Emacs configuration file

harritaylor

March 8, 2020

Contents

1 Meta / ETC

1.1 Startup timer

```
(add-hook 'emacs-startup-hook
          (lambda ()
            (message "Emacs ready in %s with %d garbage collections."
                     (format "%.2f seconds"
                             (float-time
                              (time-subtract after-init-time before-init-time)))
                     gcs-done)))
```

1.2 GC

Lexical scoping for the init-file is needed, it can be specified in the header. Make startup faster by reducing the frequency of garbage collection. The default is 800 kilobytes. Measured in bytes. These are the first lines of the actual configuration.

```
;;; -*- lexical-binding: t -*-
(setq gc-cons-threshold (* 50 1000 1000))
```

Tangle and compile this file on save automatically:

```
(defun tangle-init ()
  "If the current buffer is 'init.org' the code-blocks are
tangled, and the tangled file is compiled."
  (when (equal (buffer-file-name)
               (expand-file-name (concat user-emacs-directory "init.org")))
    ;; Avoid running hooks when tangling.
    (let ((prog-mode-hook nil))
      (org-babel-tangle)
      (byte-compile-file (concat user-emacs-directory "init.el")))))

(add-hook 'after-save-hook 'tangle-init)
```

This helps get rid of `functions might not be defined at runtime` warnings. See `https://github.com/jwiegley/use-package/issues/590`

```
;; (eval-when-compile
;;   (setq use-package-expand-minimally byte-compile-current-file))
```

1.3 Use package

Initialize package and add Melpa source.

```
(require 'package)
(let* ((no-ssl (and (memq system-type '(windows-nt ms-dos))
                    (not (gnutls-available-p))))
    (proto (if no-ssl "http" "https")))

    (add-to-list 'package-archives (cons "melpa" (concat proto "://melpa.org/packages/"
    ;;(add-to-list 'package-archives (cons "melpa-stable" (concat proto "://stable.melp

    (when (< emacs-major-version 24)
     ;; For important compatibility libraries like cl-lib
      (add-to-list 'package-archives '("gnu" . (concat proto "://elpa.gnu.org/packages/
(package-initialize)
```

Install use-package.

```
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))

(eval-when-compile (require 'use-package))
(setq use-package-always-ensure t)

;; this package is useful for overriding major mode keybindings
(use-package bind-key)
```

1.4 Modifier keys

Emacs control is Ctrl. Emacs Super is Command. Emacs Meta is Alt. Right Alt (option) can be used to enter symbols like em dashes -.

```
(setq mac-right-command-modifier 'super)
(setq mac-command-modifier 'super)

(setq mac-option-modifier 'meta)
(setq mac-left-option-modifier 'meta)
(setq mac-right-option-modifier 'meta)

(setq mac-right-option-modifier 'nil)
```

2 Visuals

```
(setq-default line-spacing 0)

(setq initial-frame-alist '((width . 135) (height . 55)))
(tool-bar-mode -1)
```

Matching parenthesis appearance.

```
(set-face-background 'show-paren-match "wheat")
(set-face-attribute 'show-paren-match nil :weight 'extra-bold)
(show-paren-mode)
```

Simple mode line.

```
(setq column-number-mode t) ;; show columns and rows in mode line
```

Show full path in title bar.

```
(setq-default frame-title-format "%b (%f)")
```

Use spaces instead of tabs.

```
(setq-default indent-tabs-mode nil)
(setq-default c-basic-indent 2)
(setq-default c-basic-offset 2)
(setq-default tab-width 2)
(setq tab-width 2)
(setq js-indent-level 2)
(setq css-indent-offset 2)
(setq c-basic-offset 2)
```

Visual lines.

```
(global-visual-line-mode t)
```

3 Sane defaults

3.1 Basics

Autosave and backup are not useful.

```
(setq make-backup-files nil) ; stop creating backup~ files
(setq auto-save-default nil) ; stop creating #autosave# files
(setq create-lockfiles nil)  ; stop creating .# files
```

Revert (update) buffers automatically when underlying files are changed externally.

```
(global-auto-revert-mode t)
```

Basic things.

```
(setq
 inhibit-startup-message t         ; Don't show the startup message
 inhibit-startup-screen t          ; or screen
 cursor-in-non-selected-windows t  ; Hide the cursor in inactive windows

 echo-keystrokes 0.1               ; Show keystrokes right away, don't show the message
 initial-scratch-message nil       ; Empty scratch buffer
 sentence-end-double-space nil     ; Sentences should end in one space, come on!
 ;; confirm-kill-emacs 'y-or-n-p      ; y and n instead of yes and no when quitting
)

(fset 'yes-or-no-p 'y-or-n-p)      ; y and n instead of yes and no everywhere else
(scroll-bar-mode -1)
(delete-selection-mode 1)
(global-unset-key (kbd "s-p"))
```

Emacs kill ring and system clipboard should be independant.

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

Quickly switch to scratch buffer with `+0`.

```
(global-set-key (kbd "s-0") (lambda ()
                              (interactive)
                              (if (string= (buffer-name) "*scratch*") (previous-buffer)
```

3.2 Which key

```
(use-package which-key
  :config
  (which-key-mode)
  (setq which-key-idle-delay 0.5))
```

3.3 OS Integration

Pass system shell environment to Emacs. This is important primarily for shell inside Emacs, but also things like Org mode export to Tex PDF don't work, since it relies on running external command `pdflatex`, which is loaded from `PATH`.

```
(use-package exec-path-from-shell
  :config
  (when (memq window-system '(mac ns))
    (exec-path-from-shell-initialize)))
```

Use `Cmd+i` to open the current folder in a new tab of Terminal:

```
(defun iterm-goto-filedir-or-home ()
  "Go to present working dir and focus iterm"
  (interactive)
  (do-applescript
   (concat
    " tell application \"iTerm2\"\n"
    "   tell current window\n"
    "     create tab with profile \"Default\"\n"
    "   end tell\n"
    "   tell the current session of current window\n"
    (format "     write text \"cd %s\" \n"
            ;; string escaping madness for applescript
            (replace-regexp-in-string "\\\\" "\\\\\\\\"
                                      (shell-quote-argument (or default-directory "~")))
    "   end tell\n"
    " end tell\n"
```

```
    " do shell script \"open -a iTerm\"\n"
    ))
  )
(global-set-key (kbd "s-i") 'iterm-goto-filedir-or-home)
```

3.4 Navigation and editing

Kill line with `Cmd-Backspace` (thanks to simpleclip, killing doesn't rewrite the system clipboard). Kill one word with `Alt-Backspace`. Also kill forward with `Alt-Shift-Backspace`.

```
(global-set-key (kbd "s-<backspace>") 'kill-whole-line)
(global-set-key (kbd "s-<delete>") 'kill-whole-line)
(global-set-key (kbd "M-S-<backspace>") 'kill-word)
(global-set-key (kbd "M-<delete>") 'kill-word)
(bind-key* "S-<delete>" 'kill-word)
```

Use `cmd` for movement and selection just like in macOS.

```
(global-set-key (kbd "s-<right>") 'end-of-visual-line)
(global-set-key (kbd "s-<left>") 'beginning-of-visual-line)

(global-set-key (kbd "s-<up>") 'beginning-of-buffer)
(global-set-key (kbd "s-<down>") 'end-of-buffer)

(global-set-key (kbd "s-l") 'goto-line)
```

macOS basics.

```
(global-set-key (kbd "s-a") 'mark-whole-buffer)       ;; select all
(global-set-key (kbd "s-s") 'save-buffer)             ;; save
(global-set-key (kbd "s-S") 'write-file)              ;; save as
(global-set-key (kbd "s-q") 'save-buffers-kill-emacs) ;; quit
```

Regular people undo-redo.

```
(use-package undo-fu)
(global-unset-key (kbd "C-z"))
(global-set-key (kbd "C-z")   'undo-fu-only-undo)
(global-set-key (kbd "C-S-z") 'undo-fu-only-redo)
(global-set-key (kbd "s-z")   'undo-fu-only-undo)
(global-set-key (kbd "s-r")   'undo-fu-only-redo)
```

Go back to previous mark (position) within buffer to go back (forward?).

```
(defun my-pop-local-mark-ring ()
  (interactive)
  (set-mark-command t))

(defun unpop-to-mark-command ()
  "Unpop off mark ring. Does nothing if mark ring is empty."
  (interactive)
      (when mark-ring
        (setq mark-ring (cons (copy-marker (mark-marker)) mark-ring))
        (set-marker (mark-marker) (car (last mark-ring)) (current-buffer))
        (when (null (mark t)) (ding))
        (setq mark-ring (nbutlast mark-ring))
        (goto-char (marker-position (car (last mark-ring))))))

(global-set-key (kbd "C-i") 'my-pop-local-mark-ring)
(global-set-key (kbd "C-o") 'unpop-to-mark-command)
```

Move between open buffers with ease.

```
(global-set-key (kbd "s-[") 'previous-buffer)
(global-set-key (kbd "s-]") 'next-buffer)

(defun vsplit-last-buffer ()
  (interactive)
  (split-window-vertically)
  (other-window 1 nil)
  (switch-to-next-buffer))

(defun hsplit-last-buffer ()
  (interactive)
  (split-window-horizontally)
  (other-window 1 nil)
  (switch-to-next-buffer))

(global-set-key (kbd "s-w") (kbd "C-x 0")) ;; just like close tab in a web browser
(global-set-key (kbd "s-W") (kbd "C-x 1")) ;; close others with shift

(global-set-key (kbd "s-'") (kbd "C-x 2"))
```

```
(global-set-key (kbd "s-5") (kbd "C-x 3"))

(global-set-key (kbd "s-K") 'kill-this-buffer)

;; (global-set-key (kbd "s-T") 'vsplit-last-buffer)
;; (global-set-key (kbd "s-t") 'hsplit-last-buffer)
```

Go to other windows easily with one keystroke `s-something` instead of `C-x something`. `Move-text` allows moving lines around with meta-up/down.

```
(eval-after-load "org"
  '(progn (setq org-metaup-hook nil)
   (setq org-metadown-hook nil)))

(use-package move-text
  :config
  (move-text-default-bindings))
```

Smarter open-line: Hit `cmd+return` to insert a new line below the current.

```
(defun smart-open-line ()
  "Insert an empty line after the current line. Position the cursor at its beginning, a
  (interactive)
  (move-end-of-line nil)
  (newline-and-indent))

(defun smart-open-line-above ()
  "Insert an empty line above the current line. Position the cursor at it's beginning,
  (interactive)
  (move-beginning-of-line nil)
  (newline-and-indent)
  (forward-line -1)
  (indent-according-to-mode))

(global-set-key (kbd "s-<return>") 'smart-open-line)
(global-set-key (kbd "s-S-<return>") 'smart-open-line-above)
```

Join lines.

```
(defun smart-join-line (beg end)
  "If in a region, join all the lines in it. If not, join the current line with the nex
```

```
  (interactive "r")
  (if mark-active
      (join-region beg end)
      (top-join-line)))

(defun top-join-line ()
  "Join the current line with the next line."
  (interactive)
  (delete-indentation 1))

(defun join-region (beg end)
  "Join all the lines in the region."
  (interactive "r")
  (if mark-active
      (let ((beg (region-beginning))
            (end (copy-marker (region-end))))
        (goto-char beg)
        (while (< (point) end)
          (join-line 1)))))

(global-set-key (kbd "s-j") 'smart-join-line)
```

Delete trailing spaces and add new line in the end of a file on save.

```
(add-hook 'before-save-hook 'delete-trailing-whitespace)
(setq require-final-newline t)
```

Allow shift selecting in org mode (I don't care about priority indicators).

Multiple cursors are a must. Make `<return>` insert a newline; `multiple-cursors-mode` can still be disabled with `C-g`.

```
(use-package multiple-cursors
  :config
  (setq mc/always-run-for-all 1)
  ;; (global-set-key (kbd "s-d") 'mc/mark-next-like-this)
  ;; (global-set-key (kbd "C-s-g") 'mc/mark-all-dwim)
  (define-key mc/keymap (kbd "<return>") nil)
  (global-set-key (kbd "s-<mouse-1>") 'mc/add-cursor-on-click))
```

Comment lines.

```
(global-set-key (kbd "s-/") 'comment-line)
```

ESC as the universal "get me out of here" command.

```
(define-key key-translation-map (kbd "ESC") (kbd "C-g"))
```

3.5 Windows

Automatic new windows are always on the bottom, not the side.

```
(setq split-height-threshold 0)
(setq split-width-threshold nil)
```

Move between windows with alt-tab

```
(global-set-key (kbd "M-<tab>") (kbd "C-x o"))
```

Shackle to make sure all windows are nicely positioned.

```
(use-package shackle
  :init
  (setq shackle-default-alignment 'below
        shackle-default-size 0.4
        shackle-rules '((help-mode           :align below :select t)
                        (helpful-mode        :align below)
                        (compilation-mode    :select t   :size 0.25)
                        ("*compilation*"     :select nil :size 0.25)
                        ("*ag search*"       :select nil :size 0.25)
                        ("*Flycheck errors*" :select nil :size 0.25)
                        ("*Warnings*"        :select nil :size 0.25)
                        ("*Error*"           :select nil :size 0.25)
                        ("*Org Links*"       :select nil :size 0.1)
                        (magit-status-mode                :align bottom :size 0.5  :inl
                        (magit-log-mode                   :same t                  :inl
                        (magit-commit-mode                :ignore t)
                        (magit-diff-mode     :select nil  :align left   :size 0.5)
                        (git-commit-mode                  :same t)
                        (vc-annotate-mode                 :same t)
                        ))
  :config
  (shackle-mode 1))
```

3.6 Edit indirect

Select any region and edit it in another buffer.

```
(use-package edit-indirect)
```

3.7 Ivy, Swiper and Counsel

Swiper

```
(use-package swiper
  :config
  (global-set-key (kbd "s-f") 'swiper-isearch))
```

Ivy

```
(use-package ivy
  :config
  (ivy-mode 1)
  (setq ivy-use-virtual-buffers t)
  (setq ivy-count-format "(%d/%d) ")
  (setq enable-recursive-minibuffers t)
  (setq ivy-initial-inputs-alist nil)
  (setq ivy-re-builders-alist
      '((swiper . ivy--regex-plus)
        (swiper-isearch . regexp-quote)
        ;; (counsel-git . ivy--regex-plus)
        ;; (counsel-ag . ivy--regex-plus)
        (counsel-rg . ivy--regex-plus)
        (t      . ivy--regex-fuzzy)))   ;; enable fuzzy searching everywhere except fo

  (global-set-key (kbd "s-b") 'ivy-switch-buffer))


(use-package ivy-rich
  :config
  (ivy-rich-mode 1)
  (setq ivy-rich-path-style 'abbrev))
```

Counsel

```
(use-package counsel
  :config
  (global-set-key (kbd "M-x") 'counsel-M-x)
  (global-set-key (kbd "s-y") 'counsel-yank-pop)
  (global-set-key (kbd "C-x C-f") 'counsel-find-file)
  (global-set-key (kbd "s-F") 'counsel-rg)
  (global-set-key (kbd "s-p") 'counsel-git))

;; When using git ls (via counsel-git), include unstaged files
(setq counsel-git-cmd "git ls-files -z --full-name --exclude-standard --others --cache
```

```
(use-package smex)
(use-package flx)
```

4 Git

Magit time

```
  (use-package magit
    :config
    (global-set-key (kbd "s-g") 'magit-status))
  (use-package magit-todos)

  (use-package hl-todo
    :config
    (setq hl-todo-keyword-faces
        '(("TODO"   . "#FF0000")
          ("FIXME"  . "#FF0000")
          ("DEBUG"  . "#A020F0")
          ("GOTCHA" . "#FF4500")
          ("STUB"   . "#1E90FF"))))
```

Navigate to projects with `Cmd+Shift+P`

```
(setq magit-repository-directories '(("\~/Projects/" . 4)))
```

```
(defun magit-status-with-prefix-arg ()
  "Call ‘magit-status‘ with a prefix."
  (interactive)
  (let ((current-prefix-arg '(4)))
    (call-interactively #'magit-status)))

(global-set-key (kbd "s-P") 'magit-status-with-prefix-arg)
```

5 Spell checking

Spell checking requires an external command to be available. Install aspell on your Mac, then make it the default checker for Emacs' ispell. Note that personal dictionary is located at ~/.aspell.LANG.pws by default.

```
(setq ispell-program-name "aspell")
```

Enable spellcehck for all text modes. TODO: disable on start.

```
(add-hook 'text-mode-hook 'flyspell-mode)
(global-set-key (kbd "s-\\") 'ispell-word)
```

6 Thesaurus

Synonym search is `Cmd+Shift+\`. It requires `wordnet`.

```
(use-package powerthesaurus
  :config
  (global-set-key (kbd "s-|") 'powerthesaurus-lookup-word-dwim)
  )
```

Word definition search.

```
(use-package define-word
```

```
:config
(global-set-key (kbd "M-\\") 'define-word-at-point))



;; (read-abbrev-file abbrev-file-name t)
;; (setq-default abbrev-mode t)
```

7 YASnippet

```
(use-package yasnippet
  :config
  (setq yas-snippet-dirs
        '("~/.emacs.d/snippets"))
  (yas-global-mode 1))
```

8 Markdown

Let's see what this does...

```
(use-package markdown-mode
  :mode (("README\\.md\\'" . gfm-mode)
         ("\\.md\\'" . markdown-mode)
         ("\\.markdown\\'" . markdown-mode))
  :init (setq markdown-command "pandoc --no-highlight"))

(eval-after-load 'markdown-mode
  `(define-key markdown-mode-map (kbd "C-s-<down>") 'markdown-narrow-to-subtree))

(eval-after-load 'markdown-mode
  `(define-key markdown-mode-map (kbd "C-s-<up>") 'widen))

(eval-after-load 'markdown-mode
  `(define-key markdown-mode-map (kbd "s-O") (lambda ()
                                               (interactive)
```

```
                                                    (markdown-kill-ring-save)
                                                    (let ((oldbuf (current-buffer)))
                                                      (save-current-buffer
                                                        (set-buffer "*markdown-output*")
                                                        (with-no-warnings (mark-whole-buf
                                                        (simpleclip-copy (point-min) (poi

;; Export without the first line (usually there's a header)
(eval-after-load 'markdown-mode
  `(define-key markdown-mode-map (kbd "M-s-O") (lambda ()
                                                    (interactive)
                                                    (markdown-kill-ring-save)
                                                    (let ((oldbuf (current-buffer)))
                                                      (save-current-buffer
                                                        (set-buffer "*markdown-output*")
                                                        (goto-char (point-min))
                                                        (kill-whole-line)
                                                        (with-no-warnings (mark-whole-buf
                                                        (simpleclip-copy (point-min) (poi
```

9 Programming

9.1 Formatting

Format everything

```
(use-package format-all)
```

10 Frames, windows, buffers

Always open in the same frame

```
(setq ns-pop-up-frames nil)
```

11 Org

Visually indent sections, which looks better for smaller files etc.

```
(setq org-startup-indented t)
(setq org-catch-invisible-edits 'error)
```

```
(setq org-cycle-separator-lines -1)
(setq calendar-week-start-day 1)
(setq org-ellipsis "")
(setq org-support-shift-select t)
```

org files

```
(setq org-directory "~/org")
(setq org-agenda-files '("~/org"))

(setq org-refile-targets (quote ((nil :maxlevel . 9)
                                 (org-agenda-files :maxlevel . 9))))
```

Code block indentation should be correct depending on language, including code highlighting.

```
(setq org-src-tab-acts-natively t)
(setq org-src-preserve-indentation t)
(setq org-src-fontify-natively t)
```

Export to HTML

```
(use-package htmlize)
```

Etc from `https://github.com/freetonik/emacs-dotfiles/blob/master/init.org`

```
  (with-eval-after-load 'org
    ;; no shift or alt with arrows
    (define-key org-mode-map (kbd "<S-left>") nil)
    (define-key org-mode-map (kbd "<S-right>") nil)
    (define-key org-mode-map (kbd "<M-left>") nil)
    (define-key org-mode-map (kbd "<M-right>") nil)
    ;; no shift-alt with arrows
    (define-key org-mode-map (kbd "<M-S-left>") nil)
    (define-key org-mode-map (kbd "<M-S-right>") nil)

    (define-key org-mode-map (kbd "C-s-<left>") 'org-metaleft)
    (define-key org-mode-map (kbd "C-s-<right>") 'org-metaright))
```

```
  (setq org-use-speed-commands t)

  (with-eval-after-load 'org
    (define-key org-mode-map (kbd "C-s-<down>") 'org-narrow-to-subtree)
    (define-key org-mode-map (kbd "C-s-<up>") 'widen))
```

Agenda and capture

```
(global-set-key (kbd "C-c c") 'org-capture)
(global-set-key (kbd "s-=") 'org-capture)
(global-set-key "\C-ca" 'org-agenda)
```

11.1 Latex

```
(require 'ox-latex)
(setq org-format-latex-options (plist-put org-format-latex-options :scale 2.0))
(setq org-highlight-latex-and-related '(latex))
(with-eval-after-load 'ox-latex
  (add-to-list
   'org-latex-classes
   '("tufte-book"

     "\\documentclass{tufte-book}
     \\input{/users/rakhim/.emacs.d/latex/tufte.tex}"
     ("\\part{%s}" . "\\part*{%s}")
     ("\\chapter{%s}" . "\\chapter*{%s}")
     ("\\section{%s}" . "\\section*{%s}")
     ("\\subsection{%s}" . "\\subsection*{%s}")
     ("\\subsubsection{%s}" . "\\subsubsection*{%s}"))))
```