Emacs configuration file

Rakhim Davletkaliyev

March 12, 2019

Contents

1 Credits

Initially inspired by larstvei's setup. Check out EmacsCast, my podcast about Emacs. I talk about my config in Episode 2.

2 Installing

I think it'll be better not to clone and use this config as is, but rather build your own config using mine as a starting point. But if you really want to try it, then follow these steps:

Clone the repo:

```
git clone https://github.com/freetonik/emacs-dotfiles
```

Make a backup of your old `.emacs.d`:

```
mv ~/.emacs.d ~/.emacs.d-bak
```

Rename cloned directory:

```
mv dot-emacs ~/.emacs.d
```

On the first run Emacs will install some packages. It's best to restart Emacs after that process is done for the first time.

3 Configurations

3.1 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")))
    ;; Comment/uncomment these two lines to enable/disable MELPA and MELPA Stable as de
    (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)
```

3.2 Manually installed packages (temporarily disabled)

```
(add-to-list 'load-path "~/.emacs.d/lisp/")
(load "edit-indirect")
```

3.3 Modifier keys

Emacs control is Ctrl. Emacs Super is Command. Emacs Meta is Alt.

```
(setq mac-right-command-modifier 'super)
;; (setq mac-left-option-modifier 'meta)
(setq mac-option-modifier 'meta)
(setq mac-command-modifier 'super)
```

Right Alt (option) can be used to enter symbols like em dashes -.

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

3.4 Meta

When this configuration is loaded for the first time, the `init.el` is the file that is loaded. It looks like this:

```
;; This file replaces itself with the actual configuration at first run.

;; We can’t tangle without org!
(require ’org)
;; Open the configuration
(find-file (concat user-emacs-directory "init.org"))
;; tangle it
(org-babel-tangle)
;; load it
(load-file (concat user-emacs-directory "init.el"))
;; finally byte-compile it
(byte-compile-file (concat user-emacs-directory "init.el"))
```

Lexical scoping for the init-file is needed, it can be specified in the header. This is the first line of the actual configuration:

```
;;; -*- lexical-binding: t -*-
```

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))
```

3.5 Visuals

```
(when (memq window-system ’(mac ns))
  (add-to-list ’default-frame-alist ’(ns-appearance . light)) ;; {light, dark}
  (add-to-list ’default-frame-alist ’(ns-transparent-titlebar . t)))

(global-display-line-numbers-mode)
```

I’ve tried many 3rd party themes, but keep coming back to the default light theme.

```
(load-theme ’tsdh-light)
```

Inconsolata font, remove the cruft and make the initial size bigger.

```
(set-face-attribute ’default nil :font "Inconsolata LGC 13")
(setq-default line-spacing 0)
(setq initial-frame-alist ’((top . 10) (left . 10) (width . 125) (height . 45)))
(tool-bar-mode -1)

;; (require ’paren)
;; (setq show-paren-delay 0)
;; (show-paren-mode 1)
(set-face-background ’show-paren-match "grey84")
;; (set-face-foreground ’show-paren-match nil)
(set-face-attribute ’show-paren-match nil :weight ’extra-bold)
```

Show parens and other pairs.

```
(use-package smartparens
  :config
  (require ’smartparens-config)
  (smartparens-global-mode t)
  (show-smartparens-global-mode t)
  (setq sp-show-pair-delay 0)

  ;; no ’’ pair in emacs-lisp-mode
  (sp-local-pair ’emacs-lisp-mode "’" nil :actions nil)
  (sp-local-pair ’markdown-mode "‘"   nil :actions ’(wrap insert))  ;; only use ‘ for w
  (sp-local-tag ’markdown-mode "s" "‘‘‘scheme" "‘‘‘")
  (define-key smartparens-mode-map (kbd "C-<right>") ’sp-forward-slurp-sexp)
  (define-key smartparens-mode-map (kbd "C-<left>") ’sp-forward-barf-sexp))
```

Wrap lines always.

```
(global-visual-line-mode 1)
```

Nice and simple mode line.

```
(setq column-number-mode t) ;; show columns in addition to rows in mode line
(set-face-attribute 'mode-line nil :background "NavajoWhite")
(set-face-attribute 'mode-line-inactive nil :background "#FAFAFA")
```

Show vi-like tilde in the fringe on empty lines.

```
(use-package vi-tilde-fringe
  :config
  (global-vi-tilde-fringe-mode 1))
```

Show full path in the title bar.

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

Never use tabs, use spaces instead.

```
(setq-default indent-tabs-mode nil)
(setq tab-width 2)

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

Which key is great for learning Emacs, it shows a nice table of possible commands.

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

Disable blinking cursor.

```
(blink-cursor-mode 0)
```

3.6 Sane defaults

I don't care about auto save and backup files.

```
(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)
```

Some 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
 initial-major-mode 'org-mode      ; org mode by default
 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
 ;; help-window-select t              ; select help window so it's easy to quit it wit
)

(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"))
(global-hl-line-mode 0)
```

I want Emacs kill ring and system clipboard to be independent. Simpleclip is the solution to that.

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

3.7 Scrolling

Nicer scrolling behavior.

```
(setq scroll-margin 10
   scroll-step 1
   next-line-add-newlines nil
   scroll-conservatively 10000
   scroll-preserve-screen-position 1)

(setq mouse-wheel-follow-mouse 't)
(setq mouse-wheel-scroll-amount '(1 ((shift) . 1)))
```

3.8 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)

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

A nice little real terminal in a popup.

```
(use-package shell-pop)
```

3.9 Navigation and editing

Kill line with `s-Backspace`, which is `Cmd+Backspace` by default. Note that thanks to Simpleclip, killing doesn't rewrite the system clipboard. Kill one word by `M+Backspace.  Also, kill forward word with =Alt-Shift-Backspace`, since `Alt-Backspace` is kill word backwards.

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

Use `super` (which is `Cmd`) for movement and selection just like in macOS.

```
(global-set-key (kbd "s-<right>") (kbd "C-e"))
(global-set-key (kbd "S-s-<right>") (kbd "C-S-e"))
(global-set-key (kbd "s-<left>") (kbd "M-m"))
(global-set-key (kbd "S-s-<left>") (kbd "M-S-m"))

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

Basic things you should expect from macOS.

```
(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

(global-set-key (kbd "s-z") 'undo)
```

Go back to previous mark (position) within buffer and 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 "s-,") 'my-pop-local-mark-ring)
(global-set-key (kbd "s-.") 'unpop-to-mark-command)
```

Since `Cmd+,` and `Cmd+.` move you back in forward in the current buffer, the same keys with `Shift` move you back and forward between open buffers.

```
(global-set-key (kbd "s-<") 'previous-buffer)
(global-set-key (kbd "s->") 'next-buffer)
```

Go to other windows easily with one keystroke `s-something` instead of `C-x something`.

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

Expand-region allows to gradually expand selection inside words, sentences, etc. `C-'` is bound to Org's `cycle through agenda files`, which I don't really use, so I unbind it here before assigning global shortcut for expansion.

```
(use-package expand-region
  :config
  (global-set-key (kbd "s-'") 'er/expand-region)
  (global-set-key (kbd "s-\"") 'er/contract-region))
;; "
```

`Move-text` allows moving lines around with meta-up/down.

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

Smarter open-line by bbatsov. Once again, I'm taking advantage of CMD and using it to quickly insert new lines above or below the current line, with correct indentation and stuff.

```
(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 whether you're in a region or not.

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

Move around with `Cmd+i/j/k/l`.

```
;; (global-set-key (kbd "s-i") 'previous-line)
;; (global-set-key (kbd "s-k") 'next-line)
;; (global-set-key (kbd "s-j") 'left-char)
;; (global-set-key (kbd "s-l") 'right-char)
```

Upcase word and region using the same keys.

```
(global-set-key (kbd "M-u") 'upcase-dwim)
(global-set-key (kbd "M-l") 'downcase-dwim)
```

Provide nice visual feedback for replace.

```
(use-package visual-regexp
  :config
  (define-key global-map (kbd "s-r") 'vr/replace))
```

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)
```

Multiple cusors 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 "M-s-d") 'mc/edit-beginnings-of-lines)
  (global-set-key (kbd "s-D") 'mc/mark-all-dwim)
  (define-key mc/keymap (kbd "<return>") nil))
```

Comment lines.

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

3.10 Dired

Enable `a` to move into a folder in Dired. This is better than default `Enter`, because `a` doesn't create additional buffers (actually, it kills the buffer and creates a new one).

```
(put 'dired-find-alternate-file 'disabled nil)

(use-package dired
  :ensure nil
  :custom
  (dired-auto-revert-buffer t)
  (dired-dwim-target t)
  (dired-hide-details-hide-symlink-targets nil)
  (dired-listing-switches "-alh")
  (dired-ls-F-marks-symlinks nil)
  (dired-recursive-copies 'always))
```

3.11 Windows

I'm still not happy with the way new windows are spawned. For now, at least, let's make it so that new automatic windows are always created on the bottom, not on the side.

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

Move between windows with Control-Command-Arrow and with `Cmd` just like in iTerm.

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

(use-package windmove
  :config
  (global-set-key (kbd "s-[")  'windmove-left)         ;; Cmd+[ go to left window
  (global-set-key (kbd "s-]")  'windmove-right)        ;; Cmd+] go to right window
  (global-set-key (kbd "s-{")  'windmove-up)           ;; Cmd+Shift+[ go to upper windo
  (global-set-key (kbd "<s-}>")  'windmove-down))      ;; Ctrl+Shift+[ go to down windo
```

Enable winner mode to quickly restore window configurations

```
(winner-mode 1)
(global-set-key (kbd "C-s-[") 'winner-undo)
(global-set-key (kbd "C-s-]") 'winner-redo)
```

Let's try Shackle one more time.

```
(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)
                        (dired-mode          :ignore t)

                        (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 0.2)

                        (neotree-mode                    :align left)
                        (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)
                        ("^\\*git-gutter.+\\*$" :regexp t :size 15 :noselect t)
                        ))
  :config
  (shackle-mode 1))
  ;; (defun my/shackle-defaults (plist)
  ;;   "Ensure popups are always aligned and selected by default. Eliminates the need
  ;;  for :align t on every rule."
  ;;   (when plist
  ;;     (unless (or (plist-member plist :align)
  ;;                 (plist-member plist :same)
  ;;                 (plist-member plist :frame))
  ;;       (plist-put plist :align t))
  ;;     (unless (or (plist-member plist :select)
  ;;                 (plist-member plist :noselect))
  ;;       (plist-put plist :select t)))
  ;;   plist)
  ;; (advice-add #'shackle--match :filter-return #'my/shackle-defaults)

  ;; (add-hook 'my/after-init-hook 'shackle-mode))
```

3.12 Edit indirect

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

3.13 Projectile (disabled)

Install Projectile.

```
(use-package projectile
  :config
  (setq projectile-enable-caching t)
  (define-key projectile-mode-map (kbd "s-P") 'projectile-command-map)
  (projectile-mode +1))
```

3.14 Helm (disabled)

```
(use-package helm-swoop)
(use-package helm
  :config
  (require 'helm-config)
  (helm-mode 1)
  (helm-autoresize-mode 1)
  (setq helm-follow-mode-persistent t)
  (global-set-key (kbd "M-x") 'helm-M-x)
  (setq helm-M-x-fuzzy-match t)
  (setq helm-buffers-fuzzy-matching t)
  (setq helm-recentf-fuzzy-match t)
  (setq helm-apropos-fuzzy-match t)
  (setq helm-split-window-inside-p t)
  ;; (global-set-key (kbd "M-y") 'helm-show-kill-ring)
  ;; (global-set-key (kbd "s-b") 'helm-mini)
  ;; (global-set-key (kbd "C-x C-f") 'helm-find-files)
  ;; (global-set-key (kbd "s-f") 'helm-swoop)
  )
(setq helm-swoop-pre-input-function
      (lambda () ""))

(use-package helm-projectile
  :config
  (helm-projectile-on))

(use-package helm-ag
  :config
  (global-set-key (kbd "s-F") 'helm-projectile-ag))

(global-set-key (kbd "s-p") 'helm-projectile-find-file)
```

3.15 Ivy, Swiper and Counsel

```
(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)
        (t      . ivy--regex-fuzzy)))   ;; enable fuzzy searching everywhere except fo

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

(use-package swiper
  :config
  ;; (global-set-key "\C-s" 'swiper)
  ;; (global-set-key "\C-r" 'swiper)
  (global-set-key (kbd "s-f") 'swiper))

(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-ag)
  (global-set-key (kbd "s-p") 'counsel-git))

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

Ivy-rich make Ivy a bit more friendly by adding information to ivy buffers, e.g. description of commands in `M-x`, meta info about buffers in `ivy-switch-buffer`, etc.

```
(use-package ivy-rich
  :config
  (ivy-rich-mode 1)
  (setq ivy-rich-path-style 'abbrev)) ;; To abbreviate paths using abbreviate-file-name
```

3.16 Counsel integration for Projectile (disabled)

```
(use-package counsel-projectile
  :config
  (counsel-projectile-mode 1)
  (global-set-key (kbd "s-F") 'counsel-projectile-ag)
  (global-set-key (kbd "s-p") 'counsel-projectile))

(setq projectile-completion-system 'ivy)
```

3.17 Git

It's time for Magit!

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

And show changes in the gutter (fringe).

```
(use-package git-gutter
  :config
  (global-git-gutter-mode 't)
  (set-face-background 'git-gutter:modified 'nil) ;; background color
  (set-face-foreground 'git-gutter:added "green4")
  (set-face-foreground 'git-gutter:deleted "red"))
```

3.18 NeoTree

```
(use-package neotree
  :config
  (setq neo-window-width 32
        neo-create-file-auto-open t
        neo-banner-message nil
        neo-mode-line-type 'neotree
        neo-smart-open t
        neo-show-hidden-files t
        neo-mode-line-type 'none
        neo-auto-indent-point t)
  (setq neo-theme (if (display-graphic-p) 'nerd 'arrow))
  (global-set-key (kbd "s-B") 'neotree-toggle))
```

3.19 Spellchecking

Spellchecking 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 spellcheck on the fly for all text modes. This includes org, latex and LaTeX.

```
(add-hook 'text-mode-hook 'flyspell-mode)
;; (add-hook 'prog-mode-hook 'flyspell-prog-mode)
```

Spellcheck current word.

```
(global-set-key (kbd "s-\\") 'ispell-word)
```

3.20 Thesaurus

Spellcheck was `Cmd+\`, synonym search is `Cmd+Shift+\`.

```
(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))
```

3.21 Auto completion

```
(use-package company
  :config
  (setq company-idle-delay 0.1)
  (setq company-global-modes '(not org-mode markdown-mode))
  (setq company-minimum-prefix-length 1)
  (add-hook 'after-init-hook 'global-company-mode))
```

3.22 Packages for programming

Here are all the packages needed for programming languages and formats.

```
(use-package yaml-mode)
(use-package markdown-mode
  :commands (markdown-mode gfm-mode)
  :mode (("README\\.md\\'" . gfm-mode)
         ("\\.md\\'" . markdown-mode)
         ("\\.markdown\\'" . markdown-mode))
  :init (setq markdown-command "pandoc"))
(use-package haml-mode)
(use-package dumb-jump
  :config
  (dumb-jump-mode))
```

Clojure.

```
(use-package clojure-mode)
(use-package cider)

(use-package clj-refactor)
(defun my-clojure-mode-hook ()
    (clj-refactor-mode 1)
    (yas-minor-mode 1) ; for adding require/use/import statements
    ;; This choice of keybinding leaves cider-macroexpand-1 unbound
    (cljr-add-keybindings-with-prefix "C-c C-m"))
(add-hook 'clojure-mode-hook #'my-clojure-mode-hook)
```

Web mode.

```
(use-package web-mode
  :mode ("\\.html\\'")
  :config
  (setq web-mode-markup-indent-offset 2))
```

Emmet.

```
(use-package emmet-mode
  :commands emmet-mode
  :init
  (setq emmet-indentation 2)
  (setq emmet-move-cursor-between-quotes t)
  :config
  (add-hook 'sgml-mode-hook 'emmet-mode) ;; Auto-start on any markup modes
  (add-hook 'web-mode-hook  'emmet-mode)
  (add-hook 'css-mode-hook  'emmet-mode)) ;; enable Emmet's css abbreviation.
```

3.23 Frames, windows, buffers

```
(defun close-all-buffers ()
  (interactive)
  (mapc 'kill-buffer (buffer-list)))

;; (set-frame-name "EDIT")
;; (make-frame '((name . "ORG")))

;; (progn
;; (make-frame '((name . "TERM")))
;;   (select-frame-by-name "EDIT")
;;   (multi-term))
;; (make-frame '((name . "ORG")))

;; (global-set-key (kbd "s-1") (lambda () (interactive) (select-frame-by-name "EDIT")))
;; (global-set-key (kbd "s-2") (lambda () (interactive) (select-frame-by-name "TERM")))
;; (global-set-key (kbd "s-3") (lambda () (interactive) (select-frame-by-name "ORG")))
```

4 Org

Visually indent sections. This looks better for smaller files.

```
(use-package org
  :config
  (setq org-startup-indented t))
```

Store all my org files in `~/Google Drive/Knowledgebase/org`.

```
(setq org-directory "~/Google Drive/Knowledgebase/org")
```

And all of those files should be in included agenda.

```
(setq org-agenda-files '("~/Google Drive/Knowledgebase/org"))
```

Refile targets should include files and down to 9 levels into them.

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

Allow shift selection with arrows. This will not interfere with some built-in shift+arrow functionality in Org.

```
(setq org-support-shift-select t)
```

While writing this configuration file in Org mode, I have to write code blocks all the time. Org has templates, so doing `<s TAB` creates a source code block. Here I create a custom template for emacs-lisp specifically. So, `<el TAB` creates the Emacs lisp code block and puts the cursor inside.

```
(eval-after-load 'org
  '(progn
    (add-to-list 'org-structure-template-alist '("el" "#+BEGIN_SRC emacs-lisp \n?\n#+E
    (define-key org-mode-map (kbd "C-'") nil)
    (global-set-key "\C-ca" 'org-agenda)))
```

And inside those code blocks indentation should be correct depending on the source language used and have code highlighting.

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

State changes for todos and also notes should go into a Logbook drawer:

```
(setq org-log-into-drawer t)
```

I keep my links in `links.org`, export them to HTML and access them via browser. This makes the HTML file automatically on every save.

```
(defun org-mode-export-links ()
  "Export links document to HTML automatically when 'links.org' is changed"
  (when (equal (buffer-file-name) "/Users/rakhim/Google Drive/Knowledgebase/org/links.
    (progn
      (org-html-export-to-html)
      (message "HTML exported"))))

(add-hook 'after-save-hook 'org-mode-export-links)
```

Quickly open todo and config files.

```
(global-set-key (kbd "\e\ec") (lambda () (interactive) (find-file "~/.emacs.d/init.org
(global-set-key (kbd "\e\em") (lambda () (interactive) (find-file "~/Google Drive/Know
(global-set-key (kbd "\e\el") (lambda () (interactive) (find-file "~/Google Drive/Know
(global-set-key (kbd "\e\eb") (lambda () (interactive) (find-file "~/code/rakhim.org/c
(global-set-key (kbd "\e\ef") (lambda () (interactive) (counsel-ag nil "~/Google Drive
```

Org capture.

```
(global-set-key (kbd "C-c c") 'org-capture)

(setq org-cycle-separator-lines 1)
```

Add closed date when todo goes to DONE state.

```
(setq org-log-done 'time)
```

Not sure about this... I want to retain Shift-Alt movement and selection everywhere, but in Org mode these bindings are important built ins, and I don't know if there is a viable alternative.

Consider switching meta-left/right to `C-c C-,` and `C-c C-.`. These are used to promote and demote subtrees.

```
(add-hook 'org-mode-hook
          (lambda()
            (progn
              (local-unset-key (kbd "M-<right>"))  ;; promoting
              (local-unset-key (kbd "M-<left>"))   ;; and demoting subtrees still work

              (local-unset-key (kbd "S-<right>"))
              (local-unset-key (kbd "S-<left>"))

              (local-unset-key (kbd "M-S-<right>"))
              (local-unset-key (kbd "M-S-<left>")) ;; select by word

              (local-set-key (kbd "C-c C-,") 'org-metaleft)
              (local-set-key (kbd "C-c C-.") 'org-metaright)
              )))

;; 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-c C-,") 'org-metaleft)
(define-key org-mode-map (kbd "C-c C-.") 'org-metaright)
```

Enable speed keys to manage headings without arrows.

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

4.1 Capture templates

```
(setq org-capture-templates
      (quote (
              ;; (("t"
              ;;   "TODO"
              ;;   entry
              ;;   (file+olp "inbox.org" "Tasks")
              ;;   "* TODO %?\n%U\n%a\n")

              ("n"
                "Note"
                entry
                (file+olp "main.org" "Notes Inbox")
                "* %?\n%U\n%a\n")
              ("j"
                "Journal"
                entry
                (file+datetree "journal.org")
                "* %U\n%?"))
              ))
```

4.2 Pandoc exporter

```
(use-package ox-pandoc)
```

4.3 Blogging with hugo

Install `ox-hugo` and enable auto export.

```
(use-package ox-hugo
  :after ox)

(use-package ox-hugo-auto-export :ensure nil :after ox-hugo)
```

Org Capture template to quickly create posts and generate slugs.

```
;; Populates only the EXPORT_FILE_NAME property in the inserted headline.
(require 'ox-hugo)
;; define variable to get rid of 'reference to free variable' warnings
(defvar org-capture-templates nil)
```

```
(with-eval-after-load 'org-capture
  (defun org-hugo-new-subtree-post-capture-template ()
    "Returns `org-capture' template string for new blog post.
See `org-capture-templates' for more information."
    (let* ((title (read-from-minibuffer "Post Title: ")) ;Prompt to enter the post titl
           (fname (org-hugo-slug title)))
      (mapconcat #'identity
                 `(
                   ,(concat "* TODO " title)
                   ":PROPERTIES:"
                   ,(concat ":EXPORT_FILE_NAME: " fname)
                   ":END:"
                   "%?\n")          ;Place the cursor here finally
                 "\n")))

  (defun org-hugo-new-subtree-post-capture-template-comic ()
    "Returns `org-capture' template string for new comic post.
See `org-capture-templates' for more information."
    (let* ((title (read-from-minibuffer "Comic Title: ")) ;Prompt to enter the post tit
           (fname (read-from-minibuffer "Image Filename: "))
           (cnumber (number-to-string (length (org-map-entries nil nil '("/Users/rakhim

      (mapconcat #'identity
                 `(
                   ,(concat "* TODO " title)
                   ":PROPERTIES:"
                   ,(concat ":EXPORT_FILE_NAME: " fname)
                   ,(concat ":EXPORT_HUGO_SLUG: " cnumber)
                   ":END:"
                   "%?\n")          ;Place the cursor here finally
                 "\n")))

  (add-to-list 'org-capture-templates
               '("b"
                 "Blog post at rakhim.org"
                 entry
                 (file+olp "/Users/rakhim/code/rakhim.org/content-org/blog.org" "Blog")
                 (function org-hugo-new-subtree-post-capture-template)))
  (add-to-list 'org-capture-templates
               '("c"
                 "Comic at rakhim.org"
                 entry
                 (file+olp "/Users/rakhim/code/rakhim.org/content-org/honestly-undefin
                 (function org-hugo-new-subtree-post-capture-template-comic))))
```

4.4 Slim HTML export

slimhtml is an emacs org mode export backend. It is a set of transcoders for common org elements which outputs minimal HTML.

```
(use-package htmlize)
(use-package ox-slimhtml)

(defun org-html-export-as-slimhtml
(&optional async subtreep visible-only body-only ext-plist)
  (interactive)
  (org-export-to-buffer 'slimhtml "*slimhtml*"
    async subtreep visible-only body-only ext-plist (lambda () (html-mode))))

;; (global-set-key (kbd "s-O") (lambda () (interactive) (org-html-export-as-slimhtml n

(global-set-key (kbd "s-O") (lambda ()
                              (interactive)
                              (org-html-export-as-slimhtml nil nil nil t)
                              (mark-whole-buffer)
                              (simpleclip-copy (point-min) (point-max))
                              (delete-window)))

;; (org-export-to-buffer 'slimhtml "*slimhtml*")
```

5 Customizations

Store custom-file separately, don't freak out when it's not found.

```
(setq custom-file "~/.emacs.d/custom.el")
(load custom-file 'noerror)
```