :defer t
  :commands selectric-mode
  :config
  (fset #'selectric-type-sound #'sacha-selectric-type-sound))
```

[2024-01-10 Wed]: I'm using the Blue Yeti microphone now, so I can use the hardware mute button instead of push to talk.

I was having a hard time remembering to go back on mute during meetings, since the LED on the mute button wasn't working at the time and the system tray icon was a little hard to notice. The LED has mysteriously decided to start working again, but push-to-talk is handy anyway. I want to be able to tap a key to toggle my microphone on and off, and hold it down in order to make it push-to-talk. It looks like my key repeat is less than 0.5 seconds, so I can set a timer that will turn things off after a little while. This code doesn't pick up any changes that happen outside Emacs, but it'll do for now. I used `pacmd list-sources` to list the sources and get the IDs.

```
;; ###autoload
(defun sacha-pacmd-set-device (regexp status)
```

```

(with-current-buffer (get-buffer-create "*pacmd*")
  (erase-buffer)
  (shell-command "pacmd list-sources" (current-buffer))
  (goto-char (point-max))
  (let (results)
    (while (re-search-backward regexp nil t)
      (when (re-search-backward "index: \\([[:digit:]]+\\)" nil t)
        (setq results (cons (match-string 1) results))
        (shell-command-to-string (format "pacmd set-source-mute %s %d"
                                          (match-string 1)
                                          (if (equal status 'on) 0 1)))))
      results)))

(defvar sach-mic-p nil "Non-nil means microphone is on")

;;;###autoload
(defun sach-mic-off ()
  (interactive)
  (sach-pacmd-set-device "Yeti" 'off)
  (sach-pacmd-set-device "Internal Microphone" 'off)
  (setq sach-mic-p nil))
;;;###autoload
(defun sach-mic-on ()
  (interactive)
  (sach-pacmd-set-device "Yeti" 'on)
  (sach-pacmd-set-device "Internal Microphone" 'on)
  (setq sach-mic-p t))
;;;###autoload
(defun sach-mic-toggle ()
  (interactive)
  (if sach-mic-p (sach-mic-off) (sach-mic-on)))

(defvar sach-push-to-talk-mute-timer nil "Timer to mute things again.")
(defvar sach-push-to-talk-last-time nil "Last time sach-push-to-talk was run")
(defvar sach-push-to-talk-threshold 0.5 "Number of seconds")

;;;###autoload
(defun sach-push-to-talk-mute ()
  (interactive)
  (message "Muting.")
  (sach-mic-off)
  (force-mode-line-update)
  (when obs-websocket-recording-p (sach-obs-websocket-add-caption "[Microphone off]")))

;;;###autoload
(defun sach-push-to-talk ()
  "Tap to toggle microphone on and off, or repeat the command to make it push to talk."
  (interactive)
  (cond
   ((null sach-mic-p) ;; It's off, so turn it on
    (when (timerp sach-push-to-talk-mute-timer)
      (cancel-timer sach-push-to-talk-mute-timer))
    (sach-mic-on)
    (when obs-websocket-recording-p (sach-obs-websocket-add-caption "[Microphone on]"))
    (setq sach-push-to-talk-last-time (current-time)))
   ((timerp sach-push-to-talk-mute-timer) ;; Push-to-talk mode
    (cancel-timer sach-push-to-talk-mute-timer)
    (setq sach-push-to-talk-mute-timer
      (run-at-time sach-push-to-talk-threshold nil #'sach-push-to-talk-mute))))

```

```
;; Might be push to talk, if we're within the key repeating time
((< (- (time-to-seconds (current-time)) (time-to-seconds sachapush-to-talk-last-time))
  sachapush-to-talk-threshold)
 (setq sachapush-to-talk-mute-timer
  (run-at-time sachapush-to-talk-threshold nil #'sachapush-to-talk-mute)))
;; It's been a while since I turned the mic on.
(t (sachapush-to-talk-mute))))

(global-set-key (kbd "<f12>") #'sachapush-to-talk)
```

```
(defvar sachamic-p)
(add-to-list 'mode-line-front-space '(:eval (if sachamic-p "*MIC*" "")))
```

19.11 More background music

```
;;;###autoload
(defun sachastream-emms-toggle-background ()
  (interactive)
  (unless (emms-playlist-buffer-list)
    (emms-play-directory "~/sync/Phone/music/freepd/"))
  (emms-pause)
  (emms-show))
```

19.12 Show Emacs-related tasks

```
;;;###autoload
(defun sachashow-emacs-tasks ()
  (interactive)
  (org-ql-search (org-agenda-files)
    '(and (todo)
      (parent (and (or (tags "stream")
        (tags "project"))
        (tags "emacs")
        (not (tags "inactive"))))))
    :title "Emacs-related project tasks"
    :sort '(date priority todo)
    :super-groups '(:auto-parent t)))
```

19.13 Stream message

```
;;;###autoload
(defun sachastream-message (message)
  (interactive "MMessage: ")
  (with-temp-file "~/proj/stream/message.html"
    (insert "<style>body { font-size: large; color: white; font-family: ↵
↵ sans-serif; padding: 10px; background-color: black }</style>"
      message))
  (shell-command "scp ~/proj/stream/message.html web:/var/www/yayemacs.com"))
```

19.14 Playing recordings

```
(use-package mpv :if sacha-laptop-p :defer t :commands mpv)
```

```
(defvar sacha-recordings-dir "~/recordings/")
;; ###autoload
(defun sacha-delete-latest-recording ()
  (interactive)
  (delete-file (sacha-latest-file sacha-recordings-dir)))
;; ###autoload
(defun sacha-open-latest-recording ()
  (interactive)
  (find-file (sacha-latest-file sacha-recordings-dir)))
;; ###autoload
(defun sacha-play-latest-recording (&optional arg)
  (interactive "P")
  (let ((latest (sacha-latest-file sacha-recordings-dir)))
    (if (and arg (file-exists-p (sacha-obs-websocket-caption-file latest)))
        (with-current-buffer (find-file-noselect (sacha-obs-websocket-caption-file ←
          ↪ (sacha-latest-file sacha-recordings-dir)))
          (goto-char (point-min))
          (subed-mpv-find-video latest)
          (pop-to-buffer (current-buffer)))
        (mpv-play (sacha-latest-file sacha-recordings-dir))))))
;; ###autoload
(defun sacha-rename-last-recording ()
  (interactive)
  (let ((latest (sacha-latest-file sacha-recordings-dir))
        (new-name (read-string "New name: " (format-time-string ←
    ↪ "%Y-%m-%d-"))))
    (rename-file latest
      (expand-file-name
        (concat new-name
          ↪
    ↪ (if (and (file-name-extension latest) (null (file-name-extension ←
    ↪ new-name)))
          ↪
    ↪ (concat "." (file-name-extension latest))
          ↪
    ↪ ""))
      sacha-recordings-dir))))
;; ###autoload
(defun sacha-upload-recording (recording tags)
  (interactive (list (let ((latest (sacha-latest-file sacha-recordings-dir "mkv\\|mp4\\|webm")))
    (read-file-name "Recording: " sacha-recordings-dir latest t)
    (read-string "Tags: " "emacs"))))
  (start-process "youtube-upload" nil "youtube-upload" recording "--privacy=unlisted" ←
    ↪ "--license=creativecommons"
    (format
      "--tags=\"%s\""
      tags)
    "--open-link"
    (format "--title=%s" (shell-quote-argument (file-name-base recording)))
    (format "--client-secrets=%s" google-video-credentials)))
```

With a [websocket connection to OBS](#), this allows me to iterate over my audio setup a little more conveniently. The workflow is:

- Press a key to start recording.
- Say something.
- Press a key to stop recording.
- Play back the recording after a brief delay.

```
;; ##autoload
(defun sacha-stream-test-recording ()
  "Start recording, stop recording, and play it back."
  (interactive)
  (obs-websocket-start-recording)
  (read-key "Press any key to stop recording.")
  (obs-websocket-stop-recording)
  (sit-for 1)
  (sacha-play-latest-recording)
  (let (file (sacha-latest-file sacha-recordings-dir))
    (when (y-or-n-p (format "Delete %s?" file))
      (delete-file file))))
```

19.15 Stream notes

```
;; ##autoload
(defun sacha-org-save-and-tangle-stream-notes ()
  (when (and (buffer-file-name)
              (string= (expand-file-name (buffer-file-name))
                        ↵
                        (expand-file-name "~/proj/stream/index.org"))
            (add-hook 'after-save-hook #'sacha-stream-publish-and-sync-notes nil t)))
  ;; ##autoload
  (defun sacha-stream-publish-and-sync-notes ()
    (interactive)
    (with-current-buffer (find-file "~/proj/stream/index.org")
      (org-html-export-to-html)
      (let ((org-icalendar-timezone "America/Toronto")
            (org-icalendar-date-time-format ":%Y%m%dT%H%M%SZ"))
        (org-icalendar-export-to-ics)
        (shell-command "rsync -aze ssh ./ web:/var/www/yayemacs.com"))))
```

```
(with-eval-after-load 'org
  (add-hook 'org-mode-hook 'sacha-org-save-and-tangle-stream-notes))
;; based on https://www.reddit.com/r/emacs/comments/57nps0/comment/d8umsr4/?context=3
(setq imp-default-user-filters '((org-mode . sacha-impatient-org-export-as-html-filter)
                                (mhtml-mode . nil)
                                (html-mode . nil)
                                (web-mode . nil)))
```

```
;; based on https://www.reddit.com/r/emacs/comments/57nps0/comment/d8umsr4/?context=3
;; ##autoload
(defun sacha-imp-htmlize-filter (buffer)
  "Alternate htmlization of BUFFER before sending to clients."
  ;; leave the result in the current-buffer
  (let ((noninteractive t)
        (org-export-use-babel nil))
    (m (with-current-buffer buffer major-mode))))
```

```

(case m
  (org-mode
    (insert
      (with-current-buffer buffer
        (org-export-as 'html))))
  (t
    (let ((html-buffer (save-match-data (htmlize-buffer buffer))))
      (insert-buffer-substring html-buffer)
      (kill-buffer html-buffer)))))

;;;###autoload
(defun sach-impatient-org-export-as-html-filter (buffer)
  (let ((output-buffer (current-buffer))
        (log-message-max nil))
    (with-current-buffer buffer
      (let ((output (org-export-as 'html)))
        (with-current-buffer output-buffer (insert output))))))

```

```

(use-package impatient-mode
  :config
    (setq impatient-mode-delay 1)
    (setq httpd-port 8085)
    (imp-set-user-filter 'my/impatient-org-export-as-html-filter))

```

19.16 Chapters

```

;;;###autoload
(defun sach-youtube-copy-chapters ()
  "Call from a VTT file with NOTE comments."
  (interactive)
  (let ((subtitles (subed-subtitle-list)))
    (kill-new
      (concat (if (elt (car subtitles) 4)
                  ""
                  "0:00 Intro\n")
              (mapconcat (lambda (o)
                            (if (elt o 4)
                                (concat (format-seconds "%m:%.2s" (/ (elt o 2) 1000))
                                        " ")
                                (elt o 4))
                            "\n")
                          subtitles
                          ""))))))

```

19.17 STARTED Download and post-process


```

(defvar sach-stream-download-dir "~/proj/yay-emacs/" "Directory to save files into.")
(defvar sach-stream-transcribe-command "~/bin/whisperx-large" "Command to use for ↵
↳ transcription.")
;;;###autoload
(defun sach-stream-download-and-process (&optional url title)
  "Download MP3 and MP4 versions, and rename to TITLE."
  (interactive (list sach-stream-url (org-entry-get (point) "ITEM")))
  (let ((base (expand-file-name (sach-make-slug title) sach-stream-download-dir))
        (default-directory sach-stream-download-dir))
    (with-current-buffer (get-buffer-create "*Download MP3")
      (let ((download-process
              (start-process "download-mp3" ↵
↳ (current-buffer) "yt-dlp" "-x" "--audio-format" "mp3"
↳
↳ url
↳
↳ "-o"
↳
↳ (concat base ".mp3"))))
        (set-process-sentinel
         download-process
         (lambda (process event)
           (message "Downloading MP3")
           (start-process "transcribe" (get-buffer-create ↵
↳ "*Download MP3")
↳
↳ sach-stream-transcribe-command
↳
↳ (concat base ".mp3")))))
      (with-current-buffer (get-buffer-create "*Download MP4")
        (set-process-sentinel
         (start-process "download-mp4" (current-buffer) "yt-dlp"
↳ "-S" ↵
↳ url
↳ "-o"
↳
↳ (concat base ".mp4"))
        (lambda (process event)
          (message "Downloading MP4: %s" (string-trim event)))))))

```

19.18 **CANCELLED** Try continuous streaming and the Google Speech Recognition API

With data logging \$0.004 USD / 15 seconds

```

(defvar sach-stream-captions-websocket nil)
(defvar sach-stream-captions-history nil)
(defvar sach-stream-captions-last-caption nil)
(defvar sach-stream-captions-insert nil "Non-nil means insert into the current buffer.")
;;;###autoload
(defun sach-stream-captions-insert () (interactive) (setq sach-stream-captions-insert (not ↵
↳ sach-stream-captions-insert)))

(define-minor-mode sach-stream-captions-minor-mode "Toggle the captions server."
  :lighter "CAP"

```

```

:global t)

;;;###autoload
(defun sacha-get-last-n-chars (text limit)
  (if (< (length text) limit)
      text
      (substring text (- (length text) limit))))

;;;###autoload
(defun sacha-stream-captions-on-message (websocket frame)
  (let* ((payload (let ((json-object-type 'plist) (json-array-type 'list)) ←
    ↪ (json-read-from-string (websocket-frame-payload frame)))
        (type (plist-get payload :type))
        (caption (string-trim (plist-get (car (plist-get (car (plist-get (plist-get payload ←
    ↪ :stream) :results)) :alternatives)) :transcript)))))

    (if (string= type "interim")
        (when (websocket-openp obs-websocket) (obs-websocket-send "SendCaptions" :text ←
    ↪ (sacha-get-last-n-chars caption 80)))
        (setq sacha-stream-captions-last-caption caption)
        (call-process "notify-send" nil nil nil caption)
        (sacha-obs-websocket-add-caption caption)
        (when sacha-stream-captions-insert (insert caption))
        (setq sacha-stream-captions-history (cons caption sacha-stream-captions-history)))))

;;;###autoload
(defun sacha-stream-captions-edit-last (caption)
  (interactive (list (read-string "Caption: " sacha-stream-captions-last-caption ←
    ↪ 'sacha-stream-captions-history sacha-stream-captions-last-caption)))
  (when (> (length caption) 0)
    (sacha-obs-websocket-add-caption caption)))

```

```

(keymap-global-set "<f11>" 'sacha-stream-captions-edit-last)

```

```

;;;###autoload
(defun sacha-stream-captions-on-close (&rest args)
  (message "Captions websocket closed.")
  (sacha-stream-captions-minor-mode 0)
  (setq sacha-stream-captions-websocket nil))

;;;###autoload
(defun sacha-stream-captions-websocket-connect ()
  (interactive)
  (setq sacha-stream-captions-history nil)
  (sacha-stream-captions-minor-mode 1)
  (setq sacha-stream-captions-websocket (websocket-open "ws://localhost:8085"
    :on-message ←
    ↪ #'sacha-stream-captions-on-message
    :on-close ←
    ↪ #'sacha-stream-captions-on-close)))

(defvar sacha-stream-captions-process nil)
;;;###autoload
(defun sacha-stream-captions-start ()
  (interactive)
  (let ((default-directory "~/proj/speech")))

```

```
(setq sach-stream-captions-process (start-process "Stream captions" (get-buffer-create ↵
↵ "stream captions") "node" "test.js"))
(sleep-for 2)
(sach-stream-captions-websocket-connect)))

;;###autoload
(defun sach-stream-captions-sentinel (process event)
  (let ((status (process-status sach-stream-captions-process)))
    (if (member status '(stop exit signal))
        (sach-stream-captions-minor-mode -1))))
;;###autoload
(defun sach-stream-captions-stop ()
  (interactive)
  (stop-process sach-stream-captions-process))
```

19.19 Make chapter markers and video time hyperlinks easier to note while I livestream org:emacs

I want to make it easier to add chapter markers to my YouTube video descriptions and hyperlinks to specific times in videos in my blog posts.

19.19.1 Capture timestamps

Using wall-clock time via Org Mode timestamps like [2026-04-16 Thu 10:40] makes more sense to me than using video offsets because they're independent of any editing I might do.

C-u C-c C-! (org-timestamp-inactive) creates a timestamp with a time. I probably do often enough that I should create a Yasnippet for it:

```
# -*- mode: snippet -*-
# name: insert time
# key: zt
# --
`(format-time-string "[%Y-%m-%d %a %H:%M]")`
```

I also have Org capture templates, like this:

```
(with-eval-after-load 'org-capture
  (add-to-list
   'org-capture-templates
   `("l" "Timestamp" item
     (file+headline ,sach-stream-inbox-file "Timestamps")
     "- %U %i%?")))
```

I've been experimenting with a custom Org Mode link type "stream:" (p. 645) which:

- displays the text in a larger font with a QR code for easier copying
- sends the text to the YouTube chat via socialstream.ninja
- adds a timestamped note using the org-capture template above

Here is an example of that link in action. It's the (Log) link that I clicked on.

```
(compile-media-sync
 '((combined (:source
```

```

↔ " /home/sacha/proj/yay-emacs/ye16-sacha-and-prot-talk-emacs.mp4"
                                     ↳
                                     :original-start-ms "51:09"
                                     :original-stop-ms "51:16"))

    (combined (:source

↔ " /home/sacha/proj/yay-emacs/ye16-sacha-and-prot-talk-emacs-link-overlay.png"
                                     ↳
                                     :output-start-ms "0:03"
                                     :output-stop-ms "0:04"))

    (combined (:source

↔ " /home/sacha/proj/yay-emacs/ye16-sacha-and-prot-talk-emacs-qr-chat-overlay.png"
                                     ↳
                                     :output-start-ms "0:05"
                                     :output-stop-ms "0:06"))))
"/home/sacha/proj/yay-emacs/ye16.1-stream-show-string-and-calculate-offset.mp4")

```

/home/sacha/proj/yay-emacs/ye16.1-stream-show-string-and-calculate-offset.mp4

I used it in [YE16: Sacha and Prot talk Emacs](#). It was handy to have a link that I could click on instead of trying to remember a keyboard shortcut and type text. For example, these are the timestamps that were filed under org-capture:

- [2026-04-16 Thu 10:39] Getting more out of livestreams
- [2026-04-16 Thu 10:57] Announcing livestreams
- [2026-04-16 Thu 11:05] Processing the recordings
- [2026-04-16 Thu 11:11] Non-packaged code

Here's a short function for getting those times:

```

(defun sach-org-time-at-point ()
  "Return Emacs time object for timestamp at point."
  (org-timestamp-to-time (org-timestamp-from-string (org-element-property :raw-value ↔
↔ (org-element-context))))))

```

Next, I wanted to turn those timestamps into a hh:mm:ss offset into the streamed video.

19.19.2 Calculate an Org timestamp's offset into a YouTube stream

I post my YouTube videos under a brand account so that just in case I lose access to my main sacha@sachachua.com Google account, I still have access via my @gmail.com account. To enable YouTube API access to my channel, I needed to get my brand account's email address and set it up as a test user.

1. Go to <https://myaccount.google.com/brandaccounts>.
2. Select the account.
3. Click on View general account info
4. Copy the ...@pages.plusgoogle.com email address there.
5. Go to <https://console.cloud.google.com/>
6. Enable the YouTube data API for my project.
7. Download the credentials.json.
8. Go to Data Access - Audience

9. Set the User type to External
10. Add my brand account as one of the Test users.
11. Log in at the command line. Here's a sample command that uses the emacs-stuff project I created and the credentials.json I downloaded:

Then the following code calculates the offset of the timestamp at point based on the livestream that contains it.

```

(defun sacha-google-youtube-live-get-broadcast-at-time (time)
  "Return the broadcast encompassing TIME."
  (seq-find
    (lambda (o)
      (or
        ;; actual
        (and
          (alist-get 'actualStartTime (alist-get 'snippet o))
          (alist-get 'actualEndTime (alist-get 'snippet o))
          (not (time-less-p time (date-to-time (alist-get 'actualStartTime ←
↪ (alist-get 'snippet o))))))
        (time-less-p time (date-to-time (alist-get 'actualEndTime (alist-get ←
↪ 'snippet o))))))
      ;; actual, not done yet
      (and
        (alist-get 'actualStartTime (alist-get 'snippet o))
        (null (alist-get 'actualEndTime (alist-get 'snippet o)))
        (not (time-less-p time (date-to-time (alist-get 'actualStartTime ←
↪ (alist-get 'snippet o))))))
      ;; scheduled
      (and
        (null (alist-get 'actualStartTime (alist-get 'snippet o)))
        (null (alist-get 'actualEndTime (alist-get 'snippet o)))
        (not (time-less-p time (date-to-time (alist-get 'scheduledStartTime ←
↪ (alist-get 'snippet o))))))
      (sort
        (seq-filter
          (lambda (o)
            (or
              (alist-get 'actualStartTime (alist-get 'snippet o))
              (alist-get 'scheduledStartTime (alist-get 'snippet o))))
          (alist-get 'items
            ↪
            (sacha-google-youtube-live-broadcasts)))
        :key (lambda (o)
          (or
            (alist-get 'actualStartTime (alist-get ←
↪ 'snippet o))
            (alist-get 'scheduledStartTime (alist-get ←
↪ 'snippet o))))
        :lessp #'string<)))

(defun sacha-google-youtube-live-seconds-offset-from-start-of-stream (wall-time)
  "Return number of seconds for WALL-TIME from the start of the stream that contains it.
Offset by `sacha-google-youtube-stream-offset-seconds'."
  (when (and wall-time
    (alist-get 'actualStartTime
      ↪
      (alist-get 'snippet
        ↪
        (sacha-google-youtube-live-get-broadcast-at-time wall-time))))
    (+ sacha-google-youtube-stream-offset-seconds
      (time-to-seconds
        (time-subtract
          wall-time
          (date-to-time
            (alist-get 'actualStartTime

```

```

↪ 'snippet
                                            (alist-get ↪
                                            ↪
↪
↪
↪ (sacha-google-youtube-live-get-broadcast-at-time wall-time)))))))))

(defun sacha-google-youtube-convert-timestamps-to-chapters (timestamps)
  "Return a string of chapters for the Youtube video.
Use this when there is no embedded player."
  (mapconcat
    (lambda (o)
      (concat
        (sacha-google-youtube-stream-offset
          o)
        " "
        (replace-regexp-in-string "^\\[.+?\\]" " " o)))
    timestamps
    "\n"))

;;;###autoload
(defun sacha-google-clear-cache ()
  "Clear cached Google access tokens and data."
  (interactive)
  (setq sacha-google-access-token nil)
  (setq sacha-google-youtube-live-broadcasts nil))

```

For example:

```

(mapcar
  (lambda (o)
    (concat
      "vtime:"
      (sacha-google-youtube-stream-offset
        o)
      " "
      (replace-regexp-in-string "^\\[.+?\\]" " " o)))
  timestamps)

```

- `vtime:19:09` Getting more out of livestreams
- `vtime:37:09` Announcing livestreams
- `vtime:45:09` Processing the recordings
- `vtime:51:09` Non-packaged code

It's not exact, but it gets me in the right neighbourhood. Then I can use the MPV player to figure out a better timestamp if I want, and I can use my custom vtime Org link time to make those clickable when people have Javascript enabled. See [YE16: Sacha and Prot talk Emacs](#) for examples.

It could be nice to log seconds someday for even finer timestamps. Still, this is handy already!

19.20 Create a Google Calendar event from an Org Mode timestamp `org:emacs`

Time zones are hard, so I let calendaring systems take care of the conversion and confirmation. I've been using Google Calendar because it synchronizes with my phone and people know what to do with the event invite. [Org Mode has iCalendar export](#), but I sometimes have a hard time getting .ics files into Google Calendar on my laptop, so I might as well just create the calendar entry in

Google Calendar directly. Well. Emacs is a lot more fun than Google Calendar, so I'd rather create the calendar entry from Emacs and put it into Google Calendar.

This function lets me start from a timestamp like `[2026-04-24 Fri 10:30]` (inserted with `C-u C-c C-!`, or `org-timestamp-inactive`) and create an event based on a template.

```
(defvar sachatime-zone "America/Toronto" "Full name of time zone.")

;;;###autoload
(defun sachatime-zone (&optional time)
  "Create a Google Calendar invite based on TIME or the Org timestamp at point."
  (interactive (list (sachatime-at-point)))
  (browse-url
   (format
    "\n\n↔ \"https://calendar.google.com/calendar/render?action=TEMPLATE&text=%s&details=%s&dates=%s&ctz=%s\"
    (url-hexify-string sachatime-title)
    (url-hexify-string sachatime-description)
    (format-time-string
     \"%Y%m%dT%H%M%S\" time)
    sachatime-zone))))

(defvar sachatime-title "Emacs Chat" "Title of calendar entry.")
(defvar sachatime-description
  "All right, let's try this! => See the calendar invite for the Google Meet link."

  Objective: Share cool stuff about Emacs workflows that's not obvious from reading configs, and ↔
  ↔ have fun chatting about Emacs

  Some ideas for things to talk about:
  - Which keyboard shortcuts or combinations of functions work really well for you?
  - What's something you love about your setup?
  - What are you looking forward to tweaking next?

  Let me know if you want to do it on stream (more people can ask questions) or off stream (we ↔
  ↔ can clean up the video in case there are hiccups). Also, please feel free to send me ↔
  ↔ links to things you'd like me to read ahead of time, like your config!"
  "Description.")
```

It uses this function to convert the timestamp at point:

```
(defun sachatime-at-point ()
  "Return Emacs time object for timestamp at point."
  (org-timestamp-to-time (org-timestamp-from-string (org-element-property :raw-value (org-element-context)))))
```

19.21 Create YouTube livestream broadcasts from Emacs Lisp

I want to take advantage of this relatively predictable focus time while the kiddo is in school by getting back into the swing of scheduling conversations with people. I've been experimenting with [livestreaming these](#)

```
(declare-function sachatime-date-to-iso-utc "sachatime-lisp")

(defun sachatime-stream-youtube-format-broadcast (&key time title description privacy
  ↔
  ↔
  ↔
  end-time
  ↔
  ↔)
```



```

↪          :title title
↪          :description description
↪          :privacy privacy
↪          :end-time end-time
↪          :auto-start auto-start
↪          :auto-stop auto-stop)))
      response)
    (setq response
      (request-response-data
        (request ↪
↪ "https://www.googleapis.com/youtube/v3/liveBroadcasts?part=snippet,contentDetails,status"
          :type "POST"
          :headers `(("Authorization" . ,(format ↪
↪ "Bearer %s" (sacha-google-access-token)))
          ↪
↪ ("Accept" . "application/json")
          ↪
↪ ("Content-Type" . "application/json")))
          :data body
          :sync t
          :parser #'json-read)))
    (push response (alist-get 'items sacha-google-youtube-live-broadcasts))
    (when thumbnail
      (sacha-stream-youtube-set-video-thumbnail (alist-get 'id response) ↪
↪ thumbnail))
      (when (called-interactively-p 'any)
        (let ((url (concat "https://youtube.com/live/" (alist-get 'id ↪
↪ response))))
          (message "%s" url)
          (kill-new url)))
        response))

;; ##autoload
(defun sacha-stream-youtube-set-video-thumbnail (video-id thumbnail)
  "Set VIDEO-ID to THUMBNAIL."
  (request "https://www.googleapis.com/upload/youtube/v3/thumbnails/set"
    :type "POST"
    :params `(("videoId" . ,video-id))
    :headers `(("Authorization" . ,(format "Bearer %s" (sacha-google-access-token)))
              ("Content-Type" .
↪
↪ ,(mailcap-file-name-to-mime-type thumbnail))
              ("Accept" . "application/json")))
    :data (with-temp-buffer
            (set-buffer-multibyte nil)
            (insert-file-contents-literally thumbnail)
            (buffer-string))
    :sync t))

;; (sacha-stream-youtube-schedule-livestream :time "2026-04-23 10:30" :end-time "2026-04-23 ↪
↪ 11:30" :title "Emacs Chat: James Endres Howell" :description "I chat with James Endres ↪
↪ Howell (https://jamesendreshowell.com/ , https://fediscience.org/@jameshowell) about ↪
↪ Emacs and life.\n\nhttps://sachachua.com/topic/emacs-chat" :privacy "public" ↪
↪ :auto-start json-false :auto-stop json-false :thumbnail ↪

```

```

↳ "~/recordings/2026-04-17-07-53-05 Emacs Chat thumbnail - James.png")
;; (sacha-stream-youtube-schedule-livestream :time "2026-05-07 10:30" :end-time "2026-05-07
↳ 11:30" :title "Emacs Chat: Shae Erisson" :description "I chat with Shae Erisson
↳ (https://shapr.github.io/ , https://recurse.social/@shapr) about Emacs and
↳ life.\n\nhttps://sachachua.com/topic/emacs-chat" :privacy "public" :auto-start
↳ json-false :auto-stop json-false :thumbnail "~/recordings/2026-04-17-07-54-23 Emacs
↳ Chat thumbnail - Shae.png")

```

Now we can quickly schedule a livestream based on the Org entry at point.

```

;; ##autoload
(defun sacha-stream-org-schedule-livestream-for-entry-at-point ()
  "Schedule a livestream for the entry at point."
  (interactive)
  (let* ((time (org-timestamp-from-string (org-entry-get (point) "SCHEDULED")))
         (start-time (org-timestamp-to-time (org-timestamp-split-range
↳ time)))
         (end-time (org-timestamp-to-time (org-timestamp-split-range
↳ time t)))
         (response (sacha-stream-youtube-schedule-livestream
                    :title
↳ (org-entry-get (point) "ITEM")
                    :description
↳ (org-export-string-as (sacha-org-subtree-text) 'sacha-plain-text t)
                    :time start-time
                    :end-time
↳ end-time
                    :privacy "public"
                    :auto-start
↳ json-false
                    :auto-stop
↳ json-false
                    :thumbnail
↳ "~/proj/yay-emacs/thumbnail.png"))
         (url (concat "https://youtube.com/live/" (alist-get 'id
↳ response))))
    (org-entry-put (point) "YOUTUBE_URL" url)
    (insert "yt:" url)
    (cl-pushnew
     response
     (alist-get 'items sacha-google-youtube-live-broadcasts))
    response))

```

```

(defun sacha-date-to-time (time)
  "Return Emacs time object for TIME."
  (when (stringp time)
    (if (and (stringp time)
             (boundp 'org-ts-regexp)
             (string-match org-ts-regexp time))
        (setq time (org-timestamp-to-time (org-timestamp-from-string
↳ (match-string 0 time))))
        (setq time (date-to-time time))))
    time)

;; ##autoload
(defun sacha-date-to-iso-utc (time)
  "Return ISO8601 for TIME."
  (format-time-string "%FT%TZ" (sacha-date-to-time time) t))

```

19.22 Insert an Org Mode link to an upcoming YouTube broadcast

```
(defun sacha-google-youtube-complete-broadcast (&optional prompt)
  "Return the ID from recent or upcoming broadcasts."
  (let ((items (mapcar (lambda (o)
    (let-alist o
      (cons (concat .snippet.title
        (if-let* ((time
          (or
            .snippet.actualStartTime
            .snippet.scheduledStartTime)))
          (format-time-string
            "%-m/%d %H:%M"
            (date-to-time time))
          ""))
        o)))
    (alist-get 'items (sacha-google-youtube-live-broadcasts)))
    (completion-extra-properties
      (or completion-extra-properties
        (list :display-sort-function 'identity))))
    (assoc-default (completing-read (or prompt "Broadcast: ")
      items)
      items
      #'string=)))

;;###autoload
(defun sacha-google-youtube-insert-embed (broadcast)
  "Insert YT embed."
  (interactive (list (sacha-google-youtube-complete-broadcast)))
  (insert (concat "yt:https://youtube.com/live/" (alist-get 'id broadcast))))

;;###autoload
(defun sacha-google-youtube-current-or-next (&optional filter)
  "Return the current or next broadcast."
  (let ((now (format-time-string "%Y-%m-%dT%H:%M:%SZ" nil t)))
    (car
```

```

        (sort
          (seq-filter
            (lambda (o)
              (let-alist o
                (and
                  (or (null filter)
                      (string-match filter ←
                        ↪ .snippet.title)))
                  (or (and .snippet.actualStartTime
                          (not ←
                            ↪ .snippet.actualEndTime))
                      (not (string< ←
                        ↪ .snippet.scheduledStartTime now))))))
            (alist-get 'items
                      ↪
                        (sacha-google-youtube-live-broadcasts)))
          :key (lambda (o) (let-alist o .snippet.scheduledStartTime))
          :lessp #'string<)))

;;;###autoload
(defun sacha-google-youtube-upcoming-broadcasts ()
  "Return upcoming broadcasts."
  (sort
    (seq-remove
      (lambda (o)
        (let-alist o
          (or (null .snippet.scheduledStartTime) ←
            ↪ .snippet.actualStartTime)))
      (alist-get 'items
                (sacha-google-youtube-live-broadcasts)))
    :key (lambda (o) (let-alist o .snippet.scheduledStartTime)) :lessp 'string<))

(defun sacha-google-youtube-org-format-medium (o)
  "Timestamp, title, and description for O."
  (let-alist o
    (let ((url (concat "https://youtube.com/live/" .id)))
      (format "[%s] %s\n\n%s"
                (date-to-time .snippet.scheduledStartTime)
                (format-time-string "%Y-%m-%d %H:%M" ←
                  ↪ (org-link-make-string
                     url
                     .snippet.title)
                    .snippet.description))))

(defun sacha-google-youtube-org-format-embed (video)
  "Current or upcoming embed for VIDEO."
  (let-alist video
    (let ((url (concat "https://youtube.com/live/" .id)))
      (if .snippet.actualStartTime
          (format "Watch the current stream on YouTube: ←
            ↪ %s\n\n%s\n\n%s"
                  (org-link-make-string ←
                    ↪ url .snippet.title)
                  (concat "yt:" url)
                  .snippet.description)
          (format "At [%s] (%s): %s\n\n%s\n\n%s"
                  (format-time-string (cdr ←
                    ↪ org-timestamp-formats)

```

```

↩ (date-to-time ↩
↩ .snippet.scheduledStartTime))

sacha-time-zone
(org-link-make-string url ↩

↩ .snippet.title)

(concat "yt:" url)
.snippet.description))))

```

20 Miscellaneous

20.1 Ledger

Make it easier to review my credit card transactions

```

(use-package ledger-mode
  :mode "\\\\.ledger$"
  :bind (:map ledger-mode-map
    ("C-c C-n" . sacha-ledger-change-account)
    ("C-c a" . sacha-ledger-set-unknown-account)
    ("C-c f" . (lambda () (interactive) (find-file (sacha-latest-file ↩
↩ "~Downloads"))))))

```

```

(use-package flycheck-ledger
  :after (flycheck ledger-mode)
  :hook (ledger-mode . flycheck-mode)
  :demand t)

```

```

(with-eval-after-load 'ledger
  (defadvice ledger-accounts-list (around sacha activate)
    "Cache"
    (setq ad-return-value (or sacha-ledger-account-list-cache
                              (setq sacha-ledger-account-list-cache ad-do-it)))))

```

```

(defvar-local sacha-ledger-account-list-cache nil)

;;;###autoload
(defun sacha-ledger-set-unknown-account (account point)
  (interactive (list (ledger-read-account-with-prompt "Account") (point)))
  (let ((extents (ledger-navigate-find-xact-extents point)))
    (save-excursion
      (goto-char (car extents))
      (if (re-search-forward "Expenses:\\\\(Unknown\\\\|Play\\\\)" (cadr extents) t)
          (replace-match account t t)
          (goto-char point)
          (beginning-of-line)
          (when (re-search-forward "\\([^\t]+\)" (line-end-position) nil)
            (replace-match account t t 1)))))

;;;###autoload
(defun sacha-ledger-go-to-beginning-of-entry ()
  "Move to the beginning of the current entry."
  (while (and (not (bobp))

```

```

        (eq (ledger-context-line-type (ledger-context-at-point))
            'acct-transaction))
    (forward-line -1)))

;;;###autoload
(defun sachal-ledger-entry-date ()
  "Returns the date of the entry containing point or nil."
  (save-excursion
    (sachal-ledger-go-to-beginning-of-entry)
    (let ((context-info (ledger-context-other-line 0)))
      (when (eq (ledger-context-line-type context-info) 'entry)
        (goto-char (line-beginning-position))
        (if (looking-at "\\([-0-9\\.\\./]+\\)")
            (match-string-no-properties 1))))))

;;;###autoload
(defun sachal-ledger-guess-mbna ()
  "Adds a sub-account for the dates for my credit card transactions."
  (interactive)
  (save-excursion
    (sachal-ledger-go-to-beginning-of-entry)
    (forward-line 1)
    (let ((amount 0) (date (sachal-ledger-entry-date)) month)
      (if (string-match "[0-9]+[-\\.]\\([-0-9]+\\)[-\\.]\\([-0-9]+\\)" date)
          (setq month (string-to-number (match-string 1 date))))
      ;; Is this a payment or a charge?
      (save-excursion
        (while (and (eq (ledger-context-line-type (ledger-context-at-point))
                        'acct-transaction)
                     (not (eobp)))
          (let ((context (ledger-context-at-point)))
            (if (ledger-context-field-value context 'amount)
                (if (string-match "MBNA" (ledger-context-field-value context 'account))
                    (setq amount (string-to-number (ledger-context-field-value context 'amount)))
                    (setq amount (- (string-to-number (ledger-context-field-value context 'amount))
                                     ↪ 'amount))))))
            (forward-line 1)))
        (save-excursion
          (while (and (eq (ledger-context-line-type (ledger-context-at-point))
                          'acct-transaction)
                       (not (eobp)))
            (let ((context (ledger-context-at-point)))
              (if (string-match "MBNA" (ledger-context-field-value context 'account))
                  (if (re-search-forward "\\(MBNA\\)[ \\t]*[-$.0-9]*[ \\t]*$" (line-end-position) t)
                      (replace-match
                        (concat "MBNA:"
                                ↪ ("January" "February" "March" "April" "May" "June" "July" ↪
                                ↪ "August" "September" "October" "November" "December")
                                (% (+ (if (> amount 0) 10 11) month) 12)))
                      t t nil 1))))
              (forward-line 1))))))

;;;###autoload
(defun sachal-ledger-change-account (account)
  (interactive (list (ledger-read-account-with-prompt (concat (ledger-xact-payee) ": ")))
  (beginning-of-line)
  (re-search-forward ledger-account-name-or-directive-regex)
  (replace-match (concat " " account " ") t t))

```

```
;;###autoload
(defun sacha-ledger-fix-unknown ()
  (interactive)
  (while (re-search-forward "Expenses:Unknown.*$ \\(\\.+\\)" nil t)
    (sacha-ledger-change-account (ledger-read-account-with-prompt
                                  (format "%s %s: " (s-trim (save-match-data (ledger-xact-payee)))
                                            (match-string 1)))))))
```

```
;;###autoload
(defun sacha-latest-file (path &optional filter)
  "Return the newest file in PATH. Optionally filter by FILTER."
  (if (listp path)
      (car
       (sort (mapcar (lambda (dir) (sacha-latest-file dir)) path)
              #'file-newer-than-file-p))
      (car
       (sort (seq-remove #'file-directory-p
                        ↩ (directory-files path 'full filter t))
              #'file-newer-than-file-p)))) ↩
```

20.2 SSH and --daemon

From <https://github.com/nhoffman/.emacs.d/blob/master/init.org>

```
;;###autoload
(defun sacha-ssh-refresh ()
  "Reset the environment variable SSH_AUTH_SOCK"
  (interactive)
  (let (ssh-auth-sock-old (getenv "SSH_AUTH_SOCK"))
    (setenv "SSH_AUTH_SOCK"
            (car (split-string
                  (shell-command-to-string
                   "ls -t $(find /tmp/ssh-* -user $USER -name 'agent.*' 2> /dev/null)"))))
    (message
     (format "SSH_AUTH_SOCK %s --> %s"
             ssh-auth-sock-old (getenv "SSH_AUTH_SOCK")))))
```

```
(sacha-ssh-refresh)
```

20.3 Encryption

```
(setq epa-file-encrypt-to '("sacha@sachachua.com"))
(setq epa-pinentry-mode 'loopback)
(setq epg-pinentry-mode 'loopback)
```

20.4 Stardew Valley

```
;;###autoload
(defun sacha-stardew-install-mod (file)
  (interactive (list (read-file-name "Zip: " "~/Downloads/")))
```



```

(call-process "unzip"

nil (get-buffer-create ←
↳ "*mods*") nil

"-uo"
file
"-d" (expand-file-name ←
↳ "/home/sacha/.local/share/Steam/steamapps/common/Stardew Valley/Mods/")
(message "Installed %s" (file-name-base file))
)

;;;###autoload
(defun sacha-stardew-install-latest-mod ()
  (interactive)
  (sacha-stardew-install-mod (sacha-latest-file "~/Downloads")))

```

21 Other cool configs you may want to check out

- [Bernt Hansen](#): Lots of Org-related config. I picked up the graph-drawing stuff from this.
- [Bastien Guerry](#): Org, Gnus, ERC - Explained in this [Emacs Chat](#) (~1h)
- [Iannis Zannos](#): Explained in this [Emacs Chat](#) (~1h)
- [Magnar Sveen](#): <http://whattheemacs.com/> has some explanations. [Emacs Chat](#) (~1h)
- [John Wiegley](#): Also see his [Emacs Lisp Development](#) talk (sorry, sucky video) and [Emacs Chat](#) video

22 Inactive/infrequent things

22.1 Old Flickr/Evernote export

```

;; I don't use these as much now that I have the functions above.
(defun sacha-evernote-extract-links (filename)
  "Extract note names and URLs from an ENEX file."
  (interactive)

  (goto-char (point-min))
  (let (list)
    (while (re-search-forward "<title>\\(\\.+?\\)</title>\\(\\.+?\\n\\)*?.*?href=\\\"\\(\\.+?\\)\\\"\" nil t)
      (setq list (cons (cons (match-string-no-properties 1) (match-string-no-properties 3)) ←
↳ list)))
    (delete-region (point-min) (point-max))
    (insert (mapconcat (lambda (x) (concat "- [" (cdr x) "]" [" (car x) "]" )) list "\n"))))

(defun sacha-flickr-extract-this-week ()
  "Extract this week's sketch titles and URLs from the flickr_metadata CSV."
  (interactive)
  (let ((base-date (apply 'encode-time (org-read-date-analyze "-fri" nil '(0 0 0))))
        start end list)
    (setq start (format-time-string "%Y-%m-%d" (days-to-time (- (time-to-number-of-days ←
↳ base-date) 6))))
    (setq end (format-time-string "%Y-%m-%d" (days-to-time (1+ (time-to-number-of-days ←
↳ base-date)))))
    (setq list (csv-parse-buffer t))

```

```

(erase-buffer)
(insert
  (mapconcat (lambda (x) (concat "- [" (car x) "]" [" (cdr x) "]")))
  (sort
    (delq nil
      (mapcar (lambda (x)
        (let ((title (cdr (assoc "FileName" x))))
          (if (and (not (string< title start))
              (string< title end))
              (cons (cdr (assoc "URL" x)) title))))
        list))
    (lambda (a b) (string< (cdr a) (cdr b)))
    )
  "\n"))))

```

22.2 Animation for Emacs chats

video:animation

```

;; ##autoload
(defun sachachua-animate-emacs-chat ()
  (interactive)
  (text-scale-set 6)
  (erase-buffer)
  (sit-for 3)
  (let ((list '("Emacs Chat: Sacha Chua"
    "interviewed by Bastien Guerry"
    ""
    "July 24, 2013"
    "sachachua.com/emacs-chat")))
    (approx-width 41)
    (approx-height 16)
    row)
  (setq row (/ (- approx-height (length list)) 2))
  (mapcar
    (lambda (x)
      (animate-string x
        row
        (/ (- approx-width (length x)) 2))
      (setq row (1+ row)))
    list)))

```

22.3 Oddmuse

```

(use-package oddmuse
  :if sachalaptop-p
  :load-path "~/vendor/oddmuse-el"
  :ensure nil
  :config (oddmuse-mode-initialize)
  :commands oddmuse-edit
  :hook (oddmuse-mode-hook .
    (lambda ()
      (unless (string-match "question" oddmuse-post)
        (when (string-match "EmacsWiki" oddmuse-wiki)
          (setq oddmuse-post (concat "uihnscksc=1;" oddmuse-post)))
        (when (string-match "OddmuseWiki" oddmuse-wiki)
          (setq oddmuse-post (concat "ham=1;" oddmuse-post)))))))

```

22.4 Beeminder

<https://github.com/sachac/beeminder.el>

This bit of code lets me track sent messages in Gnus:

```
(defun sach-beeminder-track-message ()
  (save-excursion
    (goto-char (point-min))
    (when (re-search-forward "Newsgroups: .*emacs")
      (goto-char (point-min))
      (when (re-search-forward "Subject: \\(.*\\)" nil t)
        (beeminder-add-data "orgml" "1" (match-string 1))))))
```

And this loads the beeminder code:

```
(use-package beeminder
  :disabled t
  :config (add-hook 'message-send-news-hook 'sach-beeminder-track-message))
```

22.5 Plover

<https://github.com/sachac/plover-websocket-el>

```
(use-package plover-websocket
  :load-path "~/proj/plover-websocket-el"
  :after websocket
  :if sach-laptop-p
  :defer t
  :config (setq plover-websocket-plover-command "cd ~/vendor/plover; tox -e launch")
  :hydra
  (sach-plover (:exit t)
    ("<f1>" plover-websocket-connect "Open websocket")
    ("<f2>" plover-websocket-add-translation "Add translation")
    ("<f3>" plover-websocket-lookup "Lookup")
    ("<f4>" plover-websocket-configure "Configure")
    ("<f5>" plover-websocket-focus "Focus")
    ("<f6>" plover-websocket-toggle-plover "Toggle Plover")
    ("<f7>" plover-websocket-quit "Quit")
    ("<f8>" sach-plover-drilling-time "Drill"))
  :bind
  ("<f6>" . #'sach-plover/body))
```

22.5.1 Looking things up

```
;; ##autoload
(defun sach-plover-search-dictionary-for-strokes-jq (stroke-regexp)
  (json-parse-string
    (shell-command-to-string
      (format "cat ~/.config/plover/main.json | jq 'with_entries(if (.key|test(\"%s\")) then ( ↵
        ↵ {key: .key, value: .value}) else empty end)'"
        stroke-regexp))
    :object-type 'alist))
(defvar sach-plover-main-dict
  (if (and sach-laptop-p (file-exists-p "~/config/plover/main.json"))
```

```

        (mapcar (lambda (o) (cons (symbol-name (car o)) (cdr o)))
          (json-read-file "~/.config/plover/main.json"))))

;; ###autoload
(defun sacha-plover-search-dictionary-for-strokes (stroke-regexp)
  (interactive "MStroke regexp: ")
  (let ((results (seq-filter (lambda (o) (string-match stroke-regexp (car o))) ←
    ↪ sacha-plover-main-dict)))
    (when (called-interactively-p 'any) (sacha-plover-display-dictionary-results results)
      results))
  (defvar sacha-plover-dict-cache nil "Alist of (filename . ((stroke . translation) ...))")
  (defvar sacha-plover-home "~/.config/plover")

;; ###autoload
(defun sacha-plover-dict (&optional filename)
  (setq filename (expand-file-name (or filename "main.json") sacha-plover-home))
  (or (cdr (assoc-default filename sacha-plover-dict-cache))
    (let ((result (mapcar (lambda (o) (cons (symbol-name (car o)) (cdr o))) ←
      ↪ (json-read-file filename))))
      (push (cons filename result) sacha-plover-dict-cache )
      result)))

;; ###autoload
(defun sacha-plover-search-dictionary-for-translation (translation &optional start file)
  (interactive "MTranslation: \nP")
  (let* ((regexp (concat "^" (regexp-quote translation) (unless start "$")))
    (results (seq-filter (lambda (o) (string-match regexp (cdr o))) (sacha-plover-dict ←
      ↪ file))))
    (when (called-interactively-p 'any) (sacha-plover-display-dictionary-results results)
      results))

;; ###autoload
(defun sacha-plover-display-dictionary-results (results)
  (with-current-buffer (get-buffer-create "*Plover*")
    (erase-buffer)
    (insert (format "%d entries\n" (length results))
      (mapconcat (lambda (o) (format "%s\t%s" (car o) (cdr o))) results "\n"))
    (goto-char (point-min))
    (display-buffer (current-buffer)))

  (defmacro sacha-with-plover-fingerspelling (&rest body)
    `(progn
      (plover-websocket-send :translation ←
        ↪ "{PLOVER:SOLO_DICT:+commands.json,+fingerspelling.json}")
      (progn (progn ,@body)
        (plover-websocket-send :translation "{PLOVER:END_SOLO_DICT}"))))

;; ###autoload
(defun sacha-consult-plover-read-stroke-or-translation ()
  (interactive)
  (let ((dict (mapcar (lambda (o) (cons (format "%s: %s" (car o) (cdr o)) o))
    (sacha-plover-dict)))
    (sacha-with-plover-fingerspelling
      (consult--read
        dict
        :prompt "Strokes/translation: "
        :category 'plover-stroke))))

;; ###autoload
(defun sacha-consult-plover-and-execute-strokes (choice)

```

```

(interactive (list (sacha-consult-plover-read-stroke-or-translation)))
(when (string-match "\\([^\ ]+\\): \\(\\.+\\)" choice)
  (plover-websocket-send :translation (match-string 2 choice) :force t ↵
    ↵ :zero_last_stroke_length t)))

;; ###autoload
(defun sacha-consult-plover-search-strokes (regexp solo-p)
  (interactive (list (with-plover-plain (read-string "Strokes: ")) current-prefix-arg))
  (consult--read
    (mapcar (lambda (o) (cons (format "%s: %s" (car o) (cdr o)) o))
      (sacha-plover-search-dictionary-for-strokes (if solo-p (concat "^" regexp ↵
        ↵ "\\(?:/\\|\\$\\)" ) (concat "^" regexp))))
    :prompt "Narrow: "))

;; (list
;; (benchmark-run 2 (sacha-plover-search-dictionary-for-strokes-jq "^THER"))
;; (benchmark-run 2 (sacha-plover-search-dictionary-for-translation "stenography" t ↵
  ↵ "typey-type.json")
;; (benchmark-run 2 (sacha-plover-search-dictionary-for-translation "stenography" t))
;; (benchmark-run 2 (sacha-plover-search-dictionary-for-strokes "^THER/"))))

```

22.5.2 Adding steno hints as I type

steno:emacs

When I type with steno, I want to see little hints. I borrowed some code from company-posframe to display hints based on the last few words, even ones I ended up fingerspelling or typing on my keyboard. This makes it easier to learn new words if I have to spell them out. There's probably a better way to do it, but this is a good start.

```

(defvar sacha-steno-hint-dict nil)
(defvar sacha-steno-hint-dictionaries
  '("~/config/plover/user.json"
    "~/vendor/steno-dictionaries/dictionaries/dict.json"))
(defvar sacha-steno-hint-buffer " *steno hint*")

;; ###autoload
(defun sacha-steno-hint-load-dictionary ()
  (interactive)
  (setq sacha-steno-hint-dict
    (seq-mapcat
      (lambda (filename)
        (with-temp-buffer
          (insert-file-contents filename)
          (goto-char (point-min))
          (json-parse-buffer :object-type 'alist)))
      sacha-steno-hint-dictionaries)))

;; ###autoload
(defun sacha-steno-hint-lookup (search)
  (let ((search-list (list search (downcase search))))
    (seq-group-by
      'cdr
      (seq-filter
        (lambda (entry)
          (member (cdr entry) search-list))
        sacha-steno-hint-dict))))

```

When I type with steno, I want to see little hints. I borrowed some code from company-posframe to display hints based on the last few words, even ones I ended up fingerspelling or typing on my keyboard. This makes it easier to learn new words if I have to spell them out. There's probably a better way to do it, but this is a good start.

```

#+CAPTION: This is how the
how the: HO*UT, HOUT
the: -T, -T/S-P
S T P H F P L T D
S K W R R B G S Z
A O E U
#+begin_src emacs-lisp
(defun my-steno-hint-dict
  (defvar my-steno-hint-dictionaries
    ("~/config/plover/user.json"
     "~/vendor/steno-dictionaries/dictionaries/dict.json"))
  (defvar my-steno-hint-buffer " *steno hint*")

  (defun my-steno-hint-load-dictionary ()
    (interactive)
    (setq my-steno-hint-dict
      (seq-mapcat
        (lambda (filename)

```

Figure 28: This is how the hint appears

```

;;###autoload
(defun sacha-steno-hint-find (&optional buffer)
  "Return a steno hint for the last 1-4 words, if any."
  (setq buffer (or buffer (current-buffer)))
  (when (buffer-live-p buffer)
    (with-current-buffer buffer
      (let ((pos (point)) result hint)
        (save-excursion
          (dotimes (i 4)
            (backward-word)
            (setq result
                    (cons
                     (sacha-steno-hint-lookup
                      (string-trim ←
                        (buffer-substring-no-properties (point) pos)))
                     result))))
          (delq nil result))))))

(defun sacha-steno-hint-display-functions '(sacha-steno-hint-show-posframe))

;;###autoload
(defun sacha-steno-hint-show-posframe (result &optional command)
  (if (and result (or (null command)
                      ←
                        (member command '(self-insert-command org-self-insert-command))))
      (progn
        (with-current-buffer (get-buffer-create sacha-steno-hint-buffer)
          (erase-buffer)

```

```

                                (insert
                                (propertize
                                (mapconcat
                                (lambda (entries)
                                (mapconcat
                                (lambda (entry)
                                (concat
                                (car entry) ": "
                                (mapconcat (lambda (entry)
↪ (stroke)
                                (symbol-name (car stroke)))
↪
                                (cdr entry) ", ")))
                                entries "\n"))
                                result "\n"
                                'face 'lispy-face-hint)
                                "\n"
                                (mapconcat 'sacha-steno-hint-propertized-layout
↪ (split-string (symbol-name (car (cadar (car result)))) "/")
↪
↪ "\n\n"))
                                (posframe-show sacha-steno-hint-buffer :position (point)
↪ :border-width 1))
                                (posframe-hide sacha-steno-hint-buffer)))

(defun sacha-steno-hint--timer nil)

;;;###autoload
(defun sacha-steno-hint-recent-when-idle ()
  (interactive)
  (when (timerp sacha-steno-hint--timer)
    (cancel-timer sacha-steno-hint--timer))
  (setq sacha-steno-hint--timer
        (run-with-idle-timer 0.1 nil #'sacha-steno-hint-recent
↪ (current-buffer) this-command)))

;;;###autoload
(defun sacha-steno-hint-recent (buffer command)
  (interactive)
  (setq sacha-steno-hint--timer nil)
  (run-hook-with-args 'sacha-steno-hint-display-functions (sacha-steno-hint-find buffer)
↪ command))

;;;###autoload
(defun sacha-steno-split-keys (s)
  "Return a list of individual steno keys for RTFCRE."
  (when (string-match "\\([STKPWHR]*\\)\\(-\\|\\|\\([AOEU*]+\\)\\)\\([FRPBLGTSDZ]*\\)" s)
    (append
      (mapcar (lambda (ch) (format "%s-" (char-to-string ch))) (match-string 1 s))
      (mapcar 'char-to-string (match-string 3 s))
      (mapcar (lambda (ch) (format "-%s" (char-to-string ch))) (match-string 4 s))))))

;; (sacha-steno-split-keys "HR-")
;; (sacha-steno-split-keys "HRAEUT")
;; (sacha-steno-split-keys "HR*T")

;;;###autoload
(defun sacha-steno-hint-propertized-layout (s)

```



```
(posframe-delete sachasteno-hint-buffer))))
```

22.5.3 Running Plover drills from Emacs

I'm learning stenography because I deal with a lot of text, and it seems interesting. I'd like to someday be able to live-caption EmacsConf, meetups, and other technical things. I've got a lot of muscle memory to pick up, which means drills drills drills drills.

```
(defvar sachaplover-drills
  (append
    (mapcar (lambda (desc)
      (cons desc (concat ↵
        ↵ "https://joshuagrams.github.io/steno-jig/learn-keyboard.html?drill=" (url-encode-url ↵
        ↵ (replace-regexp-in-string "\\+" "%2B" desc))))))
      '("Left hand, bottom row"
        "Right hand, bottom row"
        "Left hand, top row"
        "Right hand, top row"
        "Right hand, full bottom row"
        "Right hand, full top row"
        "Vowels"
        "Left hand"
        "Right hand"
        "All keys"
        "Left + Right"
        "Left + Vowel"
        "Vowel + Right"
        "Left + Vowel + Right"
        "Columns: D, B, L, -N"
        "x")))
    (mapcar (lambda (desc)
      (cons desc (concat ↵
        ↵ "https://joshuagrams.github.io/steno-jig/learn-plover.html?hints=yes&type=randomly&timeLimit=26&drill=" ↵
        ↵ (url-encode-url (replace-regexp-in-string "\\+" "%2B" desc))))))
      '("One Syllable Words" "Consonant Clusters" "Where's the TRUFT?" "Dropping ↵
        ↵ Unstressed Vowels" "Inversion" "The Fifth Vowel Key" "Long Vowel Chords" "Diphthong ↵
        ↵ Chords" "Vowel Disambiguator Chords" "The Missing Keys" "The Remaining Missing ↵
        ↵ Letters" "Review Through Missing Letters" "Digraphs" "Review Through Digraphs" "Common ↵
        ↵ Compound Clusters" "Review Through Common Compound Clusters" "Common Briefs 1-20" ↵
        ↵ "Common Briefs 21-40" "Common Briefs 41-60" "Common Briefs 61-80" "Common Briefs ↵
        ↵ 81-100")))))

(defvar sachaplover-drill-history nil "Previous drills")
(defvar sachaplover-drill-file "~/proj/plover-notes/README.org")

;; ##autoload
(defun sachaplover-stenojig-custom-drill (words)
  (interactive "MWords: ")
  (plover-websocket-resume-plover)
  (unwind-protect
    (progn
      (browse-url-chrome (concat ↵
        ↵ "file:///home/sacha/vendor/steno-jig/from-url.html?go=true&type=randomly&timeLimit=2&name=test&hints=true&6
        ↵ (url-encode-url words))))
      (read-string "Ignore this: "))
    (plover-websocket-suspend-plover)))
```

[illegible]

22.5.4 Making it easier to execute commands

```
(setq enable-recursive-minibuffers t)
```

```
;;;###autoload
(defun sacha-org-replace-heading (new-text)
  (interactive (list (read-string (concat (org-get-heading t t t t) ": "))))
  (org-back-to-heading)
  (when (looking-at org-complex-heading-regexp)
    (replace-match new-text t t nil 4)))
```

```
;;;###autoload
(defun sacha-plover-process-inbox-entries ()
  (interactive)
  (catch 'exit
    (while t
      (plover-websocket-send :stroke '["K-" "P-" "A-" "*"])
      (sacha-read-command-string
        (lambda () (concat (org-get-heading t t t t) ": ")))
      '(("replace and post"
        (lambda () (interactive)
          (call-interactively 'sacha-org-replace-heading)
          (call-interactively 'sacha-org-mark-done-and-add-to-journal)
          (org-forward-heading-same-level 1)))
        ("edit" sacha-org-replace-heading)
        ("post" sacha-org-mark-done-and-add-to-journal)
        ("refile" org-refile)
        ("to do" org-todo)
        ("next" org-forward-heading-same-level)
        ("open link" (lambda () (interactive)
          (save-excursion
            (when (re-search-forward org-link-any-re nil t)
              (goto-char (match-beginning 0))
              (org-open-at-point))))))
        ("yesterday" (lambda () (interactive)
          (save-excursion
            (re-search-forward org-element--timestamp-regexp)
            (goto-char (match-beginning 0))
            (org-timestamp-down-day))))
        ("previous" org-backward-heading-same-level)
        ("new journal" sacha-journal-post)
        ("practice" (lambda () (interactive) (quantified-track "steno") (browse-url ↵
↵ "https://didoesdigital.com/typey-type/progress"))))
        ("lowercase" downcase-word)
        ("capitalize" capitalize-dwim)
        ("clean" sacha-org-clean-up-inbox)
        ("replace heading" sacha-org-replace-heading)
        ("cut subtree" org-cut-subtree)
        ("export subtree to 11ty" (lambda () (interactive) (org-11ty-export-to-11ty t t)))
        ("exit" (throw 'exit nil)))
      (lambda (input)
        (sacha-org-replace-heading input)
        (call-interactively 'sacha-org-mark-done-and-add-to-journal)
        (org-forward-heading-same-level 1))
      t))))

(defmacro sacha-read-command-string (prompt commands default-fn &optional include-commands)
  (declare (debug t))
  `(let* ((command
```

```

      (consult--read
        (append ,commands
          (if ,include-commands
            (let (res)
              (mapatoms
                (lambda (o)
                  (when (commandp o) (push (symbol-name o) res))))
              res)))
        :prompt (cond
          ((functionp ,prompt) (funcall ,prompt))
          ((stringp ,prompt) ,prompt)
          (t "Command: "))
        :category 'function
        :sort nil))
      (entry (assoc-default command ,commands)))
    (cond
      ((and entry (listp (car entry)))
        (if (functionp (car entry))
          (funcall (car entry))
          (eval (car entry) t)))
      (entry (call-interactively (car entry)))
      ((commandp (intern command)) (call-interactively (intern command)))
      ((functionp ,default-fn) (funcall ,default-fn command))))))

;;;###autoload
(defun sacha-read-commands ()
  (interactive)
  (cond
    ((derived-mode-p 'org-mode)
     (sacha-plover-process-inbox-entries))
    ((derived-mode-p 'subed-mode)
     (sacha-plover/edit-subtitles))))

```

22.5.5 Suggesting briefs

Only checks one dictionary for now, but probably good enough

```

;;;###autoload
(defun sacha-plover-briefpedia (translation)
  (interactive "MTranslation: ")
  (with-current-buffer (url-retrieve-synchronously (concat ↵
    ↵ "http://briefpedia.com/AjaxTables3.php?search=" (url-encode-url translation)))
    (goto-char (point-min))
    (re-search-forward "^$")
    (while (re-search-forward "</?\\(th\\)[ >]" nil t)
      (replace-match "td" nil nil nil 1))
      (goto-char (point-min))
    (re-search-forward "^$")
    (save-excursion
      (insert "<div>")
      (goto-char (point-max)) (insert "</div>"))
    (let* ((data (xml-parse-region (point-min) (point-max)))
      (entries (mapcar (lambda (o) (string-trim (dom-text o))) (dom-by-tag (dom-by-id ↵
    ↵ data "divEnglishTable") 'a)))
      (conflicts (seq-group-by 'car
        (mapcar (lambda (row) (mapcar (lambda (cell) (string-trim ↵
    ↵ (dom-texts cell))) (dom-by-tag row 'td)))

```

```

                                (cdr (dom-by-tag (dom-by-id data "divCrossTable") ←
↪ 'tr))))))
                                (result
                                (mapcar (lambda (entry) (cons entry (mapcar ←
↪ 'cadr (assoc-default entry conflicts)))) entries)))
                                (when (called-interactively-p 'any)
                                (message "%s"
                                (mapconcat (lambda (entry)
↪
                                (concat (car entry)
↪
                                (if (cdr entry)
↪
↪
↪
↪ (concat " ("
↪
↪
                                (string-join (cdr entry) ", ")
↪
↪
                                ")")
↪
↪
                                ""))))
↪
                                result
↪
                                "; ")))
                                result)))

;;;###autoload
(defun sacha-plover-read-outline-for-brief (base-prompt)
  (let* ((prompt (or base-prompt "Outline: "))
         new-brief
         (brief (with-plover-plain (read-string prompt)))
         (sacha-conflicts (sacha-plover-check-for-conflict brief)))
    (while sacha-conflicts
      (setq prompt (format "%s%s conflicts %s (alt: %s): "
                           (if base-prompt (concat base-prompt "\n") "")
                           brief (car sacha-conflicts) (string-join (cdr sacha-conflicts) ", ")))
      (setq new-brief (with-plover-plain (read-string prompt)))
      (if (string= new-brief "")
          (setq sacha-conflicts nil)
          (setq brief new-brief)
          (setq sacha-conflicts (sacha-plover-check-for-conflict brief))))
    brief))

;;;###autoload
(defun sacha-plover-brief-with-check (translation)
  (interactive "MTranslation: ")
  (setq translation (string-trim translation))
  (let ((brief (sacha-plover-read-outline-for-brief (format "Outline for %s: " translation))))
    (when brief
      (kill-new (format "| %s | %s |" brief translation))
      (plover-websocket-add-translation brief translation))))

;;;###autoload
(defun sacha-plover-briefpedia-suggest (translation)
  (interactive "MTranslation: ")
  (setq translation (string-trim translation))

```

```

(let* ((entries (sacha-plover-briefpedia translation))
      (current (sacha-plover-search-dictionary-for-translation translation))
      (brief
       (sacha-plover-read-outline-for-brief
        (concat
         (if current (format "Current: %s\n" (mapconcat 'car current "; ")) "")
         (if entries
          (concat (mapconcat
                   (lambda (entry)
                     (let ((dict-conflict (sacha-plover-check-for-conflict (car entry))))
                       (cond
                        ((and (cdr entry) dict-conflict)
                         (format "%s - dict conflict: %s (%s)\nbrief conflict: %s"
                                (car entry)
                                (car dict-conflict)
                                (string-join (cdr dict-conflict) "; ")
                                (string-join (cdr entry) "; ")))
                        ((cdr entry)
                         (format "%s - brief conflict: %s"
                                (car entry)
                                (string-join (cdr entry) "; ")))
                        (t (car entry))))
                     entries
                     "\n")
                 "\nOutline: ")
          "No suggestions. Outline: "))))))
      (when brief
        (kill-new (format "| %s | %s |" brief translation))
        (plover-websocket-add-translation brief translation))))

;; ###autoload
(defun sacha-plover-check-for-conflict (outline)
  (let* ((case-fold-search nil)
        (translation (cdar (sacha-plover-search-dictionary-for-strokes (concat "^" outline
        ↪ "$"))))
        (alternatives (and translation (sacha-plover-search-dictionary-for-translation ↪
        ↪ translation))))
    (if translation (cons translation (mapcar 'car alternatives)))))

```

22.5.6 Practising within Emacs

Main function: `M-x sacha-practise-steno`, called in an Org table of | translation | outline |

```

;; ###autoload
(defun sacha-practise-steno-interleave (base item)
  "Interleave BASE words with item."
  (cons item
        (-interleave base (make-list (length base) item))))
;; Copied from elfeed--shuffle
;; ###autoload
(defun sacha-practise-steno-shuffle (seq)
  "Destructively shuffle SEQ."
  (let ((n (length seq)))
    (progn seq
           (dotimes (i n)
            (cl-rotatef (elt seq i) (elt seq (+ i (cl-random (- n i))))))))))
;; ###autoload
(defun sacha-practise-steno-repeat (seq times)

```

```

(funcall 'append (make-list times seq)))
(defface sacha-practise-steno-correct '((t :foreground "green")) "Correct.")
(defface sacha-practise-steno-wrong '((t :foreground "red")) "Wrong.")
(defface sacha-practise-steno-highlight '((t :background "white" :foreground "black")) "Focus.")
(defface sacha-practise-steno-base '((t :height 150)) "Base.")
(defvar sacha-practise-steno-items nil)
(defvar sacha-practise-steno-index 0)
(defvar sacha-practise-steno-buffer-name "*Steno practice*")
(defvar sacha-practise-steno-start-of-input nil)
(defvar sacha-practise-steno-current-overlay nil)
(defvar sacha-practise-steno-previous-overlay nil)
(defvar sacha-practise-steno-highlight-overlay nil)
(defvar sacha-practise-steno-stroke-buffer nil)
(defvar sacha-practise-steno-for-review nil)

;; From ↵
↵ https://stackoverflow.com/questions/1249497/command-to-center-screen-horizontally-around-cursor-on-emacs
;; ###autoload
(defun sacha-horizontal-recenter ()
  "Make the point horizontally centered in the window."
  (interactive)
  (let ((mid (/ (window-width) 2))
        (pixel-pos (car (window-absolute-pixel-position)))
        (pixel-mid (/ (window-pixel-width) 2))
        (line-len (save-excursion (end-of-line) (current-column)))
        (cur (current-column)))
    (while (< pixel-mid pixel-pos)
      (set-window-hscroll (selected-window)
                          (1+ (window-hscroll)))
      (setq pixel-pos (car (window-absolute-pixel-position)))))

;; ###autoload
(defun sacha-practise-steno--handle-correct ()
  (if sacha-practise-steno-previous-overlay
      (move-overlay sacha-practise-steno-previous-overlay (overlay-start ↵
                                                            ↵ sacha-practise-steno-previous-overlay)
                  (+ (overlay-end sacha-practise-steno-previous-overlay) (match-end 0)))
      (setq sacha-practise-steno-previous-overlay
            (make-overlay (overlay-end sacha-practise-steno-previous-overlay)
                          (+ (overlay-end sacha-practise-steno-previous-overlay) (match-end 0))))
      (overlay-put sacha-practise-steno-previous-overlay 'evaporate t)
      (overlay-put sacha-practise-steno-previous-overlay 'face 'sacha-practise-steno-correct)))

;; ###autoload
(defun sacha-practise-steno--mark-incorrect-and-fixed ()
  (let ((ov (make-overlay (overlay-end sacha-practise-steno-previous-overlay)
                          (+ (overlay-end sacha-practise-steno-previous-overlay) ↵
                              ↵ (match-beginning 0)))))
    (overlay-put ov 'face 'sacha-practise-steno-wrong)
    (overlay-put ov 'evaporate t))
  ;; make a new overlay
  (setq sacha-practise-steno-previous-overlay (copy-overlay ↵
                                                    ↵ sacha-practise-steno-previous-overlay))
  (move-overlay sacha-practise-steno-previous-overlay
                (+ (overlay-end sacha-practise-steno-previous-overlay) (match-beginning 0))
                (+ (overlay-end sacha-practise-steno-previous-overlay) (match-end 0)))
  (setq sacha-practise-steno-for-review (append sacha-practise-steno-for-review (list (elt ↵
                                                                                          ↵ sacha-practise-steno-items sacha-practise-steno-index))))
  ;; highlight the sample as incorrect, too

```

```

(let ((incorrect-sample (copy-overlay sach-practise-steno-highlight-overlay)))
  (overlay-put incorrect-sample 'face 'sach-practise-steno-wrong)
  (save-excursion
    (goto-char (overlay-start sach-practise-steno-highlight-overlay))
    (insert (make-string
              (+
               (if (bolp) 1 0)
               (match-beginning 0))
              ?\ )))))

;;;###autoload
(defun sach-practise-steno--move-to-next-item ()
  (setq sach-practise-steno-stroke-buffer nil)
  (setq sach-practise-steno-index (1+ sach-practise-steno-index))
  (move-overlay sach-practise-steno-current-overlay (overlay-end ←
    ↪ sach-practise-steno-previous-overlay) (point))
  (if (elt sach-practise-steno-items sach-practise-steno-index)
      (move-overlay sach-practise-steno-highlight-overlay
                    (1+ (overlay-end sach-practise-steno-highlight-overlay))
                    (+ (overlay-end sach-practise-steno-highlight-overlay)
                       1 (length (car (elt sach-practise-steno-items ←
    ↪ sach-practise-steno-index))))))
      (when sach-practise-steno-for-review
        (goto-char (point-max))
        (kill-new (mapconcat 'car sach-practise-steno-for-review " "))
        (insert "\nFor review: " (mapconcat 'car sach-practise-steno-for-review " "))))))

;;;###autoload
(defun sach-practise-steno--handle-completed-item ()
  ;; extend the feedback overlay to the current point
  (if (= (match-beginning 0) 0)
      (sach-practise-steno--handle-correct)
      ;; mark incorrect area
      (sach-practise-steno--mark-incorrect-and-fixed))
  (sach-practise-steno--move-to-next-item))

;;;###autoload
(defun sach-practise-steno-check (&rest _)
  (interactive)
  (let* ((sample (car (elt sach-practise-steno-items sach-practise-steno-index)))
        (input (and (< (overlay-end sach-practise-steno-previous-overlay) (point))
                     (buffer-substring-no-properties (overlay-end ←
    ↪ sach-practise-steno-previous-overlay) (point)))))
    (when (and sample input)
      (if (string-match (concat " *" (regexp-quote sample) " *") input)
          (sach-practise-steno--handle-completed-item)
          ;; still in progress
          (move-overlay sach-practise-steno-current-overlay
                        (overlay-start sach-practise-steno-current-overlay)
                        (1+ (point))))
      (sach-horizontal-recenter))))

;;;###autoload
(defun sach-practise-steno-store-strokes (payload)
  (when (and (plist-get payload :stoked) (string= (buffer-name) ←
    ↪ sach-practise-steno-buffer-name))
    (let ((current-item (elt sach-practise-steno-items sach-practise-steno-index))
          (rtfcre (plist-get (plist-get payload :stoked) :rtfcre)))
      (save-excursion

```



```

(goto-char (point-max))
(insert (if (bolp) " " " ") rtfcre))
(when (and (cadr current-item)
  (> (- (overlay-end sachapracsteno-current-overlay)
    (overlay-start sachapracsteno-current-overlay))
    (length (car current-item))))
  (setq sachapracsteno-stroke-buffer (append sachapracsteno-stroke-buffer ←
    (list rtfcre)))
  (momentary-string-display (format " (%s -> %s)"
    (string-join sachapracsteno-stroke-buffer " ")
    (cadr current-item))
    (point)
    ?\0
    ""))))))

;;;###autoload
(defun sachapracsteno (items)
  "Display ITEMS for practicing.
ITEMS should be a list like ((word) (word) (word))."
  (interactive (list (let ((table (org-table-to-lisp)))
    (if table
      (if current-prefix-arg
        (subseq table
          (1- (org-table-current-line))
          (min (length table) (+ (org-table-current-line) ←
            current-prefix-arg -1))))
      table)
    sachapracsteno-items))))
  (with-current-buffer (get-buffer-create sachapracsteno-buffer-name)
    (erase-buffer)
    (insert "\n" (mapconcat 'car items " " ) "\n")
    (save-excursion (insert "\n\n"))
    (toggle-truncate-lines 1)
    (setq sachapracsteno-items items
      sachapracsteno-index 0
      sachapracsteno-start-of-input (point)
      sachapracsteno-for-review nil
      sachapracsteno-current-overlay (make-overlay (point) (1+ (point))))
      sachapracsteno-previous-overlay (make-overlay (point) (point))
      sachapracsteno-stroke-buffer nil
      sachapracsteno-highlight-overlay (make-overlay (1+ (point-min)) (+ 1 ←
        (point-min) (length (car (car items))))))
    (buffer-face-set "sachapracsteno-base")
    (overlay-put sachapracsteno-previous-overlay 'face 'sachapracsteno-correct)
    (overlay-put sachapracsteno-previous-overlay 'evaporate t)
    (overlay-put sachapracsteno-highlight-overlay 'face 'sachapracsteno-highlight)
    (overlay-put sachapracsteno-highlight-overlay 'evaporate t)
    (overlay-put sachapracsteno-current-overlay 'modification-hooks ←
      '(sachapracsteno-check))
    (overlay-put sachapracsteno-current-overlay 'insert-in-front-hooks ←
      '(sachapracsteno-check))
    (overlay-put sachapracsteno-current-overlay 'face 'sachapracsteno-wrong)
    (overlay-put sachapracsteno-current-overlay 'evaporate t)
    ;; (add-hook 'after-change-functions 'sachapracsteno-check nil t)
    (add-hook 'plover-websocket-on-message-payload-functions 'sachapracsteno-store-strokes)
    (switch-to-buffer (current-buffer))))

;;;###autoload
(defun sachapracsteno-word-list (words)

```

```

(interactive (list (mapcar 'list (split-string (read-string "Words: "))))
(sacha-practise-steno words))

;; (call-interactively 'sacha-practise-steno)

```

22.5.7 Editing subtitles

```

;;;###autoload
(defun sacha-subed-subtitle-set-text (text)
  (interactive "MNew text: ")
  (subed-jump-to-subtitle-text)
  (delete-region (point) (or (subed-jump-to-subtitle-end) (point)))
  (insert text))

;;;###autoload
(defun sacha-plover/edit-find-target (input)
  (or (looking-at (concat "\\b" (regexp-quote input) "\\b"))
      (re-search-forward (concat "\\b" (regexp-quote input) "\\b")
                        nil t)))

;;;###autoload
(defun sacha-plover/edit-subtitles ()
  (interactive)
  (catch 'exit
    (while t
      (sacha-read-command-string
       "Command: "
       '(("toggle" subed-mpv-toggle-pause)
         ("jump" (lambda () (interactive) (subed-mpv-jump-to-current-subtitle)))
         ("split [text before split]" subed-split-subtitle)
         ("center" recenter-top-bottom)
         ("previous" (lambda () (interactive) (subed-merge-with-previous) (fill-paragraph)))
         ("merge next" (lambda () (interactive) (subed-merge-with-next) (fill-paragraph)))
         ("slow" (lambda () (interactive) (subed-mpv-playback-speed 0.5)))
         ("fast" (lambda () (interactive) (subed-mpv-playback-speed 2)))
         ("scroll" scroll-up-command)
         ("fill" fill-paragraph)
         ("next [text]" search-forward)
         ("replace <text>")
         ("previous [text]" search-backward)
         ("cap [text]" capitalize-word)
         ("delete [text]" kill-word)
         ("", [text] (lambda () (interactive) (insert ",")))
         ("end [text] - adds period and capitalizes next word" (lambda () (interactive) ↵
         ↵ (insert ".") (capitalize-word 1)))
         ("oops" 'undo)
         ("exit" (throw 'exit nil))))
      (lambda (input)
        (cond
         ((string-match "^split \\(.+\\) *$" input)
          (when (sacha-plover/edit-find-target (match-string 1 input))
            (goto-char (match-end 0))
            (subed-split-subtitle)
            (fill-paragraph)))
         ((string-match "^delete \\(.+\\) *$" input)
          (when (sacha-plover/edit-find-target (match-string 1 input))
            (replace-match "")))

```

```

((string-match "^, \\(\\.+\\) *$" input)
 (when (sacha-plover/edit-find-target (match-string 1 input))
  (goto-char (match-end 0))
  (insert ",")))
((string-match "^end \\(\\.+\\) *$" input)
 (when (sacha-plover/edit-find-target (match-string 1 input))
  (goto-char (match-end 0))
  (insert ".")
  (unless (save-excursion (subed-jump-to-subtitle-end))
   (subed-forward-subtitle-text))
  (capitalize-word 1)))
((string-match "^zap \\(\\.+\\) $" input)
 (delete-region (point)
  (sacha-plover/edit-find-target (match-string 1 input))))
((string-match "^replace \\(\\.+\\) $" input)
 (kill-word 1)
 (insert (match-string 1 input)))
((string-match "^cap \\(\\.+\\) *$" input)
 (when (sacha-plover/edit-find-target (match-string 1 input))
  (replace-match (capitalize (match-string 0)) t t)))
((string-match "^... \\(\\.+\\) *$" input)
 (when (sacha-plover/edit-find-target (match-string 1 input))
  (insert "... ")))
((string-match "^next \\(\\.+\\) *$" input)
 (sacha-plover/edit-find-target (match-string 1 input)))
((string-match "^previous \\(\\.+\\) *$" input)
 (re-search-backward (concat "\\b" (regexp-quote (match-string 1 input)) "\\b") nil t)
 (goto-char (match-end 0)))
(t (re-search-forward (concat "\\b" (regexp-quote input) "\\b")))
;; (t (sacha-subed-subtitle-set-text input))
))
nil))))

```

22.5.8 Using inotify to add Plover Clippy suggestions into Emacs

Update 2021-06-19: Changed to a vertical layout, added extra notes, simplified

I don't have a lot of screen space on my laptop, so I don't usually have the [Plover](#) suggestion window open as I type. I came up with a [Plover plugin](#) to let me flash the last [Plover Clippy](#) suggestion as a temporary notification. It went by too quickly, though, so I wrote something that uses inotify to monitor the clippy.txt log and put it in an Emacs buffer instead. It results in text like this:

```

Clippy
KHREUP PEU
added
ATD

```

```

(defvar sacha-clippy-recent-suggestions nil "Recent suggestions, limited by ↵
  ↵ `sacha-clippy-recent-suggestions-limit`.")
(defvar sacha-clippy-recent-suggestions-limit nil "If non-nil, keep this many suggestions.")
(defvar sacha-clippy-extra-notes nil "Extra notes to add at the end.")
;; ###autoload
(defun sacha-clippy-last ()
  (let ((value (string-trim (shell-command-to-string "tail -1 ~/.config/plover/clippy.txt | ↵
    ↵ cut -c 23-"))))
    (when (string-match "^\\(.*?\\)[ \\t]+|| .*? -> \\(\\.+\\)" value)
      (cons (match-string 1 value) (match-string 2 value)))))

```

```

;;;###autoload
(defun sacha-clippy-show (&rest _)
  (interactive)
  (with-current-buffer (get-buffer-create "*Clippy*")
    (let ((last (sacha-clippy-last)))
      (if sacha-clippy-recent-suggestions-limit
          (progn
            (unless (equal last (car sacha-clippy-recent-suggestions))
              (setq sacha-clippy-recent-suggestions (seq-take (cons last ←
↪ sacha-clippy-recent-suggestions) sacha-clippy-recent-suggestions-limit)))
            (erase-buffer)
            (insert (mapconcat (lambda (o) (format "| %s | %s |\n" (car o) (cdr o))) ←
↪ sacha-clippy-recent-suggestions "")))
          (unless (equal last (car sacha-clippy-recent-suggestions))
            (setq sacha-clippy-recent-suggestions (cons last sacha-clippy-recent-suggestions))
            (goto-char (point-min))
            (insert (format "| %s | %s |\n" (car last) (cdr last)))))))
    (when (get-buffer-window (current-buffer))
      (set-window-point (get-buffer-window (current-buffer)) (point-min)))))

;;;###autoload
(defun sacha-insert-symbol (symbol-name)
  (interactive (list
    (let ((orig-buffer (current-buffer)))
      (completing-read
        "Insert symbol: "
        #'help--symbol-completion-table
        (lambda (vv)
          ;; In case the variable only exists in the buffer
          ;; the command we switch back to that buffer before
          ;; we examine the variable.
          (with-current-buffer orig-buffer
            (or (get vv 'variable-documentation)
              (functionp vv)
              (and (boundp vv) (not (keywordp vv))))))))))
  (insert symbol-name))

;;;###autoload
(defun sacha-insert-variable-value (symbol-name)
  (interactive (list
    (let ((orig-buffer (current-buffer)))
      (completing-read
        "Insert variable: "
        #'help--symbol-completion-table
        (lambda (vv)
          ;; In case the variable only exists in the buffer
          ;; the command we switch back to that buffer before
          ;; we examine the variable.
          (with-current-buffer orig-buffer
            (or (get vv 'variable-documentation)
              (and (boundp vv) (not (keywordp vv))))))))))
  (insert (symbol-value (intern symbol-name))))

;;;###autoload
(defun sacha-insert-function (symbol-name)
  "Insert function name."
  (interactive (list
    (completing-read
      "Insert function: "

```

```

                #'help--symbol-completion-table
                'functionp)))
(insert symbol-name))

(defvar sacha-clippy-monitor nil)
;;;###autoload
(defun sacha-clippy-toggle-monitor ()
  (interactive)
  (if (inotify-valid-p sacha-clippy-monitor)
      (progn
        (message "Turning off")
        (inotify-rm-watch sacha-clippy-monitor))
      (message "Turning on")
      (setq sacha-clippy-monitor
            (inotify-add-watch
             (expand-file-name "~/.config/plover/clippy.txt") 'modify
             #'sacha-clippy-show))))

```

22.5.9 Stenoing interface

```

(defvar sacha-plover-quick-notes "~/proj/plover-notes/scratch.org")
(defvar sacha-plover-current-stroke-buffer "*Current stroke*")
;;;###autoload
(defun sacha-plover-add-note (string)
  (interactive "MNote: ")
  (with-current-buffer (find-file-noselect sacha-plover-quick-notes)
    (goto-char (point-min))
    (insert string)
    (unless (bolp) (insert "\n"))))

;;;###autoload
(defun sacha-plover-add-last-clippy-to-notes ()
  (interactive)
  (sacha-plover-add-note (format "| %s | %s |\n" (caar sacha-clippy-recent-suggestions) (cdar ↵
    ↵ sacha-clippy-recent-suggestions))))

;;;###autoload
(defun sacha-plover-scroll-notes ()
  (interactive)
  (message "Hello")
  (when (get-buffer-window (get-file-buffer sacha-plover-quick-notes))
    (with-selected-window (get-buffer-window (get-file-buffer sacha-plover-quick-notes))
      (scroll-up))))

;;;###autoload
(defun sacha-plover-scroll-notes-down ()
  (interactive)
  (message "World")
  (when (get-buffer-window (get-file-buffer sacha-plover-quick-notes))
    (with-selected-window (get-buffer-window (get-file-buffer sacha-plover-quick-notes))
      (scroll-down))))

;;;###autoload
(defun sacha-plover-spectra-last-clippy ()
  (interactive)
  (browse-url (format "http://localhost:8081/?outline=%s&translation=%s"

```

```

        (car (split-string (cdar sach-clippy-recent-suggestions) ", "))
        (caar sach-clippy-recent-suggestions))))

;;;###autoload
(defun sach-plover-layout-windows ()
  "Organize my windows."
  (interactive)
  (delete-other-windows)
  (when plover-websocket-stroke-buffer-name
    (with-selected-window (split-window-below -4)
      (switch-to-buffer plover-websocket-stroke-buffer-name)))
  (with-selected-window (split-window-right 100)
    (switch-to-buffer (get-buffer-create "*Clippy*")))
  (when sach-plover-quick-notes
    (with-selected-window (split-window-below 10)
      (switch-to-buffer (find-file sach-plover-quick-notes))))))

;;;###autoload
(defun sach-plover-clear-stroke-log ()
  (interactive)
  (with-current-buffer (get-buffer-create plover-websocket-stroke-buffer-name)
    (erase-buffer)))

```

```
(setq plover-websocket-stroke-buffer-name "*Stroke log*")
```

22.5.10 Cheat sheets

```

;;;###autoload
(defun sach-steno-quick-help ()
  (interactive)
  (with-selected-window
    (display-buffer-at-bottom
      (find-file-noselect "~/proj/plover-notes/cheat-sheet.txt")'())
    ;; ... mark it as dedicated to prevent focus from being stolen
    (set-window-dedicated-p (selected-window) t)
    ;; ... and shrink it immediately.
    (fit-window-to-buffer)))

(defhydra sach-hydra/cheatsheet/plover ()
  "SKHW- symbols -LTZ modifiers TWR- journal phrases
newparSKWRAURBGS bsPW-FP capKPA !space!capTK-LS cap!spaceKPA rmspcTK-FPS*
number: dupeD, revEU, 00/#OD, 00Z, $DZ, timeK- or -BG
`KH-FG ^KR-RT ~T*LD <AEPBGT =QA*LS >A*EPBGT |PAO*EUP \\_R*UND
-H-N --TK-RB ,KW-BG ;SKWR*RBGS :capSTPH-FPLT :KL-N !SKHRAPL
?H-F /OI .nspP-P ... SKWR-RBGS 'A*E,AE \"KW-GS,KR-GS
(PREN,* [PWR-BGT,* {TPR-BGT,* @KWRAT $TK-PL *STA*R
\\SPWHRAERB \\&SP-PBD #HAERB percPERS +PHR*US
retro KA*PD cap last *UPD cap all HRO*ERD lowered #* star AFPS add space TK-FPS del space
next HRO*ER lower KPA*L cap all
mode SPH-: RL lower R reset T Title -FPLT _RBGS")

(defhydra sach-hydra/cheatsheet/jade-plover-phrasing ()
  "S: SWR I, KPWR you, KWHR he, SKWHR she, TWH they, TWR we, KPWH it, STKPWHR nothing
M: OE don't (AOE really don't OEU don't really)
AU didn't, E doesn't, O can't, A or U really, AOEU don't even
E: PB know, P want, RPL remember, BL believe, FG forget, R are

```

```

BG can, BGD could, BGT can't, BLG like, BLGT like to, BLGTS likes to
BLT believe that, BS said, BT be the, BTS be said to, BTZ say to
D had, F have, FGT forgot, FLG feel like, FLGT felt like, FLT felt
FPLT must, FR ever, FRB wish, FRBT wish to, FS was, FT have to, FTS has to, FZ has, GT get, L ↔
↔ will, LG love, PBD need, PBG think, PBL mean,
PLD mind, PLG imagine, PLT might
"
)

(defhydra sachahydra/cheatsheet/emily-symbols ()
  "SKHW+ A (spc before) O (spc after) * (cap)
    v   E       U   EU
FG ws  Tab Backspace Del   Esc
RPBG   Up  Left      Right Down
FPBL   ↑  ←         →  ↓
FRPBG  PgU Home      End   PgD
blank  ' ' {*!}      {*?} spc
FPL    (  [          <   {
RBG    )  ]          >   }
'F *L +G &FBG \"FP #FRLG $RPBL percFRPB
,B -PL .R /RP :LG ;RB =PBLG @FRPBLG \\FB \\^RPG
_BG `P |PB ~FPBG
-S 2x -T 3x -ST 4x"
)

(defhydra sachahydra/cheatsheet/emily-modifiers ()
  "-LTZ F (C-) R (S-) P(s-) B(M-)
Z is STKPW
AO makes SKWR binary 0-9
Symbols with *, AO variants
TR tab delete backspace esc
KPWR up left down right
KPWHR pgup end home pgdown
blank esc tab return spc
TPH ( < [ {
KWR ) > ] }
P `
H '
!HR \"PH #TKHR $KPWH percPWHR &SKP *T +K ,W -TP .R /WH :TK ;WR
=TKPW ?TPW @TKPWHR \\PR ^KPR |PW ~TPWR")

```

22.5.11 Coding with Plover

```

;; ##autoload
(defun sachaplover-insert-defun ()
  "Prompt for parts of a function definition."
  (interactive)
  (insert "(defun ")
  (plover-websocket-send :translation "{MODE:LOWER}{MODE:SET_SPACE:-}")
  (insert (replace-regexp-in-string "-$" "" (read-string "Function name: ")))
  (insert " (")
  (plover-websocket-send :translation "{MODE:SET_SPACE: }")
  (let ((args (replace-regexp-in-string "\\<optional\\>" "&optional" (string-trim (read-string ↔
    ↔ "Args: "))))
    (insert args)
    (insert ")\n")
    (if (y-or-n-p "Interactive? ")
        (if (string= args "")

```

```

      (insert "(interactive)\n")
      (insert "(interactive (list))\n"))))
    (plover-websocket-send :translation "{MODE:RESET}{}{-[]}")
    (insert (format "\t%s\n"
      (replace-regexp-in-string "\"" "\\\"" (string-trim (read-string "Docstring: ↵
      ↵")))))
    (save-excursion (insert ")") (lisp-normalize-1))
    (plover-websocket-send :translation "{MODE:LOWER}"))

;;###autoload
(defun sacha-plover-insert-defvar ()
  (interactive)
  "Define a variable."
  (insert "(defvar ")
  (plover-websocket-send :translation "{MODE:LOWER}{MODE:SET_SPACE:-}")
  (insert (replace-regexp-in-string "-$" "" (read-string "Variable name: ")))
  (insert " ")
  (plover-websocket-send :translation "{MODE:RESET}{}{-[]}")
  (insert (string-trim (read-string "Default value: ")))
  (insert (format " \t%s\n"
    (replace-regexp-in-string "\"" "\\\"" (string-trim (read-string "Docstring: ↵
    ↵")))))

```

```

;;###autoload
(defun sacha-org-edit-special-dwim ()
  (interactive)
  (cond
    ((org-src-edit-buffer-p) (org-edit-src-exit))
    ((org-in-src-block-p) (org-edit-special))
    ((derived-mode-p 'org-mode)
     (org-insert-structure-template "src emacs-lisp")
     (org-edit-special))))
;;###autoload
(defun sacha-org-execute-special-dwim ()
  (interactive)
  (cond
    ((org-src-edit-buffer-p) (eval-buffer))
    ((org-in-src-block-p) (org-babel-execute-src-block))
    (t (eval-buffer))))

```

22.5.12 Measuring WPM

```

(use-package typing-speed :if sacha-laptop-p :load-path "~/elisp"
  :config (setq typing-speed-window 120))

```

22.5.13 Displaying frequency-sorted completions with stroke hints

```

(defun sacha-company-strokedict--grep-executable "grep")

;;###autoload
(defun sacha-company-strokedict--candidates (prefix)
  "Fetches the candidates matching PREFIX."
  (mapcar (lambda (o)
    (let ((data (split-string o "\t")))
      (propertize (car data) 'meta (cadr data)))))

```



```

        (split-string
         (shell-command-to-string (concat
                                   sacha-company-strokedict--grep-executable
                                   " -i "
                                   (shell-quote-argument (concat "^" prefix))
                                   " "
                                   "~/config/plover/annotated.txt -m 10"))
         "\n"))))

;;;###autoload
(defun sacha-company-strokedict--annotation (candidate)
  (let ((stroke (get-text-property 0 'meta candidate)))
    (if stroke
        (format " (%s)" stroke)
        "")))

;;;###autoload
(defun sacha-company-strokedict (command &optional arg &rest ignored)
  "`company-mode' backend for user-provided dictionaries. Dictionary files are lazy
loaded."
  (interactive (list 'interactive))
  (cl-case command
    (interactive (company-begin-backend 'sacha-company-strokedict))
    (candidates (sacha-company-strokedict--candidates arg))
    (prefix (when-let ((prefix (company-grab-word))) (substring-no-properties prefix)))
    (annotation (sacha-company-strokedict--annotation arg))
    (sorted t)
    (duplicates t)
    (no-cache t)))

```

This code added stroke annotations from the Typey Type dictionary to the frequency-sorted word list from https://github.com/hermitdave/FrequencyWords/blob/master/content/2018/en/en_full.txt.

```

import json

with open("/home/sacha/tmp/en_full.txt") as f:
    lines = f.readlines()
with open("/home/sacha/.config/plover/dictionaries/typey-type.json") as f:
    typey = json.load(f)
typey_inv = {v: k for k, v in typey.items()}
with open("/home/sacha/.config/plover/dictionaries/combined.json") as f:
    combined = json.load(f)
combined_inv = {}
for k, v in combined.items():
    if v in combined_inv:
        combined_inv[v] = combined_inv[v] + ', ' + k
    else:
        combined_inv[v] = k
with open("/home/sacha/.config/plover/annotated.txt", "w") as f:
    for line in lines:
        word = line.split()[0]
        if word in typey_inv:
            f.write("%s\t%s\n" % (word, typey_inv[word]))
        elif word in combined:
            f.write("%s\t%s\n" % (word, combined[word]))
        else:
            f.write("%s\n" % word)

```

22.6 Key chords

I'm on a Dvorak keyboard, so these might not work for you. Experimenting with this. `key-chord` lets you define keyboard shortcuts that use ordinary keys typed in quick succession. I haven't been using this lately, though...

Some code from <http://emacsredux.com/blog/2013/04/28/switch-to-previous-buffer/>

```
;; ###autoload
(defun sacha-key-chord-define (keymap keys command)
  "Define in KEYMAP, a key-chord of two keys in KEYS starting a COMMAND.
  \nKEYS can be a string or a vector of two elements. Currently only elements
  that corresponds to ascii codes in the range 32 to 126 can be used.
  \nCOMMAND can be an interactive function, a string, or nil.
  If COMMAND is nil, the key-chord is removed.

  MODIFICATION: Do not define the transposed key chord.
  "
  (if (/= 2 (length keys))
      (error "Key-chord keys must have two elements"))
  ;; Exotic chars in a string are >255 but define-key wants 128..255 for those
  (let ((key1 (logand 255 (aref keys 0)))
        (key2 (logand 255 (aref keys 1))))
    (define-key keymap (vector 'key-chord key1 key2) command)))
```

```
(fset 'key-chord-define 'sacha-key-chord-define)
```

Now let's set up the actual keychords.

```
(use-package key-chord
  :if sacha-laptop-p
  :hydra (sacha-key-chord-commands
    ()
    "Main"
    ("k" kill-sexp)
    ("h" sacha-org-jump :color blue)
    ("x" sacha-org-finish-previous-task-and-clock-in-new-one "Finish and clock in" ↵
    ↵ :color blue)
    ("b" helm-buffers-list :color blue)
    ("f" find-file :color blue)
    ("a" sacha-org-check-agenda :color blue)
    ("c" (call-interactively 'org-capture) "capture" :color blue)
    ("t" (org-capture nil "T") "Capture task")
    (". " repeat)
    ("C-t" transpose-chars)
    ("o" sacha-org-off-sacha-computer :color blue)
    ("w" sacha-engine-mode-hydra/body "web" :exit t)
    ("m" imenu :color blue)
    ("i" sacha-capture-timestamped-note-with-screenshot :exit t)
    ("n" sacha-capture-timestamped-note "Timestamped note" :exit t)
    ("q" quantified-track :color blue)
    ("r" sacha-describe-random-interactive-function)
    ("l" org-insert-last-stored-link)
    ("L" sacha-org-insert-link))
  :init
  (setq key-chord-one-key-delay 0.16)
  (setq key-chord-two-keys-delay 0.002)
  (key-chord-define-global "uu" 'undo))
```

```
(key-chord-define-global "jr" 'sacha-goto-random-char-hydra/sacha-goto-random-char)
(key-chord-define-global "kk" 'kill-whole-line)
(key-chord-define-global "et" 'sacha-stream-message)
(key-chord-define-global "em" 'embark-act)
(key-chord-define-global ".t" 'sacha-stream/body)
(key-chord-define-global "jj" 'avy-goto-word-1)
(key-chord-define-global "yy" 'sacha-window-movement/body)
(key-chord-define-global "jw" 'switch-window)
(key-chord-define-global "jl" 'avy-goto-line)
(key-chord-define-global "j." 'join-lines/body)
(key-chord-define-global "FF" 'find-file)
(key-chord-define-global "qq" 'sacha-quantified-hydra/body)
(key-chord-define-global "hh" 'sacha-key-chord-commands/body)
(key-chord-define-global "xx" 'er/expand-region)
(key-chord-define-global " " 'sacha-insert-space-or-expand)
(key-chord-define-global "vv" 'god-mode-all)
(key-chord-define-global "JJ" 'sacha-switch-to-previous-buffer)
(key-chord-mode -1)) ;; disable for now
```

Hmm, good point about `C-t` being more useful as a Hydra than as `transpose-char`. It turns out I actually do use `C-t` a fair bit, but I can always add it back as an option.

```
(keymap-global-set "C-t" 'sacha-key-chord-commands/body)
```

22.7 Exwm

Hmmm, I'm having a hard time getting used to this.

```
(use-package exwm
  :if sacha-laptop-p
  :init
  (progn
    (require 'exwm-config)
    (exwm-config-default)
    (exwm-enable)
    (exwm-input-set-key (kbd "s-p") 'fhd/toggle-exwm-input-line-mode-passthrough)
    (exwm-input-set-key (kbd "s-i") #'fhd/exwm-input-toggle-mode)))
```

```
;; ↩
  ↪ https://emacs.stackexchange.com/questions/33326/how-do-i-cut-and-paste-effectively-between-applications-whi
;; ###autoload
(defun fhd/exwm-input-line-mode ()
  "Set exwm window to line-mode and show mode line"
  (call-interactively #'exwm-input-grab-keyboard)
  (exwm-layout-show-mode-line))

;; ###autoload
(defun fhd/exwm-input-char-mode ()
  "Set exwm window to char-mode and hide mode line"
  (call-interactively #'exwm-input-release-keyboard)
  (exwm-layout-hide-mode-line))

;; ###autoload
(defun fhd/exwm-input-toggle-mode ()
  "Toggle between line- and char-mode"
  (interactive))
```

```

(with-current-buffer (window-buffer)
  (when (eq major-mode 'exwm-mode)
    (if (equal (second (second mode-line-process)) "line")
        (fhd/exwm-input-char-mode)
        (fhd/exwm-input-line-mode))))))

;;;###autoload
(defun fhd/toggle-exwm-input-line-mode-passthrough ()
  (interactive)
  (if exwm-input-line-mode-passthrough
      (progn
        (setq exwm-input-line-mode-passthrough nil)
        (message "App receives all the keys now (with some simulation)"))
      (progn
        (setq exwm-input-line-mode-passthrough t)
        (message "emacs receives all the keys now"))))
  (force-mode-line-update))

```

22.8 Emacspeak

```
(setq emacspeak-prefix (kbd "s-e"))
```

```

;;;###autoload
(defun sacha-emacspeak ()
  (interactive)
  (load-file "/home/sacha/vendor/emacspeak/lisp/emacspeak-setup.el")
  (setq emacspeak-use-auditory-icons t)
  (setq-default emacspeak-use-auditory-icons t)
  (setq-default dtk-quiet nil)
  (setq dtk-quiet nil))

;;;###autoload
(defun sacha-emacspeak-quiet ()
  (interactive)
  (setq emacspeak-use-auditory-icons nil)
  (setq-default emacspeak-use-auditory-icons nil)
  (setq-default dtk-quiet t)
  (setq dtk-quiet t)
  (dtk-interp-sync)
  (ad-disable-regexp "emacspeak"))

```

22.9 TOBLOG Manage photos with geeqie

image

Opening images directly in Emacs seems a little slow. Geeqie is pretty fast (after generating thumbnails) and can be remotely controlled via the command-line. I wrote a few functions to help me flip through images, add extra stuff to filenames, change dates, and insert references.

```

(defvar sacha-scan-directory "~/sync/scans/")
(defvar sacha-ipad-directory "~/sync/ipad")
(defvar sacha-portfolio-directory "~/sync/portfolio")
(defvar sacha-camera-directory "~/sync/camera")
(defvar sacha-private-sketches-directory "~/sync/private-sketches")
(defvar sacha-sketches-directory "~/sync/sketches")

;;;###autoload

```

```

(defun sacha-scans-dired () (interactive) (dired sacha-scan-directory "-lt"))
;; ###autoload
(defun sacha-geeie-next ()
  (interactive)
  (shell-command "geeie --remote -n"))
;; ###autoload
(defun sacha-geeie-previous ()
  (interactive)
  (shell-command "geeie --remote -b"))
;; ###autoload
(defun sacha-geeie-filename ()
  (string-trim (shell-command-to-string "geeie --remote --tell")))
;; ###autoload
(defun sacha-geeie-insert-file-link ()
  (interactive)
  (insert (org-link-make-string (concat "file:" (string-trim (shell-command-to-string "geeie ↵
    ↵ --remote --tell"))))))

```

```

;; ###autoload
(defun sacha-geeie-view (filenames)
  (interactive "f")
  (start-process-shell-command
   "geeie" nil
   (concat
    "geeie --remote "
    (mapconcat
     (lambda (f)
       (concat "file:" (shell-quote-argument f)))
     ((listp filenames) filenames)
     ((file-directory-p filenames)
      (list (car (seq-filter #'file-regular-p (directory-files filenames t))))
      (t (list filenames)))
     " "))))

```

```

(defvar sacha-rotate-jpeg-using-exiftran nil)

;; ###autoload
(defun sacha-rotate-image-clockwise (filename)
  (if (and sacha-rotate-jpeg-using-exiftran
           (string-match "jpe?g" (file-name-extension filename)))
      (call-process "exiftran" nil nil nil "-i" "-9" filename)
      (call-process "mogrify" nil nil nil "-rotate" "90" filename)))

;; ###autoload
(defun sacha-rotate-image-counterclockwise (filename)
  (if (and sacha-rotate-jpeg-using-exiftran
           (string-match "jpe?g" (file-name-extension filename)))
      (call-process "exiftran" nil nil nil "-i" "-2" filename)
      (call-process "mogrify" nil nil nil "-rotate" "270" filename)))

;; ###autoload
(defun sacha-geeie-rotate-clockwise ()
  (interactive)
  (sacha-rotate-image-clockwise (sacha-geeie-filename))
  (sacha-geeie-view (sacha-geeie-filename)))

```

```

;;;###autoload
(defun sacha-geeie-rotate-counterclockwise ()
  (interactive)
  (sacha-rotate-image-counterclockwise (sacha-geeie-filename))
  (sacha-geeie-view (sacha-geeie-filename)))

;;;###autoload
(defun sacha-rename-file-based-on-modification-time (filename)
  "Rename files to their modification time."
  (rename-file filename
                (expand-file-name
                 (concat
                  (format-time-string "%Y-%m-%d_%H%M%S"
                                        (file-attribute-modification-time (file-attributes filename)))
                  "."
                  (file-name-extension filename))
                 (file-name-directory filename))))

;;;###autoload
(defun sacha-geeie-change-date (filename new-time)
  (interactive (list (sacha-geeie-filename)
                    (let (t)
                      ((org-read-date-prefer-future nil))
                      (org-read-date nil t))))
  (let ((new-file (expand-file-name
                  (replace-regexp-in-string
                    "[0-9]*"
                    (format-time-string "%Y%m%d" new-time)
                    (file-name-nondirectory filename))
                  (file-name-directory filename)
                  filename))))
    (rename-file filename new-file)
    (sacha-geeie-view new-file)))

;;;###autoload
(defun sacha-geeie-rename-current (old-filename new-filename)
  (interactive
   (list (sacha-geeie-filename)
         (read-string "Filename: " (concat (file-name-base (sacha-geeie-filename)) " "))))
  (rename-file old-filename
               (expand-file-name
                (concat new-filename "."
                        (file-name-extension old-filename)
                        (file-name-directory old-filename))
                (file-name-directory old-filename)))

;;;###autoload
(defun sacha-geeie-crop-to-rectangle ()

```

```

(interactive)
(call-process
 "mogrify" nil nil nil "-crop"
 (string-trim (shell-command-to-string "geeque --remote --get-rectangle"))
 (sacha-geeque-filename))
(sacha-geeque-view (sacha-geeque-filename)))

;;;###autoload
(defun sacha-geeque-scans ()
  "Rename files and open the first one."
  (interactive)
  (mapc 'sacha-rename-file-based-on-modification-time (directory-files sacha-scan-directory t <
    ↪ "^scan"))
  (call-process "geeque" nil nil nil "--remote" (concat "file:" (shell-quote-argument <
    ↪ (seq-find 'file-regular-p (directory-files "~/sync/scans" t <
    ↪ "^([0-9]).*\\(gif\\|png\\|jpg\\)")))))

;;;###autoload
(defun sacha-geeque-delete-and-next ()
  (interactive)
  (let ((file (sacha-geeque-filename)))
    (sacha-geeque-next)
    (delete-file file t)))

```

```

(use-package ewmctrl :defer t)

```

```

;;;###autoload
(defun sacha-geeque-setup ()
  (interactive)
  (shell-command "wmctrl -r :ACTIVE: -b remove,maximized_vert,maximized_horz; xdotool <
    ↪ getactivewindow window-size 50% 100%")
  (shell-command "geeque &"))

```

```

(use-package pretty-hydra
 :config
 (pretty-hydra-define sacha-geeque ()
  ("Open"
   (("oo" sacha-geeque-setup "Setup")
    ("op" (sacha-geeque-view sacha-portfolio-directory) "Portfolio")
    ("oc" (sacha-geeque-view sacha-camera-directory) "Camera")
    ("oi" (sacha-geeque-view sacha-ipad-directory) "iPad")
    ("ox" (sacha-geeque-view "~/screenshots") "Screenshots")
    ("os" sacha-geeque-scans "Scans"))
   "Modify"
   (("[" sacha-geeque-rotate-counterclockwise "CCW")
    ("]" sacha-geeque-rotate-clockwise "CW")
    ("r" sacha-geeque-rename-current "Rename")
    ("d" sacha-geeque-change-date "Change date")
    ("c" sacha-geeque-crop-to-rectangle "Crop")
    ("k" (start-process "krita" nil "krita" (sacha-geeque-filename)) "krita")
    ("O" (shell-command (format "mogrify -auto-orient %s" <
    ↪ (shell-quote-argument (sacha-geeque-filename)))) "Rotate based on EXIF")
    ("g" (start-process "gimp" nil "gimp" (sacha-geeque-filename)) "gimp"))
   "Navigate"
   (("n" sacha-geeque-next "Next"))

```

```

        ("p" sacha-geeque-previous "Previous")
        ("x" sacha-geeque-delete-and-next "Delete"))

"Save"
((("p" (rename-file (sacha-geeque-filename)

↳      (expand-file-name (file-name-nondirectory (sacha-geeque-filename)) ↳
↳ sacha-sketches-directory))
      "Portfolio")
      ("s" (rename-file (sacha-geeque-filename)

↳      (expand-file-name (file-name-nondirectory (sacha-geeque-filename)) ↳
↳ sacha-sketches-directory))
      "Sketch"))

"Other"
(("<up>" (forward-line -1) :hint nil)
      ("<down>" forward-line :hint nil)

      ("im" (insert (format "{{<photo nas=\"1\" src=\"%s\">}}" ↳
↳ (sacha-geeque-filename))))
      ("if" (insert (sacha-geeque-filename) "\n")
      "Insert filename")
      ("v" (sacha-geeque-view (string-trim (thing-at-point 'line))) "View")
      ("il" (insert "- " (sacha-geeque-filename) "\n") "Insert filename as ↳
↳ list item")))))))

```

```

;;;###autoload
(defun sacha-move-portfolio-files ()
  (interactive)
  (mapc (lambda (f)
        (let ((new-dir
              (cond
                ((string-match ↳
↳ "#private" f) sacha-private-sketches-directory)
                ((string-match ↳
↳ "#me\\>" f) sacha-sketches-directory)
                (t ↳
↳ sacha-portfolio-directory))))
          (when new-dir (rename-file f (expand-file-name ↳
↳ (file-name-nondirectory f) new-dir))))
        (seq-filter
          'file-regular-p
          (directory-files sacha-scan-directory t "[0-9]+.##"))
        (shell-command-to-string "make-sketch-thumbnails"))

```

22.10 EmacsConf

```

;;;###autoload
(defun sacha-emacsconf-search-mail (talk)
  (interactive (list (emacsconf-complete-talk)))
  (emacsconf-with-talk-heading talk
    (notmuch-search (format "from:%s or to:%s" (org-entry-get (point) "EMAIL")
↳
↳ (org-entry-get (point) "EMAIL")))))

```

```

(use-package emacsconf

```



```

:after hydra
:bind (("C-c e" . emacsconf/body)
      ("M-g t" . emacsconf-go-to-talk))
:init
(require 'emacsconf-autoloads)
:hook
(message-send . emacsconf-mail-check-for-zzz-before-sending)
:config
(setq emacsconf-refresh-schedule-from-org t)
(defhydra emacsconf
  (:exit t)
  ("t" emacsconf-go-to-talk "talk")
  ("n" emacsconf-mail-notmuch-search-for-talk "notmuch search")
  ("f" emacsconf-cache-find-file "file")
  ("c" (find-file emacsconf-org-file) "conf.org")

  ("C" (let ((default-directory (file-name-directory emacsconf-org-file)))
         (call-interactively #'projectile-find-file)) "org dir")
  ("w" (let ((default-directory emacsconf-directory))
         (call-interactively #'projectile-find-file)))
  ("o" emacsconf-main-org-notebook-heading "org notes")
  ("a" (let ((default-directory emacsconf-ansible-directory))
         (call-interactively #'projectile-find-file)) "ansible")
  ("A" emacsconf-prep-agenda "agenda")
  ("I" emacsconf-extract-irc/body "IRC extract")
  ("ie" emacsconf-insert-talk-email "email")
  ("it" emacsconf-insert-talk-title "title")
  ("O" (switch-to-buffer (erc-get-buffer "#emacsconf-org"))))
  ("l" (let ((default-directory "~/proj/emacsconf/lisp"))
         (call-interactively #'projectile-find-file)))
  ("b" emacsconf-backstage-dired "backstage")
  ("u" emacsconf-upload-dired "upload")
  ("vie" emacsconf-volunteer-insert-email "volunteer email")
  ("U" emacsconf-res-upload-dired "upload"))
:load-path "~/proj/emacsconf/lisp")
(keymap-global-set "M-g t" 'emacsconf-go-to-talk)

```

22.11 ChatGPT, AI, and large-language models

```

(use-package chat
  :disabled t
  :vc (:url "https://github.com/iwahbe/chat.el"))
(use-package org-ai
  :disabled t
  :vc (:url "https://github.com/rkism/org-ai"))
(use-package khoj
  :after org
  :disabled t
  :quelpa (khoj :fetcher github :repo "debanjum/khoj" :files (:defaults ↵
    ↵ "src/interface/emacs/khoj.el"))
  :bind ("C-c s" . 'khoj))

```

```

;; ##autoload
(defun gptel-api-key-from-environment (&optional var)
  (lambda ()
    (getenv (or var

```

```

(thread-first
  (type-of gptel-backend)
  (symbol-name)
  (substring 6)
  (upcase)
  (concat "_API_KEY"))))))

```

```

(use-package gptel
  :commands (gptel gptel-send gptel-set-topic gptel-menu)
  :defer t
  :config
  (setq sachagptel-groq
    (gptel-make-openai "Groq"
      :host "api.groq.com"
      :endpoint "/openai/v1/chat/completions"
      :stream t
      :key (gptel-api-key-from-environment "GROQ_API_KEY")
      :models '(llama-3.3-70b-versatile
                 llama-3.1-8b-instant
                 openai/gpt-oss-20b
                 openai/gpt-oss-120b)))
  (setq sachagptel-gemini
    (gptel-make-gemini "Gemini"
      :key (gptel-api-key-from-environment "GEMINI_API_KEY")
      :stream t
      :models '(gemini-3-flash-preview
                 gemini-2.5-flash
                 gemini-2.5-pro
                 gemini-2.5-flash-preview-09-2025
                 gemini-2.5-flash-lite)))
  (setq sachagptel-gemini-paid
    (gptel-make-gemini "Gemini - paid"
      :key (gptel-api-key-from-environment "GEMINI_PAID_API_KEY")
      :stream t
      :models '(gemini-3-flash-preview
                 gemini-2.5-flash
                 gemini-2.5-flash-preview-09-2025
                 gemini-2.5-flash-lite)))
  (setq sachagptel-mistral
    (gptel-make-openai "Mistral"
      :key (gptel-api-key-from-environment "MISTRAL_API_KEY")
      :host "api.mistral.ai"
      :endpoint "/v1/chat/completions"
      :stream t
      :models '(mistral-medium
                 mistral-large-2411)))
  (setq gptel-model 'gemini-3-flash-preview
        gptel-backend sachagptel-gemini
        gptel-log-level 'info)
  :hook
  (gptel-post-stream . gptel-auto-scroll)
  (gptel-post-response . gptel-end-of-response))

```

So that I don't have to use `gptel-menu` just to change the model:

```

;; ##autoload
(defun sachagptel-set-model ()
  "Interactively set the gptel model."

```

```
(interactive)
(require 'gptel-transient)
(let* ((infix (get 'gptel--infix-provider 'transient--suffix))
      (reader (oref infix reader))
      (result (funcall reader "Model: ")))
  (setq gptel-backend (car result))
  (setq gptel-model (cadr result))))
```

22.11.1 agent-shell

From <https://github.com/xenodium/agent-shell> :

```
;;;###autoload
(defun sacha-agent-shell-dot-subdir (subdir)
  (let* ((cwd (string-remove-suffix "/" (agent-shell-cwd)))
        (sanitized (replace-regexp-in-string "/" "-" (string-remove-prefix "/" cwd))))
    (expand-file-name subdir (locate-user-emacs-file (concat "agent-shell/" sanitized)))))
```

```
(use-package agent-shell
  :config
  (setopt agent-shell-preferred-agent-config (agent-shell-anthropic-make-claude-code-config))
  (setopt agent-shell-session-strategy 'prompt)
  (setopt agent-shell-dot-subdir-function #'sacha-agent-shell-dot-subdir)
)
```

22.11.2 Interact with Google Gemini web interface through Spookfox

Moved most of the code to [learn-lang-spookfox.el](#) , notably:

- learn-lang-spookfox-ai-flycheck-send
- learn-lang-spookfox-ai-send-text-and-get-json
- learn-lang-spookfox-ai-latest-response-html

```
;;;###autoload
(defun sacha-spookfox-ai-replace-with-code ()
  (interactive)
  (erase-buffer)
  (insert (learn-lang-spookfox-ai-get-latest-code)))

;;;###autoload
(defun sacha-spookfox-ai-ediff-with-code ()
  (interactive)
  (with-current-buffer (get-buffer-create "*ai*")
    (erase-buffer)
    (insert (learn-lang-spookfox-ai-get-latest-code)))
  (ediff-buffers (current-buffer) (get-buffer-create "*ai*")))
```

22.12 Chronos timers

Tip from zaeph. Might not use this much yet, as I don't spend that much time in front of my computer. Could be handy someday, though.

```
(use-package chronos
  :config
  (add-hook 'chronos-expiry-functions #'chronos-desktop-notifications-notify)
  (add-hook 'chronos-expiry-functions #'chronos-buffer-notify))
```

22.13 Idle timer

This snippet is from John Wiegley - <http://lists.gnu.org/archive/html/emacs-orgmode/2010-03/msg00367.html>. It shows the org agenda when Emacs is idle.

Thanks to winner-mode, I can get back to my previous buffers with C-c left.

```
(defvar sacha-on-air nil "Non-nil means I'm on air, hide stuff.")

;;;###autoload
(defun sacha-jump-to-org-agenda-when-idle ()
  (interactive)
  (unless sacha-on-air
    (let ((buf (get-buffer "*Org Agenda*"))
          wind)
      (if buf
        (if (setq wind (get-buffer-window buf))
            (select-window wind)
            (if (called-interactively-p 'any)
                (progn
                  (select-window (display-buffer buf t t))
                  (org-fit-window-to-buffer)
                  ;; (org-agenda-redo)
                ))
            (with-selected-window (display-buffer buf)
              (org-fit-window-to-buffer)
              ;; (org-agenda-redo)
            )))
        (call-interactively 'org-agenda-list))))
  ;;(let ((buf (get-buffer "*Calendar*")))
  ;;  (unless (get-buffer-window buf)
  ;;    (org-agenda-goto-calendar)))
  )
```

```
(run-with-idle-timer 300 t 'sacha-jump-to-org-agenda-when-idle)
```

22.14 Completion at point?

<https://protesilaos.com/codelog/2024-11-28-basic-emacs-configuration/>

```
(use-package corfu :init (global-corfu-mode)
  :bind (:map corfu-map ("<tab>" . corfu-complete))
  :config
  (setq tab-always-indent 'complete)
  (setq corfu-preview-current nil)
  (setq corfu-min-width 20)

  (setq corfu-popupinfo-delay '(1.25 . 0.5))
  (corfu-popupinfo-mode 1) ; shows documentation after `corfu-popupinfo-delay'
```

```
;; Sort by input history (no need to modify `corfu-sort-function').
(with-eval-after-load 'savehist
  (corfu-history-mode 1)
  (add-to-list 'savehist-additional-variables 'corfu-history))
)
(use-package cape
  :bind (("M-/" . completion-at-point))
  :init
  (add-to-list 'completion-at-point-functions #'cape-dabbrev)
  (add-to-list 'completion-at-point-functions #'cape-file)
  (add-to-list 'completion-at-point-functions #'cape-elisp-block)
  (add-to-list 'completion-at-point-functions #'cape-abbrev)
  (add-to-list 'completion-at-point-functions #'cape-dict)
  (add-to-list 'completion-at-point-functions #'cape-line)
)
```

22.15 Enable minibuffer completion

[2013-03-31 Sun] Superseded by ido-hacks?

It can be difficult to remember the full names of Emacs commands, so I use `icomplete-mode` for minibuffer completion. This also makes it easier to discover commands.

```
(icomplete-mode 1)
```

22.16 Encryption

```
(require 'org-crypt)
(org-crypt-use-before-save-magic)
(setq org-tags-exclude-from-inheritance (quote ("crypt")))

(setq org-crypt-key nil)
;; GPG key to use for encryption
;; Either the Key ID or set to nil to use symmetric encryption.

;; (setq auto-save-default nil)
;; Auto-saving does not cooperate with org-crypt.el: so you need
;; to turn it off if you plan to use org-crypt.el quite often.
;; Otherwise, you'll get an (annoying) message each time you
;; start Org.

;; To turn it off only locally, you can insert this:
;;
;; # -*- buffer-auto-save-file-name: nil; -*-
```

22.17 Tools for organizing

```
;;;###autoload
(defun sacha-rename-bank-statements ()
  (interactive)
  (let ((months '("Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec")))
    (cl-loop for i from 1 to 12 do
```

```

        (message "%d" i)
        (goto-char (point-min))
        (while (re-search-forward (elt months (1- i)) nil t)
          (ignore-errors
            (replace-match (format "%02d" i))
          )))
      )
    )
  )
)

;; ###autoload
(defun sachar-rename-scanned-receipts ()
  "Display and rename the scanned or saved files."
  (interactive)
  (delete-other-windows)
  (mapc (lambda (o)
    (find-file o)
    (let ((new-name (concat (read-string "New filename: ") ".jpg")))
      (kill-buffer)
      (unless (string= new-name ".jpg")
        (rename-file o new-name))))
    (or (if (derived-mode-p 'dired-mode)
      (dired-get-marked-files))
      (directory-files default-directory t "^[_0-9]+\\.jpg")))))

```

22.18 Fun and games

22.18.1 TODO Make memes from Emacs

```

(use-package meme
  :load-path "~/vendor/meme"
  :init (provide 'imgur) ; fake this
  :defer t
  :commands meme
  :config
  (setq meme-dir "~/vendor/meme/images")
  (setq meme-font "Roboto"))

```

22.18.2 Cubing

Rubik's Cube

```

(use-package eagle
  :disabled t
  :vc (:url "https://codeberg.org/akib/emacs-eagle.git")
  :defer t)
(use-package cube
  :vc (:url "https://codeberg.org/akib/emacs-cube.git")
  :defer t)

```

Diagrams

```

;; Start of cubing code
(defun sachar-cubing-pos (size n i)
  (list
    (* (/ size n) (% i n))
    (* (/ size n) (/ i n))))

;; ###autoload
(defun sachar-cubing-last-layer-arrows (arrows)

```



```

(with-temp-buffer
  (svg-print svg)
  (buffer-string))))

(defvar sacha-cubing-colors '((?R . "#ff0000")
  ↪
  (?G . "#00ff00")
  ↪
  (?B . "#0000ff")
  ↪
  (?O . "#ed7117")
  ↪
  (?Y . "#ffff00")
  ↪
  (?W . "#ffffff")
  ↪
  (?\? . "#666666"))))

;;###autoload
(defun sacha-cubing-last-layer-with-sides (sides top arrows)
  "Draw a diagram of the top of the cube.
  The style is similar to https://www.cubescills.com/uploads/pdf/tutorials/pll-algorithms.pdf .
  SIDES is a string specifying colors going clockwise from the back-left side.
  TOP is a string specifying colors going from left to right, top to bottom.
  Arrows are defined as a list of lists of the form ((from to) (from to t) ...).
  Cells are numbered from left to right, top to bottom, with the top left box being 0.
  Ex: (sacha-cubing-last-layer-with-sides \"ORRB00GGGRBB\" \"YYYYYYYY\" '((3 1 t) (2 8 t)))
  "

  (let* ((size 99)
         (n 3)
         (side-size 10)
         (cell-size (/ (- size (* 2 side-size)) n))
         (arrow-color "#000")
         (svg (svg-create size size)))

    (svg--append
     svg
     (dom-node
      'defs
      nil
      (dom-node
       'marker
       '((id . "arrowhead")
        (markerWidth . "10")
        (markerHeight . "7")
        (refX . "0")
        (refY . "3.5")
        (orient . "auto-start-reverse")))
      (dom-node
       'polygon
       `(((fill . ,arrow-color)
         (points . "0 0, 4 3.5, 0 7"))))))

    ;; Draw the sides. It's a string of colors going clockwise from back left
    (when sides
      (dotimes (i (* n 4))
        (apply 'svg-rectangle
                (append
                 (list svg)
                 (pcase (/ i n)

```



```

↪ cell-size) side-size)                                (0 (list (+ (* (% i n) ↪
↪                                                         ↪
↪                                                         ↪
↪ cell-size                                             ↪
↪ side-size))                                           ↪
↪                                                         (1 (list (+ side-size ↪
↪ (* n cell-size))                                     ↪
↪                                                         ↪
↪ (+ (* (% i n) cell-size) side-size)                 ↪
↪                                                         ↪
↪ side-size                                             ↪
↪                                                         ↪
↪ cell-size))                                           ↪
↪                                                         (2 (list (+ (* (- n (% ↪
↪ i n) 1) cell-size) side-size)                       ↪
↪                                                         ↪
↪ (+ (* n cell-size) side-size)                       ↪
↪                                                         ↪
↪ cell-size                                             ↪
↪                                                         ↪
↪ side-size))                                           ↪
↪                                                         (3 (list 0
↪                                                         ↪
↪ (+ (* (- n (% i n) 1) cell-size) side-size)         ↪
↪                                                         ↪
↪ side-size                                             ↪
↪                                                         ↪
↪ cell-size)))                                           ↪
↪                                                         (list
↪                                                         :stroke-width 1
↪                                                         :stroke "#666"
↪                                                         :fill (assoc-default (elt ↪
↪ sides i)                                             ↪
↪                                                         ↪
↪                                                         sacha-cubing-colors ↪
↪                                                         ↪
↪                                                         'eq
↪                                                         ↪
↪                                                         (assoc-default ?\? ↪
↪ sacha-cubing-colors))))))
↪                                                         ;; Draw the top face specified by a string of colors going from left to right, ↪
↪ top to bottom
↪ (dotimes (i (* n n))
↪   (let ((pos (sacha-cubing-pos (* cell-size n) n i)))
↪     (svg-rectangle
↪       svg
↪       (+ side-size (car pos))
↪       (+ side-size (cadr pos))
↪       cell-size
↪       cell-size
↪       :fill (if top
↪                                                         (assoc-default (elt ↪
↪ top i) sacha-cubing-colors
↪                                                         ↪
↪                                                         'eq

```

```

↩️                                     ↩️
↩️                                     (assoc-default ?\? sachacubing-colors))
                                       (assoc-default ?\? ↩️
↩️ sachacubing-colors)))
                                       :stroke-width 1
                                       :stroke "#666"))))
;; Draw the arrows
(dolist (arrow arrows)
  (let ((from (car arrow))
        (to (cadr arrow)))
    (apply 'svg-line
            (append
              (list svg)
              (mapcar (lambda (o) (+ ↩️
↩️                                     ↩️
↩️                                     (sachacubing-pos (* n cell-size) n from))
                    (mapcar (lambda (o) (+ ↩️
↩️                                     ↩️
↩️                                     (sachacubing-pos (* n cell-size) n to))
                    (list
                      :stroke-width 2
                      :stroke arrow-color
                      :opacity 0.5
                      :marker-start (if (elt arrow ↩️
↩️                                     ↩️
↩️                                     (url("#arrowhead")))
                      :marker-end ↩️
↩️                                     (url("#arrowhead"))))))))
      (with-temp-buffer
        (svg-print svg)
        (buffer-string))))
;; end of cubing code
```

22.18.3 Minecraft

<https://github.com/rasensuihei/mcf>

```
(use-package mcf
  :load-path "~/vendor/mcf"
  :mode ("\\.mcf\\.function\\'" . mcf-mode)
  ;; rcon settings are in my .emacs.secrets file
  :commands (mcf-rcon mcf-mode)
)
```

22.18.4 Typing of Emacs

```
(use-package typing :disabled t
:init
(autoload 'typing-of-emacs "typing" nil t)
:config
(progn
  (setq toe-starting-length 6)
  (setq toe-starting-time-per-word 2)
  (setq toe-max-length 20)))
```

22.19 Speech synthesis (experimental)

```
(use-package speechd-el)
(with-eval-after-load 'speechd-speak
  (setq speechd-speak-ignore-command-keys
    (append
      '(lispy-delete-backward
        lispy-delete
        python-indent-dedent-line-backspace)
      speechd-speak-ignore-command-keys)))
```

```
(defvar sacha-espeak-command "c:/program files (x86)/espeak/command_line/espeak.exe")
;; ##autoload
(defun sacha-say (string &optional speed)
  (interactive "MString: ")
  (setq speed (or speed 175))
  (call-process sacha-espeak-command nil nil nil string "-s" speed))
```

22.20 Comparison-shopping with Org Mode

emacs.org

I don't like shopping. We're lucky to be able to choose, but I get overwhelmed with all the choices. I'm trying to get the hang of it, though, since I'll need to shop for lots of things for A- over the years. One of the things that's stressful is comparing choices between different webpages, especially if I want to get A-'s opinion on something. Between the challenge of remembering things as we flip between pages and the temptations of other products she sees along the way... Ugh.

I think there are web browser extensions for shopping, but I prefer to work within Org Mode so that I can capture links from my phone's web browser, refile entries into different categories, organize them with keyboard shortcuts, and tweak things the way I like. So if I have subheadings with the **NAME**, **PRICE**, **IMAGE**, and **URL** properties, I can make a table that looks like this:

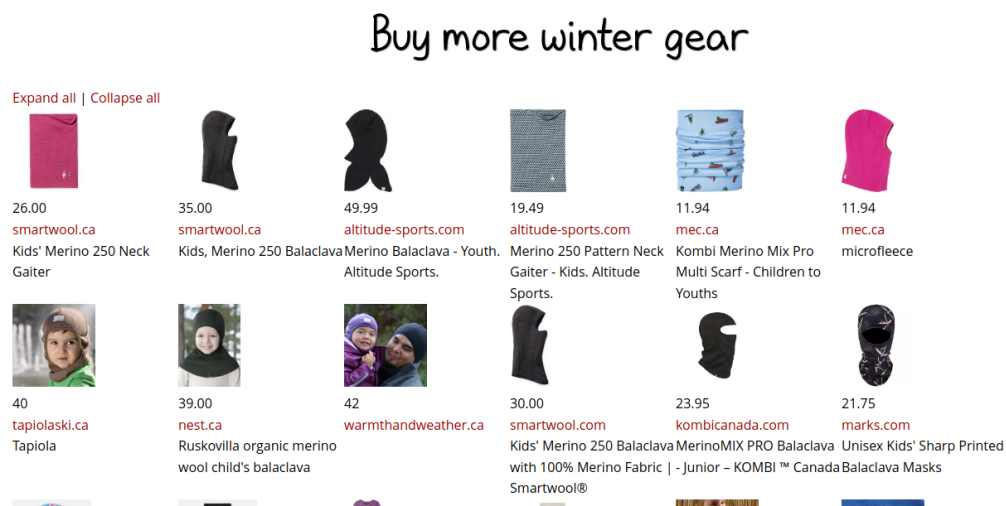


Figure 29: Comparison-shopping

using code that looks like this:

```
#+begin_src emacs-lisp :eval yes :exports results :wrap EXPORT html
(sacha-org-format-shopping-subtree)
```

```
#+end_src
```

and I can view the table by exporting the subtree with HTML using `org-export-dispatch` (`C-c C-e C-s h o`). When I add new items, I can use `C-u C-c C-e` to reexport the subtree without navigating up to the root.

Here's the very rough code I use for that:

```
(defvar sach-get-shopping-details-functions
  '(sacha-org-shopping-get-details-from-spookfox
    sach-get-shopping-details-amazon
    sach-get-shopping-details-uniqlo
    sach-get-shopping-details-manually))

;;;###autoload
(defun sach-get-shopping-details-manually (link)
  (when (string-match "theshoecompany\\|dsw" link)
    (browse-url link)
    (list
     (cons 'url link)
     (cons 'image (read-string "Image: "))
     (cons 'price (read-string "Price: ")))))

;;;###autoload
(defun sach-get-shopping-details-amazon (link)
  (when (string-match "amazon.ca" link)
    (with-current-buffer (url-retrieve-synchronously link)
      (goto-char (point-min))
      (re-search-forward "^$")
      (let ((doc (libxml-parse-html-region (point) (point-max))))
        (list (cons 'name (dom-text (dom-by-tag doc 'title)))
              (cons 'description (dom-texts (dom-by-id doc "productDescription")))
              (cons 'image (dom-attr (dom-by-tag doc "imgTagWrapperId") 'img) 'src))
              (cons 'price (dom-texts (dom-by-id doc "priceblock_ourprice"))))))))

;;;###autoload
(defun sach-shopping-reformat-ld-data (data)
  (let-alist data
    (list (cons 'name .name)
          (cons 'url (or .url .@id))
          (cons 'brand .brand.name)
          (cons 'description .description)
          (cons 'rating .aggregateRating.ratingValue)
          (cons 'ratingCount .aggregateRating.reviewCount)
          (cons 'image (cond
                     ((stringp .image) .image)
                     ((stringp .image.url) .image.url)
                     (t (elt .image 0))))
          (cons 'price
                (assoc-default 'price (cond
```

```

↪                                     ((arrayp .offers)
↪                                     ↪
↪                                     (elt .offers 0))
↪                                     ↪
↪                                     (t .offers))))))

;;;###autoload
(defun sacha-get-shopping-details ()
  (goto-char (point-min))
  (let (data)
    (cond
      ((re-search-forward " data-section-data
> nil t)
      (setq data (json-read))
      (let-alist data
        (list (cons 'name .product.title)
              (cons 'brand .product.vendor)
              (cons 'description .product.description)
              (cons 'image (if (stringp .image) .image (concat "https:" ↪
↪ .product.featured_image)))
              (cons 'price (/ .product.price 100.0))))))
      ((and (re-search-forward "<script type=\"application/ld\\+json\\>" nil t)
            (null (re-search-forward "Fabric Fabric\\|angryballerinafabrics" nil t))) ; ↪
↪ Carter's, Columbia?
      (setq data (json-read))
      (if (vectorp data) (setq data (elt data 0)))
      (if (assoc-default '@graph data)
          (setq data (assoc-default '@graph data)))
      (if (vectorp data) (setq data (elt data 0)))
      (sacha-shopping-reformat-ld-data data))
      (t
       (goto-char (point-min))
       (re-search-forward "^$")
       (let* ((doc (libxml-parse-html-region (point) (point-max)))
              (result
               `((name . ,(string-trim (dom-text ↪
↪ (dom-by-tag doc "title"))))
               (description . ,(string-trim ↪
↪ (dom-text (dom-by-tag doc "title")))))
              ))
       (mapc (lambda (property)
               (let ((node
↪
↪ (dom-search
↪
↪ doc
↪
↪ (lambda (o)
↪
↪ (delq nil
↪
↪ (mapcar (lambda (p)
↪
↪ (or ↪
↪ (string= (dom-attr o 'property) p)
↪
↪
↪ (string-match p (or (dom-attr o 'class) ""))))
↪

```

```

      (cdr ←
      ↪ property))))))
      (when node ←
      ↪
      ↪ (or (dom-attr node 'content)
      ↪
      ↪ (string-trim ←
      ↪ (dom-text node))))))
      '((name "og:title" "pdp-product-title")
        (brand "og:brand")
        (url "og:url")
        (image "og:image")
        (description "og:description")
        (price "og:price:amount" "product:price:amount" "pdp-price-label")))
      result)
    ))))
;; ##autoload
(defun sach-org-insert-shopping-details ()
  (interactive)
  (save-excursion
    (org-insert-heading)
    (when (string-match "^https://" (car kill-ring))
      (save-excursion (yank)))
    (sach-org-update-shopping-details)
    (when (org-entry-get (point) "NAME")
      (org-edit-headline (org-entry-get (point) "NAME")))))
;; ##autoload
(defun sach-org-shopping-get-details-from-spookfox (&optional link)
  "Get shopping details from Spookfox for sites that use Javascript."
  (let* ((current-url
    ↪ (spookfox-js-injection-eval-in-active-tab ←
    ↪ "window.location.href" t))
    (data
    ↪ (when (or (not link)
    ↪ (string= link ←
    ↪ current-url))
    ↪ (json-parse-string
    ↪ (spookfox-js-injection-eval-in-active-tab
    ↪ "JSON.stringify({
name: window.rawData?.Product?.Name || ←
  ↪ document.querySelector('meta[property="og:title\\"]')?.getAttribute('content'),
image: document.querySelector('meta[property="og:image\\"]')?.getAttribute('content'),
description: window.rawData?.Product?.Description || ←
  ↪ document.querySelector('meta[property="og:description\\"]')?.getAttribute('content'),
url: document.querySelector('link[rel="canonical\\"]')?.getAttribute('href') || ←
  ↪ window.location.href,
price: window.rawData?.Product?.MinPrice || ←
  ↪ document.querySelector('.nl-price--total')?.textContent?.replace(/^\s*$/, ' ')
}))"
    ↪ t)
    ↪ :object-type 'alist))))
    (when (alist-get 'name data)
      data)))
;; (sach-org-get-shopping-details-uniqlo ←

```

```

↪ "https://www.uniqlo.com/ca/en/products/E451023-000?colorCode=COL07&sizeCode=KSS020")
;;###autoload
(defun sacha-org-update-shopping-details ()
  (interactive)
  (let (data)
    (if (and (< (point)
                                                         (save-excursion ↪
                                                         ↪ (org-end-of-subtree)))
         (re-search-forward org-link-any-re ↪
                             (save-excursion (org-end-of-subtree)) t))
        (let ((link (org-element-property :raw-link ↪
                                             (org-element-context))))
          (setq
            data
            (or (run-hook-with-args-until-success ↪
              'sacha-get-shopping-details-functions link)
                (with-current-buffer ↪
                  (url-retrieve-synchronously link)
                    (sacha-get-shopping-details))))
            (setq data (sacha-org-shopping-get-details-from-spookfox)))
        (when data
          (let-alist data
            (org-entry-put (point) "NAME" \.name)
            (org-entry-put (point) "URL" \.url)
            (org-entry-put (point) "BRAND" \.brand)
            (org-entry-put (point) "DESCRIPTION" (replace-regexp-in-string ↪
              "&#039;" " '" (replace-regexp-in-string "\n" " " (or \.description ""))))
            (org-entry-put (point) "IMAGE"
                           (if ↪
                             (string-match "https?" \.image)
                             ↪
                             ↪
                             (concat "https:" \.image)))
            (org-entry-put (point) "PRICE" (cond ((stringp \.price) ↪
              ↪ \.price) ((numberp \.price) (format "%.2f" \.price)) (t "")))
            (if \.rating (org-entry-put (point) "RATING" (if (stringp ↪
              ↪ \.rating) \.rating (format "%.1f" \.rating))))
            (if \.ratingCount (org-entry-put (point) "RATING_COUNT" (if ↪
              ↪ (stringp \.ratingCount) \.ratingCount (number-to-string \.ratingCount)))))))
;;###autoload
(defun sacha-org-format-shopping-subtree (&optional height large)
  (concat
    "<style>body { max-width: 100% !important } #content { max-width: 100% !important } ↪
    ↪ .item img { max-height: "
    (or height "100px")
    " } .item img:hover { max-height: " (or large "400px") " }</style><div ↪
    ↪ style=\"display: flex; flex-wrap: wrap; align-items: flex-start\">"
    (string-join
      (save-excursion
        (org-map-entries
          (lambda ()
            (if (org-entry-get (point) "URL")
                (format
                  "<div class=item style=\"width: ↪
                  ↪ %s\"><div><a href=\"%s\"><img src=\"%s\" height=100></a></div>
                  <div>%s</div>
                  <div><a href=\"%s\">%s</a></div>

```

```

<div>%s</div>
<div>%s</div></div>"

                                (or height "200px")
                                (org-entry-get (point) "URL")
                                (org-entry-get (point) "IMAGE")
                                (or (org-entry-get (point) "PRICE") "")
                                (org-entry-get (point) "URL")
                                (url-domain (url-generic-parse-url ←
↪ (org-entry-get (point) "URL")))

                                (or (org-entry-get (point) "NAME") "")
                                (or (org-entry-get (point) "NOTES") ""))
                                ""))
                                nil
                                (if (org-before-first-heading-p) nil 'tree)))
                                ""
                                "</div>"))

;;###autoload
(defun sach-get-shopping-details-uniqlo (link)
  (when (string-match ←
↪ "https://www.uniqlo.com/ca/en/products/\\([^[?]+\\)\\(\\(\\?\\(\\.\\*\\)\\)\\)?\" link)
    (let ((code (match-string 1 link))
          (params (org-protocol-convert-query-to-plist ←
↪ (match-string 3 link)))
          item)
      (setq item
              (car
               (assoc-default
                'items
                (assoc-default
                 'result
                 (with-current-buffer
↪ (url-retrieve-synchronously
              (concat ←
↪ "https://www.uniqlo.com/ca/api/commerce/v3/en/products/" code))
              (goto-char (point-min))
              (re-search-forward "^$")
              (json-parse-buffer ←
↪ :object-type 'alist :array-type 'list :null-object nil))))))
        (list
         (cons 'price
               (or (assoc-default
                    'value
                    (or
                     (assoc-default 'promo ←
                     (assoc-default 'base ←
↪ (assoc-default 'prices item))
↪ (assoc-default 'prices item))))
                 ""))
         (cons 'image
               (assoc-default
                'url
                (seq-find
                 (lambda (entry)
                  (or (null (plist-get params ←
↪ :colorCode)))
                  (string=
↪

```



```

↪ (concat "COL" (or (assoc-default 'colorCode entry) ""))
↪ (plist-get params :colorCode)))
↪ 'images item)))) (assoc-default 'main (assoc-default ↪
      (cons 'price
              (or (assoc-default
                  'value
                  (assoc-default
                    'promo
                    (assoc-default ↪
↪ 'prices item))))
      (assoc-default 'base ↪
      ""))
      (cons 'name
              (assoc-default 'name item))
      (cons 'description
              (concat (assoc-default 'longDescription item) ↪
↪ " - "
      ↪
      (assoc-default 'washingDescription item)))
      (cons 'url link))))))

```

At some point, it would be nice to keep track of how I feel about different return policies, and to add more rules for automatically extracting information from different websites. ([org-chef](#) might be a good model.) In the meantime, this makes it a little less stressful to look for stuff.

23 Style stuff and other exports

Update the URL as we scroll: ([based on this](#))

You can [e-mail me](#) or [find me on Mastodon](#).

24 Index of sections by module

This uses code defined in [Tangle Emacs config snippets to different files and add boilerplate](#) (p. 250).

`elisp:sacha-emacs-config-update-module-info`

- `sacha-ai`
 - [ChatGPT, AI, and large-language models](#) (p. 709)
 - [agent-shell](#) (p. 711)
- `sacha-appearance`
 - [Set up a color scheme](#) (p. 64)
- `sacha-bbb`
 - [Write a function to help with crontab entries](#) (p. 549)
- `sacha-blog`
 - [Completing blog posts](#) (p. 48)
 - [List all blog posts that match a category or title search](#) (p. 273)

- Making it easier to add a category to a blog post (p. 274)
- embark-11ty (p. 281)
- Listing exported Org posts (p. 285)
- Comments (p. 286)
- Adding Mastodon toots as comments in my 11ty static blog (p. 553)
- mastodon.el: Copy toot URL after posting; also, copying just this post with 11ty (p. 560)
- Tooting a link to the current post (p. 575)
- Blog (p. 600)
- sachabrigade
 - Automating buttons (p. 615)
 - Removing sections (p. 621)
 - Formatting calls to action (p. 623)
 - Changing link colours (p. 624)
 - Just the headings (p. 626)
 - Wrapping it up (p. 626)
 - Getting a Google Docs draft ready for Mailchimp via Emacs and Org Mode (p. 628)
 - Bike Brigade: working with Mailchimp images (p. 632)
 - Emacs: Updating a Mailchimp campaign using a template, sending test e-mails, and scheduling it (p. 638)
- sachacode
 - Format source (p. 256)
 - Python (p. 502)
 - Web development (p. 503)
 - LSP (p. 504)
 - Javascript (p. 531)
 - HTML (p. 533)
 - SQLite (p. 549)
- sachacomplete
 - Extended command list (p. 38)
 - Emacs completion and handling accented characters with orderless (p. 46)
 - Snippets (p. 529)
- sachaconsult
 - Quickly search my code (p. 337)
- sachaelisp
 - Prefix for writing functions (p. 507)
 - Easily override existing functions (p. 507)

- [Edebug](#) (p. 513)
- [ERT](#) (p. 513)
- [Buttercup](#) (p. 515)
- [Eldoc](#) (p. 520)
- [Sorting](#) (p. 525)
- [Evaluation](#) (p. 525)
- [Stubbing](#) (p. 527)
- [Collecting Emacs News from Mastodon](#) (p. 564)
- [Combining Mastodon timelines using mastodon.el](#) (p. 569)
- [Using Spookfox to scroll Firefox up and down from Emacs](#) (p. 592)
- [Ledger](#) (p. 674)
- [sacha-emacs](#)
 - [Reload](#) (p. 17)
 - [Repeatable commands](#) (p. 41)
 - [which-key and which-key-posframe](#) (p. 45)
 - [Tangle Emacs config snippets to different files and add boilerplate](#) (p. 250)
 - [Key chords](#) (p. 702)
- [sacha-emacsconf](#)
 - [Prepare for EmacsConf screenshots or recordings](#) (p. 497)
 - [EmacsConf](#) (p. 708)
- [sacha-emacspeak](#)
 - [Emacspeak](#) (p. 704)
- [sacha-embark](#)
 - [Using Embark to act on video](#) (p. 26)
 - [Using Embark to act on audio](#) (p. 26)
 - [Using Embark to offer context-sensitive actions for Org elements](#) (p. 28)
 - [Whichkey and Embark](#) (p. 29)
 - [Embark and images](#) (p. 31)
 - [Embark and subed](#) (p. 36)
 - [Embark and erefactor-rename-symbol-in-buffer](#) (p. 38)
 - [Making it easier to add a category to a blog post](#) (p. 274)
 - [Tip from Omar: embark-around-action-hooks](#) (p. 338)
 - [Act on current message with Embark](#) (p. 608)
- [sacha-eshell](#)
 - [Consult directory navigation](#) (p. 51)
 - [Correctly complete commands in subdirectories](#) (p. 547)

- [sacha-file](#)
 - [Working with or renaming a set of files](#) (p. 498)
- [sacha-fun](#)
 - [Controlling my stream audio from Emacs: background music, typing sounds, and push to talk](#) (p. 654)
 - [Stardew Valley](#) (p. 676)
 - [Diagrams](#) (p. 714)
- [sacha-git](#)
 - [Magit - nice git interface](#) (p. 537)
 - [Finding repos with uncommitted changes](#) (p. 539)
 - [Use difftastic](#) (p. 540)
 - [Checking things out](#) (p. 542)
- [sacha-helm](#)
 - [Helm completion with sacha-helm-org-sketches](#) (p. 431)
 - [Digital index piles with Emacs](#) (p. 444)
- [sacha-hydra](#)
 - [Hydra keyboard shortcuts](#) (p. 41)
 - [Emacs Hydra: Allow completion when I can't remember the command name](#) (p. 44)
- [sacha-image](#)
 - [Embank and images](#) (p. 31)
 - [Converting handwriting to text](#) (p. 32)
 - [Renaming and storing](#) (p. 33)
 - [Org Mode: Create a quick timestamped note and capture a screenshot](#) (p. 297)
 - [Photos](#) (p. 355)
 - [Rotate clockwise or counterclockwise](#) (p. 392)
 - [Emacs: Extract part of an image to another file](#) (p. 393)
 - [Make an image square](#) (p. 396)
 - [Animate highlighting part of an image](#) (p. 396)
 - [Artrage](#) (p. 397)
 - [Interactively recolor a sketch](#) (p. 424)
 - [Rename scanned index cards](#) (p. 441)
 - [Supernote](#) (p. 454)
 - [Manage photos with geeqie](#) (p. 704)
 - [Tools for organizing](#) (p. 713)
- [sacha-irc](#)
 - [Internet Relay Chat](#) (p. 550)

- [Search logs](#) (p. 551)
- [sacha-journal](#)
 - [Journal](#) (p. 345)
 - [Working with journal entries](#) (p. 352)
 - [Tagging journal entries](#) (p. 353)
 - [Moments](#) (p. 356)
 - [Slicing and dicing the journal entries](#) (p. 356)
 - [Easily backfill my journal](#) (p. 440)
- [sacha-ledger](#)
 - [Ledger](#) (p. 674)
- [sacha-mail](#)
 - [Send mail asynchronously](#) (p. 607)
 - [Add comment to blog post](#) (p. 609)
 - [Approve or discard Mailman messages](#) (p. 611)
- [sacha-mastodon](#)
 - [Mastodon](#) (p. 551)
 - [Adding Mastodon toots as comments in my 11ty static blog](#) (p. 553)
 - [mastodon.el: Copy toot content as Org Mode](#) (p. 555)
 - [mastodon.el: Mention people based on regexp](#) (p. 555)
 - [mastodon.el: Collect handles in clipboard \(Emacs kill ring\)](#) (p. 558)
 - [mastodon.el: Copy toot URL after posting; also, copying just this post with 11ty](#) (p. 560)
 - [Storing Mastodon links in Org mode](#) (p. 564)
 - [Collecting Emacs News from Mastodon](#) (p. 564)
 - [Copy Mastodon link for Emacs News](#) (p. 567)
 - [Combining Mastodon timelines using mastodon.el](#) (p. 569)
 - [Following people](#) (p. 574)
 - [Tooting a link to the current post](#) (p. 575)
 - [Compose a Mastodon toot with the current Org subtree](#) (p. 577)
 - [Posting the latest screenshot with mastodon.el](#) (p. 577)
 - [Making it easier to toot my config](#) (p. 580)
 - [Capture](#) (p. 580)
 - [Completion](#) (p. 580)
 - [Copy Mastodon toot URL as author link](#) (p. 582)
 - [Collect my recent toots in an Org file so that I can refile them](#) (p. 582)
 - [Archive toots on my blog](#) (p. 583)
- [sacha-misc](#)

- Using Embark and qrencode to show a QR code for the Org Mode link at point (p. 25)
- Web development (p. 503)
- Alignment (p. 506)
- Emacs and my phone (p. 604)
- sachamultimedia
 - FFmpeg (p. 463)
 - Transcript editing (p. 466)
 - Using word-level timing information when editing subtitles or captions in Emacs (p. 476)
 - Showing captions (p. 485)
 - Org Mode: Insert YouTube video with separate captions (p. 493)
 - Preparing to record YouTube shorts (p. 497)
 - Simple streaming with FFmpeg (p. 652)
 - Controlling my stream audio from Emacs: background music, typing sounds, and push to talk (p. 654)
 - More background music (p. 657)
 - Stream message (p. 657)
 - Playing recordings (p. 657)
 - Stream notes (p. 659)
 - Chapters (p. 660)
 - Try continuous streaming and the Google Speech Recognition API (p. 661)
- sachanavigate
 - Obscure Emacs package appreciation: backup-walker (p. 18)
 - Hydra keyboard shortcuts (p. 41)
 - Tag files (p. 542)
- sachanet
 - SSH and `-daemon` (p. 676)
- sachalog11ty
 - 11ty static site generation (p. 265)
- sachalogaudio
 - Audio (p. 328)
- sachalogbabel
 - Tangle Emacs config snippets to different files and add boilerplate (p. 250)
 - Org Babel: Detangle just the current block (p. 255)
 - Run source blocks in an Org Mode subtree by custom ID (p. 256)
 - Execute named babel block (p. 257)
 - JSON (p. 257)

- [Fix block indentation](#) (p. 258)
- [sacha-org-bookmarks](#)
 - [Bookmarks](#) (p. 246)
- [sacha-org-captions](#)
 - [Captions](#) (p. 330)
- [sacha-org-contacts](#)
 - [mastodon.el: Mention people based on regexp](#) (p. 555)
- [sacha-org-def](#)
 - [Linking to and exporting function definitions in Org Mode](#) (p. 373)
 - [Still allow linking to the file](#) (p. 376)
 - [Including variables](#) (p. 376)
- [sacha-org-export](#)
 - [Changing Org Mode underlines to the HTML mark element](#) (p. 259)
 - [Include inline SVGs in Org Mode HTML and Markdown exports](#) (p. 260)
 - [Org Mode: Export HTML, copy files, and serve the results via simple-httpd so that media files work](#) (p. 262)
 - [Moving my Org post subtree to the 11ty directory](#) (p. 281)
 - [Remove heading from TOC](#) (p. 287)
 - [Add a note to the bottom of blog posts exported from my config file](#) (p. 295)
 - [Cleaning up export](#) (p. 299)
 - [Publish without prompting](#) (p. 300)
 - [Abbreviations](#) (p. 303)
 - [Adding a custom header argument to Org Mode source blocks and using that argument during export](#) (p. 304)
 - [Org Mode: Asynchronous export and tangle of a large file](#) (p. 308)
 - [Plain text](#) (p. 310)
 - [Fix elisp links](#) (p. 341)
 - [Save when Emacs loses focus](#) (p. 371)
- [sacha-org-image](#)
 - [Org Mode sketch: links](#) (p. 427)
- [sacha-org-package](#)
 - [Package links](#) (p. 370)
- [sacha-org-project](#)
 - [Using an Emacs Lisp macro to define quick custom Org Mode links to project files; plus URLs and search](#) (p. 332)
- [sacha-org-protocol](#)
 - [Links from org-protocol](#) (p. 339)

- Org protocol: following Org links from outside Emacs (p. 342)
- [sacha-org-sketch](#)
 - [Save edited text for sketch post](#) (p. 392)
- [sacha-org-treemap](#)
 - [Treemap visualization](#) (p. 359)
- [sacha-org-validate](#)
 - [Validation](#) (p. 381)
 - [Keep only unique headings](#) (p. 381)
 - [No blank titles, no duplicate paths](#) (p. 381)
- [sacha-org-video](#)
 - [Videos](#) (p. 324)
- [sacha-org-vtime](#)
 - [Linking to a specific time in a video](#) (p. 327)
 - [Simplify inserting audio links](#) (p. 472)
- [sacha-org-yt](#)
 - [YouTube](#) (p. 322)
 - [Org Mode: Insert YouTube video with separate captions](#) (p. 493)
- [sacha-org](#)
 - [Using Embark to insert files as Org INCLUDEs](#) (p. 27)
 - [Renaming and storing](#) (p. 33)
 - [Hydra keyboard shortcuts](#) (p. 41)
 - [consult-omni](#) (p. 52)
 - [Using web searches and bookmarks to quickly link placeholders in Org Mode](#) (p. 54)
 - [Inserting code](#) (p. 245)
 - [11ty static site generation](#) (p. 265)
 - [Linking to blog topics](#) (p. 270)
 - [Linking to blog posts](#) (p. 271)
 - [Moving my Org post subtree to the 11ty directory](#) (p. 281)
 - [Include Mastodon, HN, Reddit fields in front matter](#) (p. 285)
 - [Copy Tasker task](#) (p. 287)
 - [Counting words without blocks](#) (p. 288)
 - [Org Mode: Including portions of files between two regular expressions](#) (p. 289)
 - [Copy linked file and change link](#) (p. 295)
 - [Org Mode: Create a quick timestamped note and capture a screenshot](#) (p. 297)
 - [Special blocks](#) (p. 301)
 - [Copy region](#) (p. 306)

- [Convert an inline link into a side note/footnote](#) (p. 314)
- [Adding Org Mode link awesomeness elsewhere: sach-org-insert-link-dwim](#) (p. 314)
- [Links to my config](#) (p. 320)
- [YouTube](#) (p. 322)
- [Copy web link](#) (p. 334)
- [Linking to headings that match a tag](#) (p. 339)
- [Dired](#) (p. 341)
- [Speed command for adding a custom ID to Org Mode posts](#) (p. 343)
- [Counting](#) (p. 362)
- [Spreadsheets](#) (p. 363)
- [Copying and sharing code](#) (p. 363)
- [Tables](#) (p. 363)
- [Invoices](#) (p. 363)
- [Counting words](#) (p. 366)
- [Allow dashes in tags](#) (p. 367)
- [Convert from Markdown](#) (p. 367)
- [Copying information from my phone](#) (p. 368)
- [Reddit](#) (p. 368)
- [Sorting Org Mode lists using a sequence of regular expressions](#) (p. 369)
- [Clipboard](#) (p. 607)
- [Setting properties](#) (p. 372)
- [Org - send things to the bottom of the list](#) (p. 378)
- [Org Mode: Format Libby book highlights exported as JSON](#) (p. 379)
- [Org Mode custom link: copy to clipboard](#) (p. 430)
- [Digital index piles with Emacs](#) (p. 444)
- [Simplify inserting audio links](#) (p. 472)
- [Collecting Emacs News from Mastodon](#) (p. 564)
- [Quantified Awesome](#) (p. 594)
- [Compare times and effort estimates](#) (p. 598)
- [List upcoming tasks so that I can see if I'm overloaded](#) (p. 602)
- [Send currently-clocked task title to file, include in stream](#) (p. 651)
- [Show Emacs-related tasks](#) (p. 657)
- [Comparison-shopping with Org Mode](#) (p. 719)
- [sacha-plover](#)
 - [Looking things up](#) (p. 679)
- [sacha-quantified](#)

- Child time! (p. 596)
- Make a tablist of my time entries (p. 596)
- Time tracking, previous weekly review (p. 601)
- Compare time use (p. 603)
- [sacha-shell](#)
 - Scan `~/bin` and turn the scripts into interactive commands (p. 501)
 - `dwim-shell-command` (p. 535)
 - Automation (p. 546)
- [sacha-similar](#)
 - Vector search (p. 385)
 - Indexing-related code (p. 385)
 - Consult-based interface for searching blog posts (p. 386)
 - Handle sketches too (p. 388)
 - Multiple sources (p. 389)
 - `emacs-rag-search?` (p. 390)
- [sacha-sketch](#)
 - Embark and images (p. 31)
 - Completing sketches (p. 49)
 - Breaking up a PDF from Supernote (p. 397)
 - Finding sketches (p. 422)
 - Renaming and recoloring sketches (p. 424)
 - Button-based interface (p. 432)
 - Templates (p. 434)
 - Get information for sketched books (p. 443)
 - Make it easy to follow up on a sketch (p. 444)
 - Digital index piles with Emacs (p. 444)
 - Sketched books (p. 449)
 - Other sketches (p. 450)
 - Other sketch-related functions (p. 452)
 - Write about half-page scans (p. 453)
 - Supernote (p. 454)
 - org-attaching the latest image from my Supernote via Browse and Access (p. 459)
- [sacha-spookfox](#)
 - Link to current webpage from Spookfox (p. 589)
 - Link to currently-selected text using Spookfox (p. 590)

- Running the current Org Mode Babel Javascript block from Emacs using Spookfox (p. 590)
- Using Spookfox to scroll Firefox up and down from Emacs (p. 592)
- Emacs and Spookfox: org-capture the current tab from Firefox or a link from the page (p. 594)
- Interact with Google Gemini web interface through Spookfox (p. 711)
- `sacha-subed`
 - Embark and subed (p. 36)
 - Interleave images with transcript (p. 464)
 - Split a transcript into phrases for subtitles (p. 465)
 - Transcript editing (p. 466)
 - Adjust subtitles (p. 467)
 - Extract part of a video (p. 468)
 - Hide IDs and times (p. 470)
 - Other subtitle code (p. 471)
 - Simplify inserting audio links (p. 472)
 - Using Emacs to fix automatically generated subtitle timestamps (p. 475)
 - Using word-level timing information when editing subtitles or captions in Emacs (p. 476)
 - Using WhisperX to get word-level timestamps for audio editing with Emacs and subed-record (p. 479)
 - Testing subtitle start times by skimming the first second (p. 483)
 - Edit text (p. 485)
 - Working with sections defined by NOTE comments (p. 488)
 - Split up oops better (p. 490)
 - Export transcript as list (p. 494)
 - Removing gaps and merging subtitles (p. 495)
- `sacha-supernote`
 - Supernote (p. 454)
 - Using Emacs Lisp to export TXT/EPUB/PDF from Org Mode to the Supernote via Browse and Access (p. 458)
 - org-attaching the latest image from my Supernote via Browse and Access (p. 459)
- `sacha-svg-dot-grid`
 - Dot-grid box templates (p. 435)
- `sacha-svg-test`
 - Identifying paths (p. 402)
- `sacha-svg`
 - Identifying paths (p. 402)

- [Linking paths](#) (p. 413)
- [Sorting paths](#) (p. 417)
- [Animating paths in order](#) (p. 419)
- [RevealJS CSS animation of sketches](#) (p. 421)
- [sacha-sync](#)
 - [Syncthing](#) (p. 605)
- [sacha-time](#)
 - [Timestamps](#) (p. 391)
 - [Using the calendar-date-echo-text variable to help plot a heatmap on a year-long calendar in Emacs](#) (p. 598)
 - [Workrave](#) (p. 600)
- [sacha-transcribe](#)
 - [Remove filler words at the start and upcase the next word](#) (p. 484)
 - [Split up oops better](#) (p. 490)
- [sacha-tts](#)
 - [Foot pedal](#) (p. 45)
 - [Speech synthesis \(experimental\)](#) (p. 719)
- [sacha-web](#)
 - [Org Mode: Export HTML, copy files, and serve the results via simple-httpd so that media files work](#) (p. 262)
 - [Adding Org Mode link awesomeness elsewhere: sacha-org-insert-link-dwim](#) (p. 314)
 - [Emacs: Open URLs or search the web, plus browse-url-handlers](#) (p. 584)
 - [Checking URLs](#) (p. 587)
 - [Parsing RSS and Atom feeds](#) (p. 588)
 - [Transforming HTML](#) (p. 616)
 - [Saving images](#) (p. 617)
 - [Cleaning up](#) (p. 621)
 - [Removing unneeded styles](#) (p. 622)
 - [Stream notes](#) (p. 659)