Emacs configuration file

Rakhim Davletkaliyev

August 10, 2018

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 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)
```

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/org/links.org")
    (progn
(org-html-export-to-html)
(message "HTML exported"))))

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

3.2 Visuals

I'm trying out Inconsolata as my code font. Also, remove the cruft and make the initial size bigger.

```
(set-frame-font "Inconsolata LGC 15")
(load-theme 'tsdh-light)
(setq initial-frame-alist '((top . 150) (left . 300) (width . 125) (height . 45)))
(tool-bar-mode -1)
```

Highlight parens without delay.

```
(setq show-paren-delay 0)
(show-paren-mode 1)
```

Wrap lines always.

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

And show line numbers.

```
(global-linum-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)
```

3.3 Sane defaults

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

```
(setq auto-save-default nil)
(setq make-backup-files nil)
```

y and n are enough.

```
(fset 'yes-or-no-p 'y-or-n-p)
```

Don't show the startup message or screen, show keystrokes right away, don't show the message in the scratch buffer, org-mode by default, sentences end with a single space, wrap lines, disable the scroll bar, answer y or n when quitting Emacs, hide the scroll bar, unset Cmd-p (I never ever want to print), and delete text when typing over selection.

```
(setq
  inhibit-startup-message t
  inhibit-startup-screen t
  echo-keystrokes 0.1
  initial-scratch-message nil
  initial-major-mode 'org-mode
  sentence-end-double-space nil
  confirm-kill-emacs 'y-or-n-p)

(visual-line-mode 1)
(scroll-bar-mode -1)
(delete-selection-mode 1)
(global-unset-key (kbd "s-p"))
(global-hl-line-mode 1)
```

~~I'm still not sure which way to go with the keys on macOS... The only truly comfortable CTRL position in the left command, but this will breaks muscle memory for some useful, but not-Emacsy things like CMD+c/v/x/z, CMD+s and CMD+a.~~

~~I'll try this setup for now.~~

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

The section above is no longer relevant: I decided to get back to the default keymap configuration and take advantage of the hyper key which is bound to Cmd by default. Caps Lock is now Control on the OS level (in macOS it's done via Preferences), and right command is also control (with the help of Karabiner Elements). Now I can use Cmd+C/V/X/Z/S, and it's also used for helm and projectile things. See other `s-bla` bindings later in this config.

3.4 Scrolling

Nicer scrolling behaviour.

```
(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)))
```

3.5 Packages

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 a
    (add-to-list 'package-archives (cons "melpa" (concat proto "://melpa.org/package
    ;;(add-to-list 'package-archives (cons "melpa-stable" (concat proto "://stable.r
    (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))
```

Install try to try packages

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

Nyan mode is essential

```
(use-package nyan-mode
  :ensure t
  :commands nyan-mode
  :config
  (nyan-mode))
```

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
  :ensure t
  :commands exec-path-from-shell-initialize
  :config
    (when (memq window-system '(mac ns x))
      (exec-path-from-shell-initialize)))
```

Expand-region is great, it 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
  :ensure t)
```

Install Helm and set some keybindings. Note that I use `helm-occur` to search current buffer. (Note: here I `require` helm before using `use-package` to get rid of the warning `functions might not be defined at runtime`.

```
(require 'helm)
(use-package helm
  :ensure t
  :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)
  (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-occur))
```

Install Projectile.

```
(require 'projectile)
(use-package projectile
  :ensure t
  :config
  (define-key projectile-mode-map (kbd "s-p") 'projectile-command-map)
  (projectile-mode +1)
  )
```

And make Helm play nice with Projectile.

```
(require 'helm-projectile)
(use-package helm-projectile
  :ensure t
  :config
  (helm-projectile-on))
```

Ag is great for fast project-wide searching. Note that `ag-helm` is only an interface. The actual Silversearcher must be installed on the OS level. See https://github.com/ggreer/the$_{\text{silversearcher}}$.

```
(use-package helm-ag
  :ensure t
  :config
  (global-set-key (kbd "s-F") 'helm-projectile-ag))
```

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

```
(use-package simpleclip
  :ensure t
  :commands
  (simpleclip-mode)
  :config
  (simpleclip-mode 1))
```

It's time for Magit!

```
(use-package magit
  :ensure t
  :config
  (global-set-key (kbd "s-m") 'magit-status))
```

Beacon is a light that follows your cursor around so you don't lose it!

```
(require 'beacon)
(use-package beacon
  :ensure t
  :config
  (beacon-mode 1))
```

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

```
(require 'which-key)
(use-package which-key
  :ensure t
  :config
  (which-key-mode)
  (setq which-key-idle-delay 0.6))
```

Spellchecking requires an external command to be available. Install `aspell` on your Mac, then make it the default checker for Emacs' `ispell`.

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

3.5.1 Packages for programming

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

Yaml stuff.

```
(use-package yaml-mode
  :ensure t)
```

3.6 Basic 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.

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

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>") (kbd "M-v"))
(global-set-key (kbd "s-<down>") (kbd "C-v"))
```

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

```
(global-set-key (kbd "s-o") (kbd "C-x o"))
(global-set-key (kbd "s-1") (kbd "C-x 1"))
(global-set-key (kbd "s-2") (kbd "C-x 2"))
(global-set-key (kbd "s-3") (kbd "C-x 3"))
(global-set-key (kbd "s-3") (kbd "C-x 3"))
(global-set-key (kbd "s-0") (kbd "C-x 0"))
(global-set-key (kbd "s-w") (kbd "C-x 0"))
(global-set-key (kbd "s-n") (kbd "C-x 2"))
```

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
  (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 beginnin
  (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)
```

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)
```

4 Org

Store all my org files in ~/org.

```
(setq org-directory "~/org")
```

And all of those files should be in included agenda.

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

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#-
    (define-key org-mode-map (kbd "C-'") nil)
    (global-set-key "\C-ca" 'org-agenda)
    (global-set-key (kbd "s-'") 'er/expand-region)))
```

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

```
(setq org-edit-src-content-indentation 0)
(setq org-src-tab-acts-natively t)
(setq org-src-preserve-indentation t)

(setq org-src-fontify-natively t)
```

I often need to export from Org to Markdown, this enables the markdown exporter backend.

```
(custom-set-variables
  '(org-export-backends (quote (ascii html icalendar latex md odt))))
```

When Emacs starts, I want to see my Main org file instead of the scratch buffer.

```
(find-file "~/org/main.org")
```

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

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

Quickly open todo and config files with Esc-Esc-letter.

```
(global-set-key (kbd "\e\em") (lambda () (interactive) (find-file "~/org/main.org")))
(global-set-key (kbd "\e\ec") (lambda () (interactive) (find-file "~/.emacs.d/init.o
(global-set-key (kbd "\e\el") (lambda () (interactive) (find-file "~/org/links.org")
```

I like to put one empty line between headers. By default, Org-mode doesn't show those lines when collapsing.

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

Org-bullets are just nice.

```
(require 'org-bullets)
(use-package org-bullets
  :ensure t
  :config
  (add-hook 'org-mode-hook (lambda () (org-bullets-mode 1))))
```