Sacha Chua's Emacs configuration

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. 2) 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. ??)

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 (setq 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. ??) 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. ??). 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. ??). 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. ??) 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:

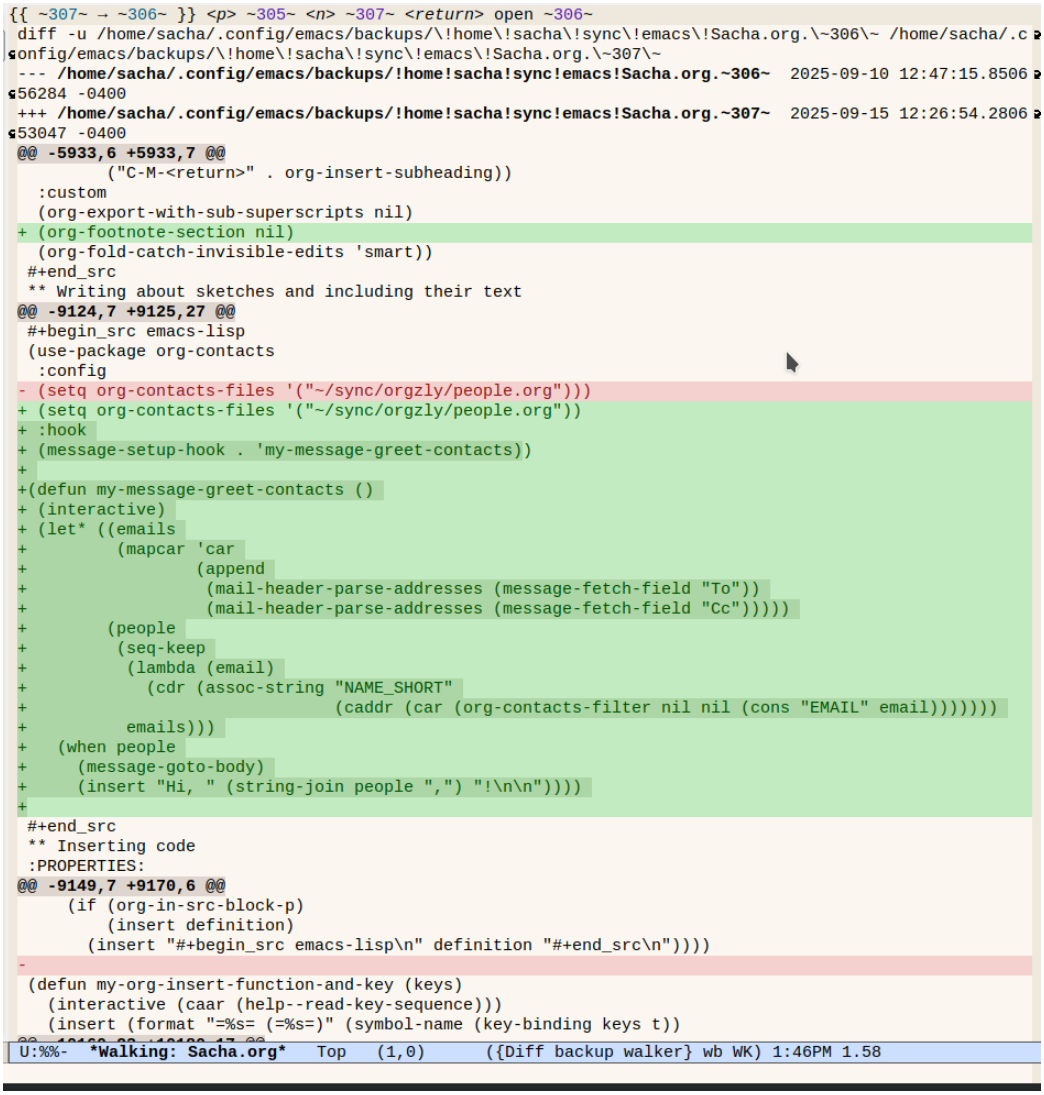

Figure 1: backup-walker diffs going the direction I want them to: additions (+) marked in green, deletions (-) in red

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)))))
```

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] vc-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.

1.10 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)
```