# Emacs Configuration

Brad Mitchell

April 8, 2019

## Contents

# 1 How to use

The `.emacs` file in home directory should only contain

```
(load-file "~/emacs-config/bootstrap.el")
```

where the path to bootstrap.el is adjusted to wherever you cloned this repo. This will also automatically pull the latest version of the config on startup, unless you forbid it with

```
(setq config-no-auto-update t)
```

# 2 Add Package Repositories

```
(require 'package)
(add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t)
(add-to-list 'package-archives '("org" . "http://orgmode.org/elpa/") t)
(package-initialize)
```

# 3 Set meta key to command

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

# 4 Security

Inspired by ogbe.net and Your Text Editor Is Malware.

## 4.1 Check TLS

```
(setq tls-checktrust t)
(setq gnutls-verify-error t)
```

## 4.2 TODO Do more with local certs and check bad ssl

# 5 Automatic package installation

## 5.1 Install `use-package`

And its dependencies if needed.

```
(mapc
 (lambda (package)
   (if (not (package-installed-p package))
       (progn
         (package-refresh-contents)
         (package-install package))))
 '(use-package diminish bind-key))
```

## 5.2 Trigger `use-package`

And force the install of missing packages.

```
(eval-when-compile
  (require 'use-package))
(require 'diminish)
(require 'bind-key)
(setq use-package-always-ensure t)
```

# 6 Configure Helm

## 6.1 Make everything fuzzy and also rebind functions.

```
(use-package flx)
(use-package helm-flx)
(use-package helm
```

```
  :demand
  :diminish helm-mode
  :bind (("M-x" . helm-M-x)
         ("M-y" . helm-show-kill-ring)
         ("C-x b" . helm-mini)
         ("C-x C-f" . helm-find-files)
         ("C-x r l" . helm-bookmarks)
         ("C-c s" . helm-occur)
         :map helm-find-files-map ;; I like these from Ido
         ("C-<tab>"         . helm-execute-persistent-action)
         ("C-<backspace>" . helm-find-files-up-one-level))
  :config
  (helm-mode 1)
  (helm-flx-mode +1)
  (setq helm-M-x-fuzzy-match t)
  (setq helm-locate-fuzzy-match t)
  (setq helm-lisp-fuzzy-completion t)
  (setq helm-bookmark-show-location t))
```

### 6.2 Imenu + Helm with `imenu-anywhere`

But I find pulling from other buffer confusing so I limit to current buffer

```
(defun imenu-anywhere-same-buffer-p (current other)
  (eq current other))

(use-package imenu-anywhere
  :bind (("C-c C-i" . helm-imenu-anywhere))
  :config (setq imenu-anywhere-buffer-filter-functions
                '(imenu-anywhere-same-buffer-p)))
```

### 6.3 Install `helm-system-packages`

```
(use-package helm-system-packages)
```

## 7 Version Control

For now it's just git, but can add more systems as necessary.

### 7.1 Enable magit

Best git client

```
(use-package magit
  :bind ("C-x g" . magit-status))
```

### 7.2 Enable `forge`

For interactions with github.

```
(use-package forge)
```

### 7.3 Add `git-timemachine`

```
(use-package git-timemachine)
```

## 8 Set personal information

```
(setq user-full-name "Brad Mitchell"
      calendar-latitude 40.232
      calendar-longitude -86.752
      calendar-location-name "Nashville, TN")
```

## 9 Utility functions

### 9.1 Generate scratch buffer

```
(defun generate-scratch-buffer ()
  "Create and switch to a temporary scratch buffer with a random
     name."
  (interactive)
  (switch-to-buffer (make-temp-name "scratch-")))
```

### 9.2 Sudo the current buffer

```
(defun sudo ()
  "Use TRAMP to ‘sudo’ the current buffer"
  (interactive)
  (when buffer-file-name
    (find-alternate-file
     (concat "/sudo:root@localhost:"
```

```
            buffer-file-name))))
```

### 9.3 Show xkcd on start

```
(use-package xkcd)
(defun showxkcd ()
  "Call this to show xkcd comic of the day on start"
  (require 'xkcd)
  (xkcd)
  (switch-to-buffer "*xkcd*"))
```

### 9.4 Replace JSON web token in buffer

This is regexp based

```
(defun replace-token (token)
  "Replace JSON web token for requests"
  (interactive "sEnter the new token: ")
  (save-excursion
    (goto-char (point-min))
    (while (re-search-forward "Bearer .*\"" nil t)
      (replace-match (concat "Bearer " token "\"")))))
```

### 9.5 Open all marked files in Dired

I like this better than the one in `dired+`

```
(eval-after-load "dired"
  '(progn
     (define-key dired-mode-map "F" 'my-dired-find-file)
     (defun my-dired-find-file (&optional arg)
       "Open each of the marked files, or the file under the point, or when prefix arg
       (interactive "P")
       (let* ((fn-list (dired-get-marked-files nil arg)))
         (mapc 'find-file fn-list)))))
```

### 9.6 Open the current file in browser.

Thanks to `https://github.com/purcell/emacs.d/blob/master/lisp/init-utils.el#L78`

```
(defun browse-current-file ()
  "Open the current file as a URL using ‘browse-url’."
  (interactive)
  (let ((file-name (buffer-file-name)))
    (if (and (fboundp 'tramp-tramp-file-p)
             (tramp-tramp-file-p file-name))
        (error "Cannot open tramp file")
      (browse-url (concat "file://" file-name)))))
```

### 9.7 XML Format function

This works well on short text, too much and it can block the system

```
(use-package sgml-mode)

(defun reformat-xml ()
  (interactive)
  (save-excursion
    (sgml-pretty-print (point-min) (point-max))
    (indent-region (point-min) (point-max))))
```

### 9.8 Refill paragraphs to be on one line

```
(defun refill-paragraphs ()
  "fill individual paragraphs with large fill column"
  (interactive)
  (let ((fill-column 100000))
    (fill-individual-paragraphs (point-min) (point-max))))
```

### 9.9 Copy filename and path to clipboard

Thanks to `http://emacsredux.com/blog/2013/03/27/copy-filename-to-the-clipboard/`

```
(defun copy-filename ()
  "Copy the current buffer file name to the clipboard."
  (interactive)
  (let ((filename (if (equal major-mode 'dired-mode)
                      default-directory
                    (buffer-file-name))))
    (when filename
      (kill-new filename)
      (message "Copied buffer file name '%s' to the clipboard." filename))))
```

### 9.10 Align docstring

I put a double space in between what I want to align and call this:

```
(defun align-docstring ()
  "Align lines by double space"
  (interactive)
  (align-regexp (region-beginning) (region-end) "\\(\\s-*\\)  " 1 1 t))
```

### 9.11 Rename local variable

```
(defun rename-local-var (name)
  (interactive "sEnter new name: ")
  (let ((var (word-at-point)))
    (mark-defun)
    (replace-string var name nil (region-beginning) (region-end))))
```

### 9.12 Increment/decrement number at point

I miss that from Vim

```
(defun increment-number-at-point ()
  (interactive)
  (skip-chars-backward "0-9")
  (or (looking-at "[0-9]+")
      (error "No number at point"))
  (replace-match (number-to-string (1+ (string-to-number (match-string 0))))))

(defun decrement-number-at-point ()
  (interactive)
  (skip-chars-backward "0-9")
  (or (looking-at "[0-9]+")
      (error "No number at point"))
  (replace-match (number-to-string (- (string-to-number (match-string 0)) 1))))
```

### 9.13 Comment a line

Before Emacs 25.1

```
(defun comment-line ()
  (interactive)
  (save-excursion
```

```
    (end-of-line)
    (set-mark (point))
    (beginning-of-line)
    (if (comment-only-p (region-beginning) (region-end))
        (uncomment-region (region-beginning) (region-end))
      (comment-region (region-beginning) (region-end)))))
```

### 9.14 Quickly edit this config file

```
(defun edit-config-file ()
  (interactive)
  (find-file (concat config-load-path "configuration.org")))
```

### 9.15 Send the current selection in an email

This uses mutt.

```
(defun email-selection ()
  (interactive)
  (copy-region-as-kill (region-beginning) (region-end))
  (let ((tmp-file (concat "/tmp/" (buffer-name (current-buffer))))
        (recipient (read-string "Enter a recipient: "))
        (subject (read-string "Enter a subject: ")))
    (find-file tmp-file)
    (yank)
    (save-buffer)
    (kill-buffer (current-buffer))
    (shell-command (concat "mutt -s \"" subject "\" " recipient " < " tmp-file))
    (shell-command (concat "rm -f " tmp-file)))
  (message "Sent!"))
```

### 9.16 Move files more intuitively

```
(defun move-file ()
  "Write this file to a new location, and delete the old one."
  (interactive)
  (let ((old-location (buffer-file-name)))
    (call-interactively #'write-file)
    (when old-location
      (delete-file old-location))))
```

### 9.17 Insert a filename at point

```
(defun insert-filename ()
  (interactive)
  (insert (read-file-name "File:")))
```

### 9.18 Insert a relative filename at point

```
(defun insert-relative-filename ()
  (interactive)
  (insert (file-relative-name (read-file-name "File: "))))
```

### 9.19 Format long function parameter list into multiline

```
(defun format-function-parameters ()
  "Turn the list of function parameters into multiline."
  (interactive)
  (beginning-of-line)
  (search-forward "(" (line-end-position))
  (newline-and-indent)
  (while (search-forward "," (line-end-position) t)
    (newline-and-indent))
  (end-of-line)
  (c-hungry-delete-forward)
  (insert " ")
  (search-backward ")")
  (newline-and-indent))
```

### 9.20 Eshell here

Thanks to Howard `https://github.com/howardabrams/dot-files/blob/master/emacs-eshell.org`

```
(defun eshell-here ()
  "Opens up a new shell in the directory associated with the
    current buffer's file. The eshell is renamed to match that
    directory to make multiple eshell windows easier."
  (interactive)
  (let* ((height (/ (window-total-height) 3)))
    (split-window-vertically (- height))
    (other-window 1)
    (eshell "new")
```

```
    (insert (concat "ls"))
    (eshell-send-input)))

(bind-key "C-!" 'eshell-here)
```

### 9.21 Show pwd relative to current project

And copy to clipboard

```
(defun relative-pwd ()
  (interactive)
  (let* ((prj (cdr (project-current)))
         (current-file buffer-file-truename)
         (prj-name (file-name-as-directory (file-name-nondirectory (directory-file-name
         (output (concat prj-name (file-relative-name current-file prj))))
    (kill-new output)
    (message output)))
```

### 9.22 Add JIRA ticket number to commit messages

```
(add-hook 'git-commit-setup-hook
    '(lambda ()
        (let ((has-ticket-title (string-match "^[A-Z]+-[0-9]+"
                                  (magit-get-current-branch)))
              (words (s-split-words (magit-get-current-branch))))
          (if has-ticket-title
              (insert (format "[%s-%s] " (car words) (car (cdr words))))))))
```

### 9.23 Get my public IP

```
(defun what-is-my-ip ()
  (interactive)
  (message "IP: %s"
           (with-current-buffer (url-retrieve-synchronously "https://api.ipify.org")
             (buffer-substring (+ 1 url-http-end-of-headers) (point-max)))))
```

## 10 Custom key bindings

### 10.1 Quickly revert a buffer

Useful if file changed on disk

```
(define-key global-map (kbd "C-c r") 'revert-buffer)
```

### 10.2 Quickly evaluate a buffer or a region

```
(define-key global-map (kbd "C-c x") 'eval-buffer)
(define-key global-map (kbd "C-c X") 'eval-region)
```

### 10.3 Use the Mac Style Home/End keys

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

### 10.4 Quickly turn on auto-fill

```
(global-set-key (kbd "C-c q") 'auto-fill-mode)
```

### 10.5 Hungry delete forward available everywhere

```
(global-set-key (kbd "C-c d") 'c-hungry-delete-forward)
```

### 10.6 Increment number easily

```
(global-set-key (kbd "C-c +") 'increment-number-at-point)
(global-set-key (kbd "C-c -") 'decrement-number-at-point)
```

### 10.7 Comment the current line

```
(global-set-key (kbd "C-x C-;") 'comment-line)
```

## 11 Org-mode

### 11.1 Set environment

```
(use-package f)
(use-package org)
(setq org-directory "~/org/")
(setq org-agendafiles '("~/org""~/org/reference/"))
```

### 11.2 Utility functions

```
(defun org-file-path (filename)
  "Return the absolute address of an org file, given its relative name."
  (concat (file-name-as-directory org-directory) filename))

(defun org-find-file ()
  "Leverage Helm to quickly open any org files."
  (interactive)
  (find-file (org-file-path (helm-comp-read "Select your org file: " (directory-files
```

### 11.3 Use syntax highlighting in source blocks while editing

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

### 11.4 Set a dark background for source blocks

```
(require 'color)
(if (display-graphic-p)
    (set-face-attribute 'org-block nil :background
                        (color-darken-name
                         (face-attribute 'default :background) 3)))
```

### 11.5 Setup Org Agenda

```
(define-key org-mode-map (kbd "C-c a") 'org-agenda)
```

### 11.6 Setup Org Capture

```
(setq org-default-notes-file (concat org-directory "/notes.org"))
(define-key global-map "\C-cc" 'org-capture)
```

### 11.7 Add more states

```
(setq org-todo-keywords
      '((sequence "TODO(t)" "WAIT(w@/!)" "|" "DONE(d!)" "CANCELED(c@)")))
(setq org-todo-keyword-faces
           '(("WAIT" . "yellow")
             ("CANCELED" . (:foreground "blue" :weight bold))))
```

### 11.8 Enable `flyspell`

```
(add-hook 'org-mode-hook 'flyspell-mode)
(add-hook 'text-mode-hook 'flyspell-mode)
```

### 11.9 Setup `org-babel`

Get additional languages? Load them all

```
(org-babel-do-load-languages
 (quote org-babel-load-languages)
 (quote ((emacs-lisp . t)
         (dot . t)
         (plantuml . t)
         (python . t)
         (gnuplot . t)
         (shell . t)
         (ledger . t)
         (org . t)
         (latex . t)
         (haskell . t))))
```

### 11.10 Enable `org-bullets`

Make org files a bit more readable

```
(use-package org-bullets
  :config (add-hook 'org-mode-hook (lambda () (org-bullets-mode 1))))
```

### 11.11 Display images inline

```
(setq org-startup-with-inline-images t)
```

### 11.12 Register more exports

```
(require 'ox-md)
```

### 11.13 Setup quick access to org files

```
(global-set-key (kbd "<f5>") 'org-find-file)
```

### 11.14 Use `org-journal`

```
(use-package org-journal
  :custom (org-journal-dir "~/org/journal" "Set journal location"))
```

## 12 Register RSS feeds

Uses `elfeed` and `elfeed-org`

```
(use-package elfeed
  :bind ("C-x w" . elfeed))

(use-package elfeed-org
  :config
  (setq rmh-elfeed-org-files (list (concat config-load-path "elfeed.org")))
  (elfeed-org))
```

The last line uses the elfeed.org file to register the feeds.

## 13 Startup behavior

### 13.1 Toggle fullscreen by default

```
(toggle-frame-maximized)
```

### 13.2 Disable the big fat toolbars

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

### 13.3 Disable the scroll bar

```
(scroll-bar-mode -1)
```

### 13.4 Disable splash screen

And set it in emacs-lisp mode

```
(setq inhibit-startup-message t)
(setq initial-major-mode 'emacs-lisp-mode)
```

### 13.5 Empty Scratch buffer

```
(setq initial-scratch-message nil)
```

### 13.6 Show xkcd comic of the day on start

Only on mac or Linux as windows support isn't there yet. Disabled for now because too slow on start.

```
(cond
 ((string-equal system-type "darwin") ; Mac OS X
  (progn
    (showxkcd)))
 ((string-equal system-type "gnu/linux") ; linux
  (progn
    (showxkcd))))
```

## 14 Window behavior

### 14.1 Disable the bell

Aweful atrocious noise on Windows

```
(setq visible-bell 1)
```

### 14.2 Adjust scrolling behavior

```
(setq mouse-wheel-scroll-amount '(1 ((shift) . 1))) ;; one line at a time
(setq mouse-wheel-progressive-speed nil) ;; don't accelerate scrolling
(setq auto-window-vscroll nil)
```

### 14.3 Always ask for confirmation before quitting

```
(setq confirm-kill-emacs 'y-or-n-p)
```

### 14.4 Highlight the current line

```
(when window-system
  (global-hl-line-mode))
```

### 14.5 Use the back/forward mouse keys

```
(global-set-key [mouse-8] 'switch-to-prev-buffer)
(global-set-key [mouse-9] 'switch-to-next-buffer)
```

## 15 Better defaults

Inspired from `https://github.com/technomancy/better-defaults`

### 15.1 Replace dabbrev with hippie-expand

```
(use-package dabbrev
  :diminish abbrev-mode)
(global-set-key (kbd "M-/") 'hippie-expand)
```

### 15.2 Replace isearch with regexp search

```
(global-set-key (kbd "C-s") 'isearch-forward-regexp)
(global-set-key (kbd "C-r") 'isearch-backward-regexp)
(global-set-key (kbd "C-M-s") 'isearch-forward)
(global-set-key (kbd "C-M-r") 'isearch-backward)
```

### 15.3 Save all backup files to a common folder

```
(setq backup-directory-alist `(("." . ,(concat user-emacs-directory
                                               "backups"))))
```

### 15.4 Replace 'yes/no' by just 'y/n

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

### 15.5 Auto reload tags

```
(setq tags-revert-without-query 1)
```

### 15.6 Disable warning for `narrow-to=region`

```
(put 'narrow-to-region 'disabled nil)
```

# 16 Load Cool Theme

```
(use-package dracula-theme
  :config (load-theme 'dracula t)
  (set-face-background 'mode-line "#510370")
  (set-face-background 'mode-line-inactive "#212020"))
```

# 17 Customize the mode-line

## 17.1 A small trim of the original

```
(setq-default mode-line-format '("%e"
                                 mode-line-front-space
                                 " "
                                 mode-line-modified
                                 " "
                                 "%[" mode-line-buffer-identification "%]"
                                 "   "
                                 "L%l"
                                 "  "
                                 mode-line-modes
                                 mode-line-misc-info
                                 projectile-mode-line
                                 " "
                                 (:propertize " " display ((space :align-to (- right 14
                                 (vc-mode vc-mode)
                                 mode-line-end-spaces))
```

## 17.2 Load Tarsius' minions

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

# 18 Fonts

Use the Hack font from chrissimpkins

```
(if (condition-case nil
        (x-list-fonts "Hack")
      (error nil))
```

```
    (progn
      (add-to-list 'default-frame-alist '(font . "Hack-10"))
      (set-face-attribute 'default nil :font "Hack-10")))
```

## 19 Terminal Configuration

Trying to make it adapt to the OS. There is surely a better way to do this.

```
(if (eq system-type 'windows-nt)
    (progn
      (setenv "PATH" (concat "C:\\cygwin64\\bin\\"
                             path-separator
                             (getenv "PATH")))
      )
  (progn
    (use-package exec-path-from-shell
      :config (exec-path-from-shell-copy-env "PATH"))
    )
  )
```

## 20 Dictionary Configuration

Because Windows sucks I have to do this to use `flyspell`

```
(if (eq system-type 'windows-nt)
    (progn
      (add-to-list 'exec-path "C:/Aspell/bin/")
      (setq ispell-program-name "aspell")
      (require 'ispell)
      )
  )
```

## 21 Snippets

I use `yasnippet` a lot.

```
(use-package yasnippet
  :diminish yas-minor-mode
  :config (yas-global-mode 1))
```

And also my package `org-sync-snippets` to keep my snippets into a single file under version control

```
(use-package org-sync-snippets
  :config (setq org-sync-snippets-org-snippets-file
                (concat (file-name-as-directory config-load-path) "snippets.org")))
```

# 22 Code Format

## 22.1 Default tab and indetation

```
(setq-default indent-tabs-mode nil)
(setq-default tab-width 4)
(setq tab-width 4)
```

## 22.2 Add a new line at the end of files

```
(setq require-final-newline t)
```

## 22.3 Delete trailing white spaces on save

```
(add-hook 'before-save-hook 'delete-trailing-whitespace)
```

## 22.4 Set Unix file coding system

```
(setq-default buffer-file-coding-system 'utf-8-unix)
(setq-default default-buffer-file-coding-system 'utf-8-unix)
(set-default-coding-systems 'utf-8-unix)
(prefer-coding-system 'utf-8-unix)
```

## 22.5 Automatically indent yanked code

Thanks to magnars

```
(defvar yank-indent-modes '(php-mode js2-mode)
  "Modes in which to indent regions that are yanked (or yank-popped)")

(defvar yank-advised-indent-threshold 1000
  "Threshold (# chars) over which indentation does not automatically occur.")

(defun yank-advised-indent-function (beg end)
  "Do indentation, as long as the region isn't too large."
```

```
  (if (<= (- end beg) yank-advised-indent-threshold)
      (indent-region beg end nil)))

(defadvice yank (after yank-indent activate)
  "If current mode is one of 'yank-indent-modes, indent yanked text (with prefix arg do
  (if (and (not (ad-get-arg 0))
           (--any? (derived-mode-p it) yank-indent-modes))
      (let ((transient-mark-mode nil))
        (yank-advised-indent-function (region-beginning) (region-end)))))

(defadvice yank-pop (after yank-pop-indent activate)
  "If current mode is one of 'yank-indent-modes, indent yanked text (with prefix arg do
  (if (and (not (ad-get-arg 0))
           (member major-mode yank-indent-modes))
      (let ((transient-mark-mode nil))
        (yank-advised-indent-function (region-beginning) (region-end)))))

(defun yank-unindented ()
  (interactive)
  (yank 1))
```

### 22.6 Define comment syntax

```
(setq comment-start "#")
```

### 22.7 Enable `poporg`

So I can comment code from an org-mode buffer

```
(use-package poporg
  :bind (("C-c \"" . poporg-dwim)))
```

### 22.8 Enable `prog-fill`

```
(use-package prog-fill
  :bind (("M-q" . prog-fill)))
```

## 23 Parenthesis Support

```
(show-paren-mode 1)
(electric-pair-mode 1)
```

# 24 Package management

## 24.1 Enable `paradox`

```
(use-package paradox
  :custom
  (paradox-execute-asynchronously t)
  :config
  (paradox-enable))
```

# 25 Internal Doc Enhancement

## 25.1 Setup `helpful` for prettier doc

```
(use-package helpful
  :bind (("C-h f" . helpful-callable)
         ("C-h v" . helpful-variable)
         ("C-h k" . helpful-key)
         ("C-h F" . helpful-function)
         ("C-h C" . helpful-command)))
```

## 25.2 Setup `elisp-demos` for code example

```
(use-package elisp-demos
  :config (advice-add 'helpful-update
                      :after #'elisp-demos-advice-helpful-update))
```

# 26 Navigation

## 26.1 Enable `avy`

```
(use-package avy
  :bind (("C-c SPC" . avy-goto-char-2)
         ("M-g f" . avy-goto-line)
         ("M-g w" . avy-goto-word-1)))
```

## 26.2 Enable `win-switch`

Super nice to switch between frames and buffers

```
(use-package win-switch
  :bind ("C-x o" . win-switch-dispatch)
```

```
  :config
  (setq win-switch-provide-visual-feedback t)
  (setq win-switch-feedback-background-color "purple")
  (setq win-switch-feedback-foreground-color "white")
  (win-switch-setup-keys-default))
```

### 26.3 Enable `which-key`

Very nice if you don't have a cheat sheet at hand

```
(use-package which-key
  :diminish which-key-mode
  :config (which-key-mode 1))
```

### 26.4 Enable `winner-mode`

```
(winner-mode 1)
```

### 26.5 Enable `smooth-scrolling`

But with a margin of 5

```
(use-package smooth-scrolling
  :config
  (smooth-scrolling-mode 1)
  (setq smooth-scroll-margin 5))
```

### 26.6 Enable `neotree`

```
(use-package neotree)
```

### 26.7 Enable `ibuffer`

```
(use-package ibuffer-vc)
(use-package ibuffer-git)
(define-key global-map (kbd "C-x C-b") 'ibuffer)
```

### 26.8 Enable `yascroll`

So much better than the default scroll bar

```
(use-package yascroll
  :config (global-yascroll-bar-mode 1))
```

### 26.9 Enable `minimap`

Not all the time, but handy.

```
(use-package minimap
  :config
  (setq minimap-window-location "right")
  (setq minimap-major-modes '(prog-mode org-mode)))
```

### 26.10 Enable `rotate`

```
(use-package rotate
  :config (global-set-key (kbd "C-|") 'rotate-layout))
```

### 26.11 Enable `anzu`

```
(use-package anzu
  :config (global-anzu-mode +1)
          (setq anzu-mode-lighter ""))
```

### 26.12 Enable `hamburger-menu`

```
(use-package hamburger-menu
  :config (setq mode-line-front-space 'hamburger-menu-mode-line))
```

### 26.13 Enable `origiami`

Great to fold text

```
(use-package origami
  :config
  (global-set-key (kbd "C-c n o") 'origami-open-node)
  (global-set-key (kbd "C-c n c") 'origami-close-node)
  (global-set-key (kbd "C-c n a") 'origami-open-all-nodes)
  (global-set-key (kbd "C-c n u") 'origami-undo)
  (global-set-key (kbd "C-c n n") 'origami-show-only-node)
  (global-set-key (kbd "C-c n TAB") 'origami-recursively-toggle-node))
```

### 26.14 Enable `eyebrowse`

To manage window configuration

```
(use-package eyebrowse
  :config (eyebrowse-mode t))
```

# 27 Edition

## 27.1 Enable `multiple-cursors`

Useful to edit multiple similar lines

```
(use-package multiple-cursors
  :bind (("C-S-c C-S-c" . mc/edit-lines)
         ("C->" . mc/mark-next-like-this)
         ("C-<" . mc/mark-previous-like-this)
         ("C-c C-<" . mc/mark-all-like-this)
         ("C-S-<mouse-1>" . mc/add-cursor-on-click)))
```

## 27.2 Enable `zzz-to-char`

```
(use-package zzz-to-char
  :bind ("M-z" . zzz-up-to-char))
```

## 27.3 Enable `whole-line-or-region`

```
(use-package whole-line-or-region
  :diminish whole-line-or-region-global-mode
  :config (whole-line-or-region-global-mode t))
```

## 27.4 Enable `viking-mode`

And add my personal twist to it.

```
(use-package viking-mode
  :diminish viking-mode
  :config
  (viking-global-mode)
  (setq viking-greedy-kill nil)
  (setq viking-enable-region-kill t)
  (setq viking-kill-functions (list '(lambda()
                                       (if (region-active-p)
                                           (kill-region (region-beginning) (region-end)
                                       (delete-char 1 t)))
                                    '(lambda()
                                       (insert (pop kill-ring)) ;; insert the char back
                                       (kill-new "") ;; start a new entry in the kill-r
                                       (viking-kill-word)
```

```
    (kill-append " " nil)) ;; append the extra space
  'viking-kill-line-from-point
  'viking-kill-line
  'viking-kill-paragraph
  'viking-kill-buffer)))
```

### 27.5 Enable `undo-tree`

```
(use-package undo-tree
  :diminish undo-tree-mode
  :config
  (global-undo-tree-mode t)
  (setq undo-tree-visualizer-diff t))
```

### 27.6 Enable `volatile-highlights`

Sweet minor mode for providing visual feedback

```
(use-package volatile-highlights
  :diminish volatile-highlights-mode
  :config
  (vhl/define-extension 'undo-tree 'undo-tree-yank 'undo-tree-move)
  (vhl/install-extension 'undo-tree)
  (volatile-highlights-mode t))
```

### 27.7 Enable `ciel`

```
(use-package ciel
  :bind (("C-c i" . ciel-ci)
         ("C-c o" . ciel-co)))
```

### 27.8 Enable `fancy-narrow`

And use it to replace normal narrowing functions

```
(use-package fancy-narrow
  :diminish fancy-narrow-mode)
```

# 28 General Code

## 28.1 Enable `projectile`

And get a shorter modeline, thanks to `https://github.com/purcell/emacs.d/blob/master/lisp/init-projectile.el#L10`

```
(use-package ag)
(use-package helm-ag)
(setq projectile-go-function nil) ;; temporary workaround
(use-package projectile
  :config
  (projectile-mode)
  (setq-default projectile-mode-line
   '(:eval
     (if (file-remote-p default-directory)
         " Proj"
       (format " Proj[%s]" (projectile-project-name)))))
  (add-to-list 'projectile-globally-ignored-directories "node_modules"))
```

With a twist of helm

```
(use-package helm-projectile
  :bind (("C-c v" . helm-projectile)
         ("C-c C-v" . helm-projectile-ag)
         ("C-c w" . helm-projectile-switch-project)))
```

## 28.2 Enable `company`

```
(use-package company
  :diminish company-mode
  :config
  (add-hook 'after-init-hook 'global-company-mode)
  (setq company-minimum-prefix-length 2)
  (setq company-dabbrev-downcase nil))
(use-package company-go)
```

## 28.3 Enable `flycheck`

```
(use-package flycheck
  :diminish flycheck-mode
  :config (flycheck-mode 1)
```

```
  (setq flycheck-phpcs-standard "PSR2")
  (add-hook 'python-mode-hook 'flycheck-mode)
  (add-hook 'emacs-lisp-mode-hook 'flycheck-mode)
  (add-hook 'json-mode-hook 'flycheck-mode)
  (add-hook 'rjsx-mode-hook 'flycheck-mode))
```

Add a little helm twist to it

```
(use-package helm-flycheck
  :bind ("C-c f" . helm-flycheck))
```

### 28.4 Enable `electric-operator`

And add a couple of rules for PHP and JS

```
(use-package electric-operator
  :config
  (electric-operator-add-rules-for-mode 'php-mode
                                        (cons " - >" "->"))
  (electric-operator-add-rules-for-mode 'php-mode
                                        (cons " / /" "// "))
  (electric-operator-add-rules-for-mode 'php-mode
                                        (cons " = > " " => "))
  (electric-operator-add-rules-for-mode 'php-mode
                                        (cons " < ?" "<?"))
  (electric-operator-add-rules-for-mode 'js2-mode
                                        (cons " = > " " => "))
  (electric-operator-add-rules-for-mode 'js2-jsx-mode
                                        (cons " = > " " => "))
  (electric-operator-add-rules-for-mode 'rjsx-mode
                                        (cons " = > " " => ")))
```

### 28.5 Enable `dumb-jump`

Great package to jump to definition

```
(use-package dumb-jump
  :config (setq dumb-jump-aggressive nil))
```

### 28.6 Enable `highlight-numbers`

Make numbers in source code more noticeable

```
(use-package highlight-numbers
  :config (add-hook 'prog-mode-hook 'highlight-numbers-mode))
```

# 29 Emacs Lisp

## 29.1 Enable `eldoc`

```
(use-package eldoc
  :diminish eldoc-mode
  :config (add-hook 'emacs-lisp-mode-hook 'eldoc-mode))
```

## 29.2 Enable `rainbow-delimiters`

But only for emacs-lisp

```
(use-package rainbow-delimiters
  :config
  (add-hook 'emacs-lisp-mode-hook 'rainbow-delimiters-mode))
```

## 29.3 Enable `paredit`

```
(use-package paredit
  :config
  (add-hook 'emacs-lisp-mode-hook 'paredit-mode))
```

## 29.4 Enable `eros`

```
(use-package eros
  :config (add-hook 'emacs-lisp-mode-hook 'eros-mode))
```

## 29.5 Enable `simple-call-tree`

```
(use-package simple-call-tree)
```

## 29.6 Enable `suggest`

```
(use-package suggest)
```

# 30 MySQL

## 30.1 Always make MySQL the default product

```
(require 'sql)
(sql-set-product "mysql")
```

## 30.2 Enable `sqlup-mode`

```
(use-package sqlup-mode
  :config (add-hook 'sql-mode-hook 'sqlup-mode))
```

## 30.3 Enable `mysql-to-org`

I love this package

```
(use-package mysql-to-org
  :config
  (add-hook 'sql-mode-hook 'mysql-to-org-mode))
```

# 31 PHP

## 31.1 Enable `php-mode`

And a bunch of hooks with it And set PSR-2 as coding style

```
(use-package php-mode
  :config
  (add-hook 'php-mode-hook 'flycheck-mode)
  (add-hook 'php-mode-hook 'electric-operator-mode)
  (add-hook 'php-mode-hook 'dumb-jump-mode)
  (add-hook 'php-mode-hook 'php-enable-psr2-coding-style))
```

## 31.2 Enable `company-php`

```
(use-package company-php
  :config
  (add-hook 'php-mode-hook 'company-mode)
  (add-hook 'php-mode-hook '(lambda ()
                              (if (not (member 'php-mode company-dabbrev-code-modes))
                                  (add-to-list 'company-dabbrev-code-modes 'php-mode))
```

### 31.3 Enable `php-eldoc`

```
(setq auto-complete-mode nil) ;; Hack while my PR is pending
(use-package php-eldoc
  :diminish eldoc-mode
  :config (add-hook 'php-mode-hook 'php-eldoc-enable))
```

### 31.4 Enable `phpcbf`

```
(use-package phpcbf
  :config (setq phpcbf-standard "PSR2"))
```

## 32 Web

### 32.1 Enable `web-mode`

So much better than `html-mode`

```
(use-package web-mode
  :mode "\\.phtml\\'"
  :mode "\\.volt\\'"
  :mode "\\.html\\'")
```

### 32.2 Enable `emmet-mode`

Adding the necessary hooks

```
(use-package emmet-mode
  :config
  (add-hook 'sgml-mode-hook 'emmet-mode) ;; Auto-start on any markup modes
  (add-hook 'css-mode-hook  'emmet-mode) ;; enable Emmet's css abbreviation.
  )
```

## 33 Javascript

### 33.1 Enable `js2-mode`

```
(use-package js2-mode
  :mode "\\.js\\'"
  :config
  (add-hook 'js2-mode-hook 'electric-operator-mode)
  (add-hook 'js2-mode-hook 'flycheck-mode)
  (setq js2-basic-offset 2))
```

### 33.2 Enable `js2-refactor`

```
(use-package js2-refactor
  :diminish js2-refactor-mode
  :defer t
  :config
  (add-hook 'js2-mode-hook #'js2-refactor-mode)
  (js2r-add-keybindings-with-prefix "C-c C-m"))
```

### 33.3 Enable `js-doc`

```
(use-package js-doc)
```

### 33.4 Enable `rjsx-mode`

Useful for React

```
(use-package rjsx-mode)
```

## 34 Typescript

### 34.1 Enable `tide`

```
(use-package tide)

(defun setup-tide-mode ()
  (interactive)
  (tide-setup)
  (flycheck-mode +1)
  (setq flycheck-check-syntax-automatically '(save mode-enabled))
  (eldoc-mode +1)
  (tide-hl-identifier-mode +1)
  (company-mode +1))

;; formats the buffer before saving
(add-hook 'before-save-hook 'tide-format-before-save)

(add-hook 'typescript-mode-hook #'setup-tide-mode)
```

### 34.2 Enable `tide` for `tsx`

```
(use-package web-mode)
```

```
(add-to-list 'auto-mode-alist '("\\.tsx\\'" . web-mode))
(add-hook 'web-mode-hook
          (lambda ()
            (when (string-equal "tsx" (file-name-extension buffer-file-name))
              (setup-tide-mode))))
;; enable typescript-tslint checker
(flycheck-add-mode 'typescript-tslint 'web-mode)
```

# 35 Go

A lot of the config is based on gocode, godef, goimports and gotags packages that you should install separately.

```
go get -u github.com/nsf/gocode
go get -u github.com/rogpeppe/godef
go get -u golang.org/x/tools/cmd/goimports
go get -u github.com/jstemmer/gotags
go get github.com/matryer/moq
```

## 35.1 Enable `go-mode`

Absolutely necessary if working in Go

```
(use-package go-mode
  :config
  (add-hook 'before-save-hook #'gofmt-before-save)
  (add-hook 'go-mode-hook 'flycheck-mode)
  (add-hook 'go-mode-hook 'dumb-jump-mode)
  (setq go-packages-function 'go-packages-go-list))
```

## 35.2 Enable `company-go`

```
(use-package company-go
  :config
  (add-hook 'go-mode-hook 'company-mode)
  (add-to-list 'company-backends 'company-go))
```

## 35.3 Enable `go-stackstracer`

```
(use-package go-stacktracer)
```

### 35.4 Enable `go-add-tags`

```
(use-package go-add-tags)
```

### 35.5 Enable `go-eldoc`

```
(use-package go-eldoc
  :diminish eldoc-mode
  :config (add-hook 'go-mode-hook 'go-eldoc-setup))
```

### 35.6 Enable `go-gopath`

```
(use-package go-gopath)
```

### 35.7 Enable `go-direx`

```
(use-package go-direx)
```

### 35.8 Enable `gotest`

```
(use-package gotest)
```

### 35.9 Integrate `moq`

Quick custom function to integrate with the moq tool to generate quick mocks

```
(defun moq ()
  (interactive)
  (let ((interface (word-at-point))
        (test-file (concat (downcase (word-at-point)) "_test.go")))
    (shell-command
     (concat "moq -out " test-file " . " interface))
    (find-file test-file)))
```

### 35.10 See test coverage faster

Simple function to see the test coverage of the current open buffer

```
(defun go-coverage-here ()
  (interactive)
  (shell-command "go test . -coverprofile=cover.out")
  (go-coverage "cover.out")
  (rotate:even-horizontal))
```

# 36 Python

## 36.1 Setup `python-mode`

Need to install IPython first with `pip install ipython`. Assuming that it would be at least IPython 5 so ask for `simple-prompt`.

```
(add-hook 'python-mode-hook 'electric-operator-mode)
(setq python-shell-interpreter "ipython"
      python-shell-interpreter-args "--simple-prompt -i")
```

## 36.2 Enable `company-jedi`

```
(use-package company-jedi
  :config (add-to-list 'company-backends 'company-jedi))
```

## 36.3 Custom functions for pipenv

```
(defun pipenv-activate ()
  (interactive)
  (pythonic-activate (shell-command-to-string "pipenv --venv")))

(defun pipenv-deactivate ()
  (interactive)
  (pythonic-deactivate))
```

# 37 Docker

## 37.1 Setup `Dockerfile-mode`

```
(use-package dockerfile-mode
  :mode "Dockerfile\\'")
```

# 38 Logs

## 38.1 Enable `syslog-mode`

```
;; (use-package syslog-mode)
```

## 38.2 Turn on `auto-revert-tail-mode` for log files

```
(add-to-list 'auto-mode-alist '("\\.log\\'" . auto-revert-tail-mode))
```

### 38.3 Quickly check syslog

```
(defun open-syslog ()
  (interactive)
  (find-file "/var/log/syslog")
  ;; (syslog-mode)
  (goto-char (point-max)))
```

## 39 Eshell

### 39.1 Configuration

```
(use-package eshell
  :init
  (setq eshell-scroll-to-bottom-on-input 'all
        eshell-error-if-no-glob t
        eshell-hist-ignoredups t
        eshell-save-history-on-exit t
        eshell-prefer-lisp-functions nil
        eshell-destroy-buffer-when-process-dies t))
```

### 39.2 Fancy prompt

Modified from `https://www.reddit.com/r/emacs/comments/6f0rkz/my_fancy_eshell_prompt/`

```
(setq eshell-prompt-function
      (lambda ()
        (concat
         (propertize "[" 'face `(:foreground "green"))
         (propertize (user-login-name) 'face `(:foreground "red"))
         (propertize "@" 'face `(:foreground "green"))
         (propertize (system-name) 'face `(:foreground "lightblue"))
         (propertize "][" 'face `(:foreground "green"))
         (propertize (format-time-string "%H:%M" (current-time)) 'face `(:foreground "y
         (propertize "][" 'face `(:foreground "green"))
         (propertize (concat (eshell/pwd)) 'face `(:foreground "white"))
         (propertize "]\n" 'face `(:foreground "green"))
         (propertize ">" 'face `(:foreground "green"))
         (propertize (if (= (user-uid) 0) " # " " $ ") 'face `(:foreground "green"))
         )))
```

### 39.3 Define visual commands and subcommands

```
(setq eshell-visual-commands '("htop" "vi" "screen" "top" "less"
                               "more" "lynx" "ncftp" "pine" "tin" "trn" "elm"
                               "vim"))

(setq eshell-visual-subcommands '("git" "log" "diff" "show" "ssh"))
```

### 39.4 Pager setup

```
(setenv "PAGER" "cat")
```

### 39.5 Enable autojump

```
(use-package eshell-autojump)
```

### 39.6 Aliases

```
(defalias 'ff 'find-file)
(defalias 'd 'dired)
```

### 39.7 Extra shell functions

#### 39.7.1 Clear function

```
(defun eshell/clear ()
  (let ((inhibit-read-only t))
    (erase-buffer)))
```

#### 39.7.2 Git

```
(defun eshell/gst (&rest args)
    (magit-status (pop args) nil)
    (eshell/echo))   ;; The echo command suppresses output
```

#### 39.7.3 Bargs and Sargs

Thanks to `http://www.howardism.org/Technical/Emacs/eshell-present.html`

```
(defun eshell/-buffer-as-args (buffer separator command)
  "Takes the contents of BUFFER, and splits it on SEPARATOR, and
runs the COMMAND with the contents as arguments. Use an argument
```

```
‘%’ to substitute the contents at a particular point, otherwise,
they are appended."
  (let* ((lines (with-current-buffer buffer
                  (split-string
                   (buffer-substring-no-properties (point-min) (point-max))
                   separator)))
         (subcmd (if (-contains? command "%")
                     (-flatten (-replace "%" lines command))
                   (-concat command lines)))
         (cmd-str  (string-join subcmd " ")))
    (message cmd-str)
    (eshell-command-result cmd-str)))

(defun eshell/bargs (buffer &rest command)
  "Passes the lines from BUFFER as arguments to COMMAND."
  (eshell/-buffer-as-args buffer "\n" command))

(defun eshell/sargs (buffer &rest command)
  "Passes the words from BUFFER as arguments to COMMAND."
  (eshell/-buffer-as-args buffer nil command))
```

#### 39.7.4 Close

```
(defun eshell/close ()
  (delete-window))
```

### 39.8 Smoother bindings

```
(add-hook 'eshell-mode-hook
          (lambda ()
            (define-key eshell-mode-map (kbd "C-M-a") 'eshell-previous-prompt)
            (define-key eshell-mode-map (kbd "C-M-e") 'eshell-next-prompt)
            (define-key eshell-mode-map (kbd "M-r") 'helm-eshell-history)))
```

### 39.9 Close window on exit

```
(defun eshell-pop--kill-and-delete-window ()
  (unless (one-window-p)
    (delete-window)))

(add-hook 'eshell-exit-hook 'eshell-pop--kill-and-delete-window)
```

# 40 Dired

## 40.1 Enable `dired-x`

```
(require 'dired-x)
```

## 40.2 Use human-readable sizes

```
(setq dired-listing-switches "-alh")
```

# 41 YAML

## 41.1 Enable `yaml-mode`

```
(use-package yaml-mode
  :config
  (add-hook 'yaml-mode-hook 'flycheck-mode)
  (add-hook 'yaml-mode-hook 'flyspell-mode))
```

## 41.2 Enable `flycheck-yamllint`

```
(use-package flycheck-yamllint)
```

## 41.3 Enable indentation highlight

```
(use-package highlight-indentation
  :config
  (set-face-background 'highlight-indentation-face "#8B6090")
  (add-hook 'yaml-mode-hook 'highlight-indentation-mode))
```

# 42 Restclient

## 42.1 Setup `restclient`

```
(use-package restclient
  :mode ("\\.restclient\\'" . restclient-mode))
```

## 42.2 Enable `company-restclient`

```
(use-package company-restclient
  :config (add-to-list 'company-backends 'company-restclient))
```

### 42.3 Integrate to `org-mode`

```
(use-package ob-restclient)
```

## 43 PlantUML

### 43.1 Enable `plantuml-mode`

```
(use-package plantuml-mode)
```

### 43.2 Download and hook up plantuml.jar

```
(let ((plantuml-directory (concat config-load-path "extra/"))
      (plantuml-link "https://superb-dca2.dl.sourceforge.net/project/plantuml/plantuml
  (let ((plantuml-target (concat plantuml-directory "plantuml.jar")))
    (if (not (f-exists? plantuml-target))
        (progn (message "Downloading plantuml.jar")
               (shell-command
                (mapconcat 'identity (list "wget" plantuml-link "-O" plantuml-target) "
               (kill-buffer "*Shell Command Output*")))
    (setq org-plantuml-jar-path plantuml-target)))
```

## 44 Redis

A few utility functions for my local Redis.

```
(defun redis--command (cmd)
  "Format command for the redis-cli.

CMD is the redis command."
  (format "echo \"%s\" | redis-cli" cmd))

(defun redis--select-key ()
  "Return all redis keys as list."
  (let* ((command (redis--command "KEYS *"))
         (keys (split-string (shell-command-to-string command) "\n")))
    (completing-read "Redis key: " keys)))

(defun redis-set-key (key value)
  "Set a key value pair in redis.
```

```
KEY is the key.
VALUE is the value."
  (interactive "sKey: \nsValue: ")
  (shell-command (redis--command (format "SET %s %s" key value))))

(defun redis-get-key ()
  "Get the value for a key in redis."
  (interactive)
  (let ((key (redis--select-key)))
    (message
     "Value: %s"
     (replace-regexp-in-string "\n" "" (shell-command-to-string (redis--command (format

(defun redis-delete-key ()
  "Delete a key from local redis instance."
  (interactive)
  (let ((key (redis--select-key)))
    (shell-command (redis--command (format "DEL %s" key)))
    (message "Key %s deleted." key)))
```

## 45 Extra Packages

No need to configure, just handy to have.

```
(use-package 2048-game)
(use-package isend-mode)
(use-package lorem-ipsum)
(use-package markdown-mode)
(use-package pdf-tools
  :defer t)
(use-package refine)
(use-package request)
(use-package csv-mode)
;; (use-package csharp-mode)
(use-package keychain-environment)
(use-package prodigy)
(use-package vlf)
(use-package helm-flyspell)
(use-package kubel)
(use-package kubernetes-helm)
```

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

# 46 Extra file loading

If I am working on a separate library, I like to have it loaded on start. Just need to place it in the extra folder.

```
(use-package load-dir
  :config (setq load-dirs (concat config-load-path "extra/")))
```

# 47 Key Frequency

Trying the `keyfreq` package to monitor my command usage

```
(use-package keyfreq
  :config
  (keyfreq-mode 1)
  (keyfreq-autosave-mode 1))
```

# 48 Cheat Sheet

Thanks to the `cheatsheet` package, I can quickly see what are my favorite keys bindings

## 48.1 Setup

```
(use-package cheatsheet
  :config
  (setq cheatsheet--group-face '(:foreground "spring green"))
  (setq cheatsheet--key-face '(:foreground "orchid")))
```

## 48.2 Common

```
(cheatsheet-add :group 'Common
                :key "C-c r"
                :description "Revert buffer")
(cheatsheet-add :group 'Common
                :key "C-c q"
                :description "Turn on/off autofill mode")
(cheatsheet-add :group 'Common
```

```
                :key "C-x u"
                :description "Show the undo-tree")
(cheatsheet-add :group 'Common
                :key "M-y"
                :description "Show the kill-ring")
(cheatsheet-add :group 'Common
                :key "C-x r l"
                :description "Show the bookmarks")
(cheatsheet-add :group 'Common
                :key "C-c s"
                :description "Swoop search through buffer")
(cheatsheet-add :group 'Common
                :key "C-x w"
                :description "Read news and RSS feeds")
(cheatsheet-add :group 'Common
                :key "C-c SPC"
                :description "Jump to char")
(cheatsheet-add :group 'Common
                :key "C-c C-d"
                :description "Hungry delete")
(cheatsheet-add :group 'Common
                :key "C-c i"
                :description "Ciel kill")
(cheatsheet-add :group 'Common
                :key "C-M-h"
                :description "Select region")
(cheatsheet-add :group 'Common
                :key "C-c +"
                :description "Increment number at point")
(cheatsheet-add :group 'Common
                :key "C-c -"
                :description "Decrement number at point")
```

### 48.3 Org

```
(cheatsheet-add :group 'Org
                :key "C-c c"
                :description "Org Capture")
(cheatsheet-add :group 'Org
                :key "C-c ^"
```

```
                :description "Org sort")
(cheatsheet-add :group 'Org
                :key "C-c C-t"
                :description "Org cycle todo")
(cheatsheet-add :group 'Org
                :key "C-c '"
                :description "Edit code block")
(cheatsheet-add :group 'Org
                :key "C-c C-c"
                :description "Evaluate source block")
(cheatsheet-add :group 'Org
                :key "C-c C-o"
                :description "Show source block eval result")
```

### 48.4 Projectile

```
(cheatsheet-add :group 'Projectile
                :key "C-c v"
                :description "Find file in project")
(cheatsheet-add :group 'Projectile
                :key "C-c C-v"
                :description "Find word in project")
(cheatsheet-add :group 'Projectile
                :key "C-c C-v C-w"
                :description "Find word under cursor in project")
(cheatsheet-add :group 'Projectile
                :key "C-c w"
                :description "Switch project")
```

### 48.5 Emacs Lisp

```
(cheatsheet-add :group 'Emacs-Lisp
                :key "C-x C-e"
                :description "Evaluate sexp")
(cheatsheet-add :group 'Emacs-Lisp
                :key "C-c x"
                :description "Evaluate buffer")
(cheatsheet-add :group 'Emacs-Lisp
                :key "C-c X"
                :description "Evaluate region")
```

```
(cheatsheet-add :group 'Emacs-Lisp
                :key "C-h f"
                :description "Describe function")
(cheatsheet-add :group 'Emacs-Lisp
                :key "C-h v"
                :description "Describe variable")
```

### 48.6 Code

```
(cheatsheet-add :group 'Code
                :key "C-c C-i"
                :description "List classes and functions in buffer via imenu")
(cheatsheet-add :group 'Code
                :key "C-x g"
                :description "Summon Magit")
(cheatsheet-add :group 'Code
                :key "C-c f"
                :description "Flycheck the buffer")
(cheatsheet-add :group 'Code
                :key "C-x C-;"
                :description "Comment out current line")
```

### 48.7 Multiple cursors

```
(cheatsheet-add :group 'Multiple-Cursors
                :key "C->"
                :description "Mark next like this")
(cheatsheet-add :group 'Multiple-Cursors
                :key "C-<"
                :description "Mark previous like this")
(cheatsheet-add :group 'Multiple-Cursors
                :key "C-c C-<"
                :description "Mark all like this")
(cheatsheet-add :group 'Multiple-Cursors
                :key "C-S-<mouse-1>"
                :description "Add cursor on click")
```

### 48.8 Dumb Jump

```
(cheatsheet-add :group 'Dumb-Jump
                :key "C-M-g"
```

```
                :description "Jump to function definition")
(cheatsheet-add :group 'Dumb-Jump
                :key "C-M-p"
                :description "Jump back to original location")
(cheatsheet-add :group 'Dumb-Jump
                :key "C-M-q"
                :description "Show tooltip")
```

### 48.9 MySQL-to-Org

```
(cheatsheet-add :group 'MySQL-to-Org
                :key "C-c C-m e"
                :description "Evaluate query in region")
(cheatsheet-add :group 'MySQL-to-Org
                :key "C-c C-m p"
                :description "Evaluate the string at point")
(cheatsheet-add :group 'MySQL-to-Org
                :key "C-c C-m s"
                :description "Open scratch buffer")
```

### 48.10 Paredit

```
(cheatsheet-add :group 'Paredit
                :key "C-("
                :description "Slurp left")
(cheatsheet-add :group 'Paredit
                :key "C-)"
                :description "Slurp right")
(cheatsheet-add :group 'Paredit
                :key "C-{"
                :description "Barf left")
(cheatsheet-add :group 'Paredit
                :key "C-}"
                :description "Barf right")
(cheatsheet-add :group 'Paredit
                :key "M-("
                :description "Wrap sexp in ()")
(cheatsheet-add :group 'Paredit
                :key "M-("
                :description "Wrap sexp in ()")
```

```
(cheatsheet-add :group 'Paredit
                :key "M-s"
                :description "Splice sexp out of ()")
```

### 48.11 Origami

```
(cheatsheet-add :group 'Origami
                :key "C-c n o"
                :description "Open node")
(cheatsheet-add :group 'Origami
                :key "C-c n c"
                :description "Close node")
(cheatsheet-add :group 'Origami
                :key "C-c n a"
                :description "Open all nodes")
(cheatsheet-add :group 'Origami
                :key "C-c n u"
                :description "Undo folding")
(cheatsheet-add :group 'Origami
                :key "C-c n n"
                :description "Fold all nodes but current one")
(cheatsheet-add :group 'Origami
                :key "C-c n TAB"
                :description "Toggle node")
```

### 48.12 Eyebrowse

```
(cheatsheet-add :group 'Eyebrowse
                :key "C-c C-w 1"
                :description "Switch to workspace 1")
(cheatsheet-add :group 'Eyebrowse
                :key "C-c C-w 2"
                :description "Switch to workspace 2")
(cheatsheet-add :group 'Eyebrowse
                :key "C-c C-w '"
                :description "Switch to previous workspace")
(cheatsheet-add :group 'Eyebrowse
                :key "C-c C-w \""
                :description "Close current workspace")
```