Emacs configuration file

Dodge W. Coates

January 21, 2018

Contents

1 DONE Introduction **6**

2 DONE Initialize [0/0] **7**

3 DONE General [0/0] **8**

4 DONE Assistance [0/0] **8**
4.1 Cosmetic . . . 8
4.1.1 `linum-mode` LINE:NUMBER:FRINGE . . . 8
4.1.2 Pretty Symbols PRETTY:APPEARANCE:UNICODE . 9
4.1.3 `flycheck-mode` CHECK:FLY . . . 9
4.1.4 `flyspell` SPELL:CHECK:FLY . . . 9
4.1.5 `rainbow-delimiters` PARENTHESES:APPEARANCE . 9
4.1.6 `rainbow-mode` . . . 10
4.2 Auto Completion . . . 10
4.2.1 `company` COMPLETION:POPUPS . . . 10
4.2.2 `auto-complete` COMPLETION:POPUPS . . . 13
4.3 `guide-key` . . . 13
4.4 Dictionary . . . 14

5 DONE Buffer navigation [0/0] **14**
5.1 General . . . 14
5.1.1 Navigation . . . 14
5.2 Narrowing . . . 15
5.2.1 `helm` SEARCH:FUZZY:POPUPS . . . 15
5.2.2 `ido` SEARCH:FUZZY . . . 22
5.2.3 `ivy` SEARCH:FUZZY:POPUPS . . . 22
5.3 Jumping . . . 23

5.3.1 avy JUMP . . . 23
5.3.2 ace JUMP . . . 24
5.3.3 jump-char JUMP . . . 25
5.4 Searching . . . 25
5.4.1 ace-isearch SEARCH:JUMP:FUZZY . . . 25
5.4.2 isearcher . . . 26
5.4.3 occur . . . 27
5.5 Grep . . . 27
5.5.1 rgrep SEARCH:REGEXP . . . 27
5.6 Re-centering . . . 28

6 DONE Windowing [0/0] 29
6.1 Basic Settings . . . 29
6.2 General Appearance . . . 31
6.2.1 Fringe APPEARANCE:GENERAL:WINDOW . . . 31
6.2.2 Cursor CURSOR:IDLE . . . 31
6.3 Finding files . . . 31
6.3.1 dired . . . 32
6.4 Selection . . . 32
6.4.1 ace-window JUMP . . . 32
6.4.2 windmove DIRECTION . . . 33
6.4.3 ace-popup-menu SELECT . . . 33
6.5 Saving/Restoring/Killing . . . 33
6.5.1 workspaces . . . 33
6.5.2 desktop-save SAVE . . . 34
6.5.3 volatile-kill-buffers . . . 34
6.6 Behavior . . . 34
6.6.1 zygospore . . . 34
6.6.2 shackle POPUPS . . . 34

7 DONE Editing [0/0] 36
7.1 Basic Settings . . . 36
7.1.1 Parentheses . . . 38
7.1.2 Useful Packages . . . 39
7.1.3 Key commands . . . 41
7.2 Common . . . 42
7.3 Diffing . . . 42
7.4 Undoing/Redoing . . . 42
7.4.1 undo-tree . . . 42
7.5 Snippet expansion . . . 43

7.5.1 `yasnippet` . . . 43
7.5.2 `hippie-expand` . . . 44
7.6 Non-ASCII symbols . . . 44
7.6.1 Encoding defaults . . . 44
7.6.2 Key translations . . . 45
7.6.3 `abbrev-mode` . . . 45
7.6.4 `char-menu` . . . 45
7.6.5 `math-symbols` . . . 46
7.7 Functions . . . 46
7.7.1 just-one-space . . . 46
7.7.2 unfill-paragraph . . . 46
7.7.3 die-tabs . . . 47
7.7.4 prelude-move-beginning-of-line . . . 47
7.7.5 defadvice kill-ring-save . . . 48
7.7.6 defadvice kill-region . . . 48
7.7.7 defadvice kill-line . . . 48
7.7.8 yank-advised-indent-function . . . 48
7.7.9 defadvice yank . . . 49
7.7.10 defadvice yank-pop . . . 49
7.7.11 indent-buffer . . . 50
7.7.12 prelude-indent-sensitive-modes . . . 50
7.7.13 indent-region-or-buffer . . . 50
7.7.14 prelude-get-positions-of-line-or-region . . . 51
7.7.15 prelude-smart-open-line . . . 51
7.7.16 prelude-smart-open-line-above . . . 51
7.7.17 toggle-comment-on-line . . . 52
7.7.18 add-file-name-to-clipboard . . . 52
7.7.19 duplicate-line . . . 52

8 TODO Software Development [0/1] 53
8.1 General Settings . . . 53
8.1.1 General . . . 53
8.1.2 Editing . . . 54
8.1.3 Navigation . . . 55
8.1.4 Folding . . . 56
8.1.5 Semantic . . . 57
8.1.6 Compilation . . . 57
8.1.7 Debugging . . . 57
8.1.8 EMR . . . 58
8.1.9 JSON . . . 58

8.1.10 REPL/Command Line . . . 58
8.2 Python . . . 59
8.2.1 General . . . 59
8.2.2 `python-mode` . . . 59
8.2.3 `py-autopep8` . . . 62
8.2.4 `elpy-mode` **DISABLED** . . . 62
8.2.5 `anaconda-mode` . . . 63
8.2.6 `ein` . . . 64
8.2.7 `ob-ipython` . . . 65
8.2.8 `company-anaconda` . . . 65
8.2.9 Python Shell . . . 65
8.3 JavaScript . . . 66
8.3.1 `js-comint` . . . 66
8.3.2 `js2-mode` . . . 68
8.3.3 `company-tern` . . . 68
8.4 C/C++ . . . 69
8.5 Lisp . . . 71
8.5.1 General Lisp Settings . . . 72
8.5.2 Emacs Lisp . . . 73
8.5.3 Clojure . . . 74
8.6 R . . . 76
8.7 Bash/shell . . . 76
8.8 HTML . . . 76
8.8.1 sgml-copy-or-kill-tag . . . 76
8.9 Git [2/3] . . . 77
8.9.1 `magit` . . . 77
8.9.2 `gist` . . . 78
8.9.3 **TODO** `git-gutter` **disabled** . . . 78

9 DONE Org Mode [0/0] 78
9.1 General . . . 78
9.2 Appearance . . . 80
9.2.1 Abbrevs . . . 82
9.3 Commands . . . 83
9.3.1 Automatically wrap org-mode markup . . . 83
9.4 Drill . . . 84
9.5 Tasks and States . . . 84
9.6 Agenda . . . 85
9.6.1 Appearance . . . 85
9.6.2 Interface for agenda . . . 86

9.6.3 Delimit prioirites **DISABLED** . . . 89
9.6.4 Update org-agenda-files . . . 90
9.7 Clock . . . 90
9.7.1 tea-time . . . 90
9.8 Babel . . . 91
9.9 Capture . . . 92
9.9.1 Template Components . . . 92
9.9.2 Captures . . . 92
9.10 Footnotes . . . 93
9.11 Exports . . . 93
9.11.1 `ox-latex` . . . 93
9.11.2 html . . . 93
9.11.3 Lots of html . . . 93
9.12 Functions . . . 99
9.12.1 org-path-completion . . . 99
9.12.2 org-capture . . . 99
9.12.3 org-back-to-top-level-heading . . . 99
9.12.4 org-summary-todo . . . 99
9.12.5 jump-to-org-agenda . . . 100
9.12.6 create-tasks-heading . . . 100

10 DONE Emacs as a Auxiliary Interface [0/0] 101
10.1 Terminal . . . 101
10.2 Edit With Emacs (Google Chrome extension) . . . 104
10.3 Gghat . . . 104
10.4 IRC . . . 104
10.4.1 ERC . . . 104
10.5 SSH . . . 105
10.5.1 Tramp . . . 105
10.6 StackExchange . . . 105
10.7 Email . . . 105
10.8 PDF viewing . . . 108
10.9 Google . . . 109
10.9.1 `gnugol` . . . 109
10.10 Chess **USER SPECIFIC SETTING TO SET** . . . 109
10.11 Calculator . . . 110
10.11.1 Usage . . . 110
10.11.2 Algebraic Notation (infix) . . . 110
10.11.3 Undo . . . 111

1 DONE Introduction

Use the following help commands liberally:

1. `C-h f describe-function`

   Select Emacs functions and provide a description of the selected command.

2. `C-h c describe-command`

   Select interactive Emacs functions ("commands") and provide a description of the selected command.

3. `C-h B describe-buffer`

   Open a buffer that describes current buffer, it's active minor modes, major mode, key bindings, etc.

4. `C-h m describe-mode`

   Opens a buffer that enumerates active modes and information on them.

5. `C-h M-k describe-keymap`

   Open a buffer that enumerates the key bindings associated with an Emacs major/minor mode.

6. `C-h p describe-personal-keybindings`

   Describe custom bindings created.

7. `C-h M-l find-function-on-key`

   Search through Emacs Lisp functions, and jump to the definition for the function selected.

8. `C-h M-f describe-file`

   Select files and show information about them.

9. `C-h o describe-option`

   Describe user variables.

10. `C-h C-o describe-option-of-type`

    Describe user variables, but first narrow them by type.

11. `C-h M-c describe-copying`

    Create a buffer that contains the entire GNU Public License.

12. `C-h C-c describe-key-briefly`

    Display a small description of a key-binding's command in the minibuffer.

`C-h f`, then typing "describe" can always be used to browse help functions. Also, M-x can be used if key-bindings are forgotten (e.g., `M-x describe-function`)

Download a package that provides some of the above help functions. `describe-personal-keybindings` is included by `use-package`.

```
(use-package help-fns+)

(global-set-key (kbd "C-h p") 'describe-personal-keybindings)
```

2 DONE Initialize [0/0]

```
(let ((inhibit-message nil))
  (message "Initializing package manager..."))

;; Keep emacs-generated custom settings in a separate file so they don't pollute init.
(setq custom-file (expand-file-name "custom.el" user-emacs-directory))
(load custom-file)
```

I run linux, but this may be useful for those who use OSX. Description:

On OS X, an Emacs instance started from the graphical user interface will have a different environment than a shell in a terminal window, because OS X does not run a shell during the login. Obviously this will lead to unexpected results when calling external utilities like make from Emacs. This library works around this problem by copying important environment variables from the user's shell. `https://github.com/purcell/exec-path-from-shell`

```
(if (eq system-type 'darwin)
  (unless (package-installed-p 'exec-path-from-shell)
    (package-install 'exec-path-from-shell)))
```

Make sure Emacs gets my bashrc environment variable:

```
(let ((path (shell-command-to-string ". ~/.bashrc; echo -n $PATH")))
  (setenv "PATH" path)
```

```
  (setq exec-path
        (append
         (split-string-and-unquote path ":")
         '("~/.local/bin/flake8")
         exec-path)))
```

Let Emacs know about some of my Emacs Lisp code

```
(add-to-list 'load-path (concat lisp-dir "goodies"))  ;; personal goodies
(add-to-list 'load-path (concat lisp-dir "org-misc")) ;; org-misc functions
```

3 DONE General [0/0]

Diminish allows selectively removing certain mode "nicknames" from the modeline. Useful for keeping the modeline from getting too cluttered.

```
(use-package diminish)
```

This code keeps the semantic idle parser from running the CPU too hot.

```
(setq semantic-idle-scheduler-setup-timer 10000) ; hack. Semantic idle scheduler using
```

Start up message.

```
(let ((inhibit-message nil))
  (message "Configuring general settings..."))
```

This binding is just annoying, so remove it.

```
(global-unset-key (kbd "ESC ESC ESC"))
```

Make `emacs-lisp-mode` the default major mode.

```
(setq default-major-mode 'emacs-lisp-mode)
```

4 DONE Assistance [0/0]

4.1 Cosmetic

4.1.1 `linum-mode` LINE:NUMBER:FRINGE

`linum-mode` is a line-numbering mode that comes packaged with Emacs.

4.1.2 Pretty Symbols PRETTY:APPEARANCE:UNICODE

Prettify certain symbols and symbol combinations. For example, use a instead of `lambda` in Emacs Lisp code.

```
(global-prettify-symbols-mode t)
```

Un-prettify when symbol is just before point.

```
(setq prettify-symbols-unprettify-at-point 'right-edge)
```

4.1.3 `flycheck-mode` CHECK:FLY

`flycheck-mode` provides highlighting mistakes for syntax errors. A lot like Microsoft Word's red squiggly lines that are used to point out grammar/spelling errors.

```
(use-package flycheck
  :init (progn
          (add-hook 'after-init-hook #'global-flycheck-mode))
  :bind (("C-c ! n" . flycheck-next-error)
         ("C-c ! p" . flycheck-previous-error)
         ("C-c ! h" . helm-flycheck))
  :config
  (setq-default flycheck-disabled-checkers '(emacs-lisp-checkdoc))
  :diminish 'flycheck-mode)
```

4.1.4 `flyspell` SPELL:CHECK:FLY

`flyspell-mode` is flycheck, but for spelling. Useful in `org-mode` and other basic text modes.

```
(use-package flyspell
  :config
  :diminish 'flyspell-mode)
```

4.1.5 `rainbow-delimiters` PARENTHESES:APPEARANCE

Automatically color parentheses pairs different colors with rainbow-delimiters. This mode is particularly useful in Lisp modes, where there are many, many, many parentheses.

```
(use-package rainbow-delimiters
  :init
  (rainbow-delimiters-mode)
  :config
  (set-face-attribute 'rainbow-delimiters-depth-1-face nil :foreground "indian red")
  (set-face-attribute 'rainbow-delimiters-depth-2-face nil :foreground "light sea gree
  (set-face-attribute 'rainbow-delimiters-depth-3-face nil :foreground "orchid")
  (set-face-attribute 'rainbow-delimiters-depth-4-face nil :foreground "goldenrod")
  (set-face-attribute 'rainbow-delimiters-depth-5-face nil :foreground "olive drab")
  (set-face-attribute 'rainbow-delimiters-depth-6-face nil :foreground "deep sky blue"
  (set-face-attribute 'rainbow-delimiters-depth-7-face nil :foreground "violet red")
  (set-face-attribute 'rainbow-delimiters-depth-8-face nil :foreground "SeaGreen2")
  (set-face-attribute 'rainbow-delimiters-depth-9-face nil :foreground "chocolate")
  (set-face-attribute 'rainbow-delimiters-unmatched-face nil :foreground "red"))
```

4.1.6 rainbow-mode

`rainbow-mode` colorizes hex color codes. Useful when customizing Emacs themes. For example: #E74C3C, #F92672, #9C91E4, #268BD2. By the way, use `C-c u c`, `helm-insert-color-hex`, to select color hexes and insert them at point.

```
(use-package rainbow-mode
  :config
  (add-hook 'org-mode-hook 'rainbow-mode)
  (add-hook 'prog-mode-hook 'rainbow-mode))
```

4.2 Auto Completion

4.2.1 company COMPLETION:POPUPS

`company-mode` is my auto-completion mode of choice. It's a pretty big package, and is highly extensible and configurable. There are company backends that people have written for all sorts of things; JavaScript, org-mode, Python, C++, etc. Throughout this config, company backends will be added for corresponding modes.

use-package Company

```
(use-package company
  :init
  (add-hook 'after-init-hook 'global-company-mode)
```

```
  ;; provides a little popup for documentation
  :bind*
  (("C-S-;" . company-files)
   ("C-;" . company-manual-begin)
   :map company-active-map
   ("C-n" . company-select-next)
   ("C-p" . company-select-previous)
   ("M-n" . company-next-page)
   ("M-p" . company-previous-page))
  :config
  (use-package company-quickhelp
    :config
    (setq company-quickhelp-max-lines 10
          company-idle-delay 2.0
          company-quickhelp-delay 0.25)
    (bind-key "M-h" 'company-quickhelp-manual-begin company-mode-map))
  (add-to-list 'company-backends 'company-anaconda)
  (progn
    (setq company-backends (delete 'company-semantic company-backends))
    (use-package helm-company
      :bind ("C-c <tab>" . helm-company)))
  (company-quickhelp-mode 1)
  :diminish 'company-mode)
```

Use company in minibuffer.

```
(with-eval-after-load "company-autoloads"
  (global-company-mode 1)

  (setq company-tooltip-limit 20
        company-minimum-prefix-length 1
        company-echo-delay 0
        company-begin-commands '(self-insert-command
                                 c-electric-lt-gt c-electric-colon
                                 completion-separator-self-insert-command)
        company-idle-delay 0.2
        company-show-numbers t
        company-tooltip-align-annotations t)

  (defvar-local company-col-offset 0 "Horisontal tooltip offset.")
```

```
(defvar-local company-row-offset 0 "Vertical tooltip offset.")

(defun company--posn-col-row (posn)
  (let ((col (car (posn-col-row posn)))
        ;; `posn-col-row' doesn't work well with lines of different height.
        ;; `posn-actual-col-row' doesn't handle multiple-width characters.
        (row (cdr (posn-actual-col-row posn))))
    (when (and header-line-format (version< emacs-version "24.3.93.3"))
      ;; http://debbugs.gnu.org/18384
      (cl-decf row))
    (cons (+ col (window-hscroll) company-col-offset) (+ row company-row-offset))))

(defun company-elisp-minibuffer (command &optional arg &rest ignored)
  "`company-mode' completion back-end for Emacs Lisp in the minibuffer."
  (interactive (list 'interactive))
  (case command
    ('prefix (and (minibufferp)
                  (case company-minibuffer-mode
                    ('execute-extended-command (company-grab-symbol))
                    (t (company-capf `prefix)))))
    ('candidates
     (case company-minibuffer-mode
       ('execute-extended-command (all-completions arg obarray 'commandp))
       (t nil)))))

(defun minibuffer-company ()
  (unless company-mode
    (when (and global-company-mode (or (eq this-command #'execute-extended-command)
                                       (eq this-command #'eval-expression)))

      (setq-local company-minibuffer-mode this-command)

      (setq-local completion-at-point-functions
                  (list (if (fboundp 'elisp-completion-at-point)
                            #'elisp-completion-at-point
                          #'lisp-completion-at-point) t))

      (setq-local company-show-numbers nil)
      (setq-local company-backends '((company-elisp-minibuffer company-capf)))
      (setq-local company-tooltip-limit 8)
```

```
        (setq-local company-col-offset 1)
        (setq-local company-row-offset 1)
        (setq-local company-frontends '(company-pseudo-tooltip-unless-just-one-fronten
                                        company-preview-if-just-one-frontend))

        (company-mode 1)
        (when (eq this-command #'execute-extended-command)
          (company-complete)))))

  (add-hook 'minibuffer-setup-hook #'minibuffer-company)
  ;;(remove-hook 'minibuffer-setup-hook #'minibuffer-company)
  ;;(add-hook 'eval-expression-minibuffer-setup-hook #'minibuffer-company)
  ;; (with-eval-after-load "company-flx-autoloads"
  ;; (company-flx-mode))
)
```

4.2.2 auto-complete COMPLETION:POPUPS

`auto-complete` is another completion package. It's more or less dormant in my configuration, but I load it here because why not.

```
(use-package auto-complete)
```

4.3 guide-key

Display possible key binding completions automatically in a small pop-up buffer with `guide-key`. For example, typing `C-h` will provide a small hint pop-up for commands under this prefix.

```
(use-package guide-key
  :init
  (guide-key-mode 1)
  (setq guide-key/guide-key-sequence '("C-x r" ;; registers and rectangles
                                       "C-x 4" ;; file finding
                                       "C-x v" ;; vc
                                       "C-x 8" ;; special characters
                                       "C-c 5" ;; vimish-fold
                                       "C-h"   ;; help
                                       "C-z"
                                       "C-c e" ;; ??
                                       "C-c C-v" ;; org-babel
```

```
                                        "C-c o")) ;; switch to specific files
  (setq guide-key/recursive-key-sequence-flag t)
  (setq guide-key/popup-window-position 'bottom)
  :diminish 'guide-key-mode)
```

4.4 Dictionary

`dictionary` provides a way to query offline dictionary servers for looking up definitions, synonyms, etc.

```
(use-package dictionary)
```

Use my `helm-dictionary` package. There is another package on `Melpa` called `helm-dictionary`, so have to be careful with this import.

Make sure that helm-dictionary directory is know to Emacs load-path. If this line weren't here, in the following `use-package` call, Emacs would look for `helm-dictionary` in the load-path, not find it, then look to download helm-dictionary from Melpa Emacs repositories (which would be the wrong helm-dictionary).

```
(add-to-list 'load-path (concat lisp-dir "helm-dictionary"))
```

Load `helm-dictionary`

```
(use-package helm-dictionary
  :commands (helm-dictionary-lookup)
  :bind
  ("C-c d" . helm-dictionary-lookup))
```

5 DONE Buffer navigation [0/0]

5.1 General

```
(diminish 'auto-revert-mode)
```

5.1.1 Navigation

```
(defun smart-beginning-of-line ()
    "Move point to first non-whitespace character or beginning-of-line.

  Move point to the first non-whitespace character on this line.
```

```
  If point was already at that position, move point to beginning of line."
    (interactive)
    (let ((oldpos (point)))
      (back-to-indentation)
      (and (= oldpos (point))
           (if (equal major-mode 'org-mode)
               (org-beginning-of-line)
             (beginning-of-line)))))

(global-set-key [home] 'smart-beginning-of-line)
(global-set-key (kbd "C-a") 'smart-beginning-of-line)
(define-key org-mode-map (kbd "C-a") 'smart-beginning-of-line)
```

5.2 Narrowing

`helm` and `ivy` are menu-based selection/narrowing libraries. They give you a way to sift through lists with pop-up fuzzy-searchable menus. `helm` and `ivy` are used as `M-x` interfaces, for example. Binding `M-x` to `helm-M-x` is a much nicer way to select commands than the Emacs default. Try it out.

These libraries are easy to use and extend, so there are many `helm` / `ivy` based packages available on `Melpa` and other Emacs package repositories. `counsel` is the name for the collection of `ivy` interfaces for common Emacs commands, such as helm commands and file finding.

`helm-swoop` is a helm-based package for sifting through buffer contents. `helm-multi-swoop` lets us do it across all open buffers. `swiper` is the `helm-swoop` equivalent for `ivy`.

`ido` is a selection/narrowing package a bit different than `helm` and `ivy`. It uses the `minibuffer` instead of a pop-up list, and I generally consider it best for selecting from a well-known list for which there isn't attached meta-data. Selecting from a list of tags or files in familiar directory, for example. `smex` is an `ido` packaged used for `M-x`.

5.2.1 `helm` SEARCH:FUZZY:POPUPS

This variables must be set before loading `helm-gtags`

```
(setq helm-gtags-prefix-key "\C-cg")
```

Load helm and set some basic defaults.

```
(use-package helm
  :init
```

```
(helm-mode 1)
:bind*
(("M-y" . helm-show-kill-ring)
 ("M-X" . helm-M-x)
 ("C-h SPC" . helm-all-mark-rings)
 ("C-x b" . helm-mini)
 ("C-x C-o" . helm-buffers-list)
 ("C-h SPC" . helm-all-mark-rings)
 ("C-c s" . helm-occur)
 ("C-h F" . helm-insert-command-name)
 :map helm-map
 ("C-c C-y" . helm-yank-selection-and-quit)
 ("C-i" . helm-select-action) ;; This is a big one. Use C-SPC to select entries,
 ;; then C-i (or TAB) to select an action to perform on
 ;; those selected entries.
 :map helm-buffer-map
 ("C-c C-k" . helm-buffer-run-kill-buffers))
:config
(helm-autoresize-mode t)
(setq
 ;; scroll 4 lines other window using M-<next>/M-<prior>
 helm-scroll-amount            4
 ;; search for library in `require' and
 ;; declare-function' sexp.
 helm-ff-search-library-in-sexp t
 ;; open helm buffer inside current window, not
 ;; occupy whole other window
 helm-split-window-in-side-p    t
 ;; limit the number of displayed canidates
 helm-candidate-number-limit 500
 ;; move to end or beginning of source when
 helm-ff-file-name-history-use-recentf nil
 ;; reaching top or bottom of source.
 helm-move-to-line-cycle-in-source     t
 ;; fuzzy matching buffer names when non-nil
 helm-buffers-fuzzy-matching           nil
 ;;  Helm size. Don't want it to be too distracting.
 helm-autoresize-max-height       25
 helm-autoresize-min-height       3)
(set-face-attribute 'helm-candidate-number nil
```

```
                    :background "salmon2"
                    :foreground "black")
  :diminish 'helm-mode)
```

Helm-dash is a great package that allows us to look up documentation with helm. It is mostly equivalent to Dash, but does not depend on it. Use 'helm-dash-install-docset' to download a docset fo a particular language (or language package).

```
(use-package helm-dash
 :bind*
 (("C-c C-?" . helm-dash-at-point)))
```

Helm-flx improves helms scoring of results. Helm-fuzzier improves its fuzzy matching. Seems useless.

```
(use-package helm-flx
  :init ;(helm-flx-mode 1)
  :config
  ; (setq helm-flx-for-helm-find-files t ;; t by default
  ;       helm-flx-for-helm-locate t)
  )

(use-package helm-fuzzier
  :init ;(helm-fuzzier-mode 1)
  )
```

`helm-swoop` is an interface for searching for lines in a buffer using helm.

```
(use-package helm-swoop
  :init (progn
          (global-set-key (kbd "C-c s") 'helm-swoop)
          (global-set-key (kbd "C-c S") 'helm-multi-swoop-all))
  :config (progn
            ; When doing isearch, hand the word over to helm-swoop
            (define-key isearch-mode-map (kbd "M-i") 'helm-swoop-from-isearch)
            ; From helm-swoop to helm-multi-swoop-all
            (define-key helm-swoop-map (kbd "M-i") 'helm-multi-swoop-all-from-helm-swo
            ; Save buffer when helm-multi-swoop-edit complete
            (setq helm-multi-swoop-edit-save t))
            ; If this value is t, split window inside the current window
```

```
(setq helm-swoop-split-with-multiple-windows t)
;; Split direcion. 'split-window-vertically or 'split-window-horizontally
(setq helm-swoop-split-direction 'split-window-vertically)
;; If nil, you can slightly boost invoke speed in exchange for text color
(setq helm-swoop-speed-or-color t)
;; Hack to make helm stop pre-inputting search
(setq helm-swoop-pre-input-function (lambda () nil)))
```

Can use `TAB` and `C-i` to perform the same action as `RETURN`, but without killing the `helm` process. This is very useful, for example, when you want to sift through Emacs Lisp documentation in a `C-h f` (`describe-function`) call.

```
(define-key helm-map (kbd "C-i") 'helm-execute-persistent-action)  ; make TAB works in
```

Some helm bindings and additional commands.

```
(define-key helm-map (kbd "C-z")  'helm-select-action)             ; list actions usin

(define-key helm-grep-mode-map (kbd "<return>")  'helm-grep-mode-jump-other-window)
(define-key helm-grep-mode-map (kbd "n")  'helm-grep-mode-jump-other-window-forward)
(define-key helm-grep-mode-map (kbd "p")  'helm-grep-mode-jump-other-window-backward)

(when (executable-find "curl")
  (setq helm-google-suggest-use-curl-p t))

(add-to-list 'helm-sources-using-default-as-input 'helm-source-man-pages)

(global-set-key (kbd "C-c 7 w") 'helm-wikipedia-suggest)
(global-set-key (kbd "C-c 7 g") 'helm-google-suggest)
(global-set-key (kbd "C-c 7 s") 'helm-surfraw)

(global-set-key (kbd "C-x r j") 'jump-to-register)

(define-key 'help-command (kbd "C-f") 'helm-apropos)
(define-key 'help-command (kbd "r") 'helm-info-emacs)
(define-key 'help-command (kbd "C-l") 'helm-locate-library)

;;; Save current position to mark ring
(add-hook 'helm-goto-line-before-hook 'helm-save-current-pos-to-mark-ring)
```

Show minibuffer history with Helm

```
(define-key minibuffer-local-map (kbd "M-p") 'helm-minibuffer-history)
```

Navigating file

```
(define-key global-map [remap find-tag] 'helm-etags-select)
(define-key global-map [remap list-buffers] 'helm-buffers-list)
```

Use Helm to list eshell history:

```
(add-hook 'eshell-mode-hook
          (lambda ()
              (local-set-key (kbd "C-c C-l") 'helm-eshell-history)))
```

Fuzzy matching for elisp helm completion. E.g., `helm-M-x` "fi ile" will have "find-file" as one of the possible completions.

```
(setq helm-lisp-fuzzy-completion t)
```

1. `helm-buffers-list` SEARCH:FUZZY:POPUPS

   Make helm-buffers-list sort the buffers.

```
(defun dwc-helm-source-buffers (buffers)
  "Return sorted source-buffers.  Helm will not sort results by default."
  (let ((last-used (subseq buffers 0 (min 5 (length buffers))))
        (buffers (subseq buffers (min 6 (length buffers))))
        dired-buffers
        other-buffers
        (buf-sort (lambda (bufs)
                    (cl-sort bufs
                             (lambda (a b)
                               (or (< (length a) (length b))
                                   (and (= (length a) (length b))
                                        (string-lessp a b))))))))
    (dolist (buf buffers)
      (if (with-current-buffer buf
            (eq major-mode 'dired-mode))
          (push buf dired-buffers)
        (push buf other-buffers)))
    (append
```

```
      (funcall buf-sort last-used)
      (funcall buf-sort other-buffers)
      (funcall buf-sort dired-buffers))))

(defun helm-buffers-sort-dired-buffers (orig-fun &rest args)
  (dwc-helm-source-buffers (apply orig-fun args)))

(advice-add 'helm-buffers-sort-transformer :around 'helm-buffers-sort-dired-buffe
```

2. Hide mode-lines under helm

   Don't display the modeline in bottom buffers when helm is active. It's distracting, useless, and unsightly. The following code was grabbed from StackExchange.

   (a) **Collect bottom buffers**

```
(defvar bottom-buffers nil
  "List of bottom buffers before helm session.
    Its element is a pair of `buffer-name' and `mode-line-format'.")

(defun bottom-buffers-init ()
  (when bottom-buffers
    (bottom-buffers-show-mode-line))
  (setq bottom-buffers
        (cl-loop for w in (window-list)
                 when (window-at-side-p w 'bottom)
                 collect (with-current-buffer (window-buffer w)
                           (cons (buffer-name) mode-line-format)))))

(add-hook 'helm-before-initialize-hook #'bottom-buffers-init)
```

   (a) **Hide mode line**

```
(defun bottom-buffers-hide-mode-line ()
  (mapc (lambda (elt)
          (with-current-buffer (car elt)
            (setq-local mode-line-format nil)))
        bottom-buffers))

(add-hook 'helm-after-initialize-hook #'bottom-buffers-hide-mode-line)
```

(a) **Restore mode line**

```
(defun bottom-buffers-show-mode-line ()
  (when bottom-buffers
    (mapc (lambda (elt)
            (with-current-buffer (car elt)
              (setq-local mode-line-format (cdr elt))))
          bottom-buffers)
    (setq bottom-buffers nil)))

(add-hook 'helm-exit-minibuffer-hook #'bottom-buffers-show-mode-line)

(defun helm-keyboard-quit-advice (orig-func &rest args)
  (bottom-buffers-show-mode-line)
  (apply orig-func args))

(advice-add 'helm-keyboard-quit :around #'helm-keyboard-quit-advice)
```

3. Change helm mode-line appearance

```
(defun helm-mode-line-hook ()
  (let ((color "#8b475d"))
    (face-remap-add-relative
     'mode-line `((:foreground ,color :background ,color) mode-line))
    (face-remap-add-relative
     'helm-candidate-number `((:foreground "black" :background ,color) mode-line))))

(add-hook 'helm-major-mode-hook 'helm-mode-line-hook)
```

4. `helm-dash`

   `helm-dash` is a wonderful utility for looking up docs using the `dash` utility. Install dash docs with `helm-dash-install-docset` (Docs are search-able by package name, so you can download the `Pandas` docs for Python or the `node.js` docs for JavaScript).

```
(use-package helm-dash
:config
; don't open the docs in chrome/firefox/whatever
(setq helm-dash-browser-func 'eww))
```

5.2.2 ido SEARCH:FUZZY

I prefer Ido for find-file. It's less distracting, and I usually down need to browse files.

```
(ido-mode t)
(bind-key* "C-x C-f" 'ido-find-file)
(bind-key "C-x S-k" 'ido-kill-buffer)
```

Ido really insists on using it's own kill-buffer by default...

```
(defun kill-buffer-wrapper () (interactive) (kill-buffer))
(unbind-key "C-x k")
(bind-key "C-x k" 'kill-buffer-wrapper)
```

Ido should save its history files where everything else does!

```
(setq-default ido-save-directory-list-file (concat saveplace-dir "ido.last"))
```

1. smex

   Use smart-M-x for `M-x`. It's discreet. I use `C-h f` (with `C-i` to preview documentation) to browse Emacs functions/commands, and `M-x` to actually execute them.

   ```
   (use-package smex
     :init (global-set-key (kbd "M-x") 'smex)
     :config
     (smex-initialize)
     :bind
     ("C-z" . magit-status))
   ```

5.2.3 ivy SEARCH:FUZZY:POPUPS

Ivy is a nice package, but I'm partial to helm. Ivy/counsel are still useful sometimes, though.

```
(use-package ivy
  :init
  (use-package counsel
    :config
    :bind*
    ("C-x y" . counsel-org-tag))
```

```
:config
(set-face-attribute 'ivy-current-match nil :background "#ffb6c1" :foreground "black"
(setq ivy-height 5)
(setq ivy-format-function 'ivy-format-function-arrow))
```

1. `swiper` SEARCH:FUZZY:POPUPS

   Sift through a buffer's contents with `swiper`. I think it's faster than `helm-swoop` (but less feature-rich), so it has that going for it.

   ```
   (use-package swiper)
   ```

5.3 Jumping

Move cursor immediately to a given location. `avy` and `ace` are different approaches to the same idea. They overlay letters on top of all character instances to which you'd like to jump. `jump-char` allows us to jump to nearest character, forward or backward; same idea, but goes to the nearest character instead of requiring that you specify the character. `ace-isearch` wraps `isearch` such that for searches > 6 characters in length, isearch automatically switches to `swiper` otherwise.

5.3.1 avy JUMP

`avy-goto-char-timer`, bound to `C-m`, is the main takeaway from `avy`, in my opinion. Use it to quickly jump between static buffers.

```
(use-package avy
  :init
  :config
  (setq avy-timeout-seconds .2)
  :bind
  (
   ("C-l"   . avy-goto-line)
   ("C-S-w"   . avy-kill-region)
   ("C-S-l" . avy-copy-line)
   ("<C-m>" . avy-goto-char-timer)
   ("C-."   . avy-goto-char)
   ("C-s"   . avy-goto-word-1)))

;;               a   s   d   f   h   j   l
(setq avy-keys '(97  115 100 102 106 108 104 113 119
```

```
                101 114 116 121 117 111 122 120 118
                98  109 44  46))

(setq avy-dispatch-alist '((?c . avy-action-copy)
                           (?k . avy-action-kill-move)
                           (?K . avy-action-kill-stay)
                           (?m . avy-action-mark)
                           (?; . avy-action-execute-code)
                           (?n . avy-narrow-region)
                           (?p . avy-action-copy-and-yank)))
```

Bind `C-x C-x` to `avy-pop-mark`

```
(global-set-key (kbd "C-x C-x") 'avy-pop-mark)

(defun avy-action-copy-and-yank (pt)
  "Copy and yank sexp starting on PT."
  (avy-action-copy pt)
  (yank))

(defun avy-action-execute-code (pt)
  (run-python)
  (python-shell-send-region
   (save-excursion (beginning-of-line) (point))
   (save-excursion (avy-action-goto pt)
                   (end-of-line)
                   (point))))

(defun avy-narrow-region (pt)
    (narrow-to-region
     (save-excursion (beginning-of-line) (point))
     (save-excursion (avy-action-goto pt)
                     (end-of-line)
                     (point))))
```

5.3.2 ace JUMP

Jump quickly to any word using just two key strokes with ace-jump-mode:

```
(use-package ace-jump-mode
  :config
```

```
(use-package ace-jump-zap
:commands ace-jump-zap-to-char
          ace-jump-zap-up-to-char)
:bind*
(("M-z" . ace-jump-zap-to-char)
 ("M-S-z" . ace-jump-zap-up-to-char)))
```

5.3.3 jump-char JUMP

```
(use-package iy-go-to-char
  :commands jump-char-forward jump-char-backward
  :bind
  (("C-r" . iy-go-to-char)
   ("C-M-r" . iy-go-to-char-backward)))
```

5.4 Searching

isearching shouldn't be that common. Usually should be jumping around within a page or occuring or something.

```
(global-set-key (kbd "C-q") 'isearch-forward)
(global-set-key (kbd "C-S-q") 'isearch-backward)
```

Isearch repeat...

```
(define-key isearch-mode-map (kbd "C-q") 'isearch-repeat-forward)
(define-key isearch-mode-map (kbd "C-S-q") 'isearch-repeat-backward)
```

5.4.1 ace-isearch SEARCH:JUMP:FUZZY

This is a mix of `ace-isearch`, `helm-swoop`, and `avy`. Pretty cool. If only one key is searched, it will use ace, if more than one and less than 6 are serached, it will use good ol' isearch, if more than 6 are searched, it will use helm-swoop. Nice idea.

```
(use-package ace-isearch
  :init
  (global-ace-isearch-mode)
  :config
  (setq ace-isearch-use-jump nil)
  (setq ace-isearch-jump-delay 2)
  :diminish 'ace-isearch-mode)
```

Make iterative searching default to regexp searching, which I find much better for building keyboard macros.

```
;(setq search-default-mode t)
```

Use `swiper` instead of `helm-swoop`.

```
(defun ace-isearch-swiper-from-isearch ()
  "Invoke ‘helm-swoop’ from ace-isearch."
  (interactive)
  (let (($query (if isearch-regexp
                    isearch-string
                  (regexp-quote isearch-string))))
    (let (search-nonincremental-instead)
      (ignore-errors (isearch-exit)))
    (swiper $query)))

(setq ace-isearch-function-from-isearch 'ace-isearch-swiper-from-isearch)
```

5.4.2 isearcher

```
(defvar isearcher--register nil)
(defvar isearcher--end-register nil)

(defun isearcher ()
  (interactive)
  (when (and isearcher--register isearcher--end-register)
    (if (not (equal (point) (marker-position isearcher--register)))
        (goto-char isearcher--register)
      (goto-char isearcher--end-register))))

(defun isearcher-exchange-point-and-mark ()
  (interactive)
  (if (use-region-p)
      (call-interactively 'exchange-point-and-mark)
    (isearcher)))

(defun isearcher--set-isearcher-register ()
  (setq-local isearcher--register (point-marker)))

(defun isearcher--set-end-isearcher-register ()
```

```
  (setq-local isearcher--end-register (point-marker)))

(add-hook 'isearch-mode-hook 'isearcher--set-isearcher-register)
(add-hook 'isearch-mode-end-hook 'isearcher--set-end-isearcher-register)

;(global-set-key (kbd "C-x C-x") 'isearcher-exchange-point-and-mark)
```

5.4.3 occur

```
(bind-key* "M-s o" 'occur)
```

5.5 Grep

5.5.1 rgrep SEARCH:REGEXP

Load and configure the rgrep code in goodies. I believe it's from John Wiegley.

```
(require 'rgrep)

(eval-after-load "grep"
  '(defadvice grep-mode (after grep-register-match-positions activate)
     (add-hook 'compilation-filter-hook 'grep-register-match-positions nil t)))

(eval-after-load "multiple-cursors"
  '(add-to-list 'mc--default-cmds-to-run-once 'mc/add-cursors-to-all-matches))

(eval-after-load "wgrep"
  '(define-key wgrep-mode-map (kbd "C-c C-æ") 'mc/add-cursors-to-all-matches))

(eval-after-load "grep"
  '(progn
     ;; Don't recurse into some directories
     (add-to-list 'grep-find-ignored-directories "target")
     (add-to-list 'grep-find-ignored-directories "node_modules")
     (add-to-list 'grep-find-ignored-directories "vendor")

     ;; Add custom keybindings
     (define-key grep-mode-map "q" 'rgrep-quit-window)
     (define-key grep-mode-map (kbd "C-<return>") 'rgrep-goto-file-and-close-rgrep)
     (define-key grep-mode-map (kbd "C-x C-s") 'wgrep-save-all-buffers)
```

```
    ;; Use same keybinding as occur
    (setq wgrep-enable-key "e")))
```

5.6 Re-centering

```
(global-set-key (kbd "M-L") 'reposition-window)
(global-set-key (kbd "C-S-l") 'recenter-top-bottom)
```

Use up and down arrows to step through occur. Means I can sit back and rifle through code.

```
(defun occur-step (arg)
  (interactive "P")
  (let* ((num 0)
         (arg (or arg 1))
         (foo (if (or (not arg) (>= arg 0))
                  'occur-next
                'occur-prev)))
    (while (< num (abs arg))
      (setq num (1+ num))
      (funcall foo)))
  (recenter))

(defun occur-step-forward (arg)
  (interactive "P")
  (let ((arg (if arg (abs arg) 1)))
    (occur-step arg))
  (occur-mode-display-occurrence))

(defun occur-step-backward (arg)
  (interactive "P")
  (let ((arg (if arg (- (abs arg)) -1)))
    (occur-step arg))
  (occur-mode-display-occurrence))

(define-key occur-mode-map (kbd "<down>") 'occur-step-forward)
(define-key occur-mode-map (kbd "<up>") 'occur-step-backward)
```

Better display occurrence

```
(add-hook 'occur-mode-find-occurrence-hook 'recenter)
```

6 DONE Windowing [0/0]

```
(let ((inhibit-message nil))
  (message "Configuring window/buffer/frame management..."))
```

6.1 Basic Settings

Add a more convenient `other-frame` binding

```
(bind-keys*
 ("M-o" . other-frame))

(global-set-key (kbd "C-x C-b") nil)
```

Remaps the other-window command. In general, commands with consecutive

```
(global-set-key (kbd "C-x o") 'other-window)
(global-set-key (kbd "C-x O") 'other-frame)
(global-set-key (kbd "C-c b") 'switch-to-other-buffer)
(global-set-key (kbd "C-x w t") 'transpose-windows)
```

Quickly move the cursor to the first instance of a character with iy-go-to-char:

```
(use-package iy-go-to-char)

(defun safe-file-visit-hook ()
  "If a file is over a given size, make the buffer read only."
  (when (> (buffer-size) (* 1024 1024))
    (print
     "Buffer set to read-only mode due to its size. See ‘safe-file-visit-hook’.")
    (setq buffer-read-only t)
    (buffer-disable-undo)
    (fundamental-mode)))

(add-hook 'find-file-hook 'safe-file-visit-hook)
```

General navigation bindings:

```
(global-unset-key (kbd "C-x 5 0"))
(global-set-key (kbd "C-x 5 DEL") 'delete-frame)

(global-unset-key (kbd "C-x 0"))
(global-set-key (kbd "C-x DEL") 'delete-window)
```

Key bindings for other files:

```
(global-set-key (kbd "C-c o f e")
                (lambda ()
                  (interactive)
                  (dwc-find-file-other-frame "~/.emacs.d/init.el")
                  (split-window-horizontally)
                  (windmove-right)
                  (find-file "~/.emacs.d/custom/")))
(global-set-key (kbd "C-c o f C-e")
                (lambda ()
                  (interactive)
                  (find-file-other-window "~/.emacs.d/custom/")))

(define-key org-mode-map (kbd "C-c o i") nil)
(global-set-key (kbd "C-c o i")
                (lambda () (interactive) (find-file "~/.emacs.d/init.el")))
(global-set-key (kbd "C-c o C-i")
                (lambda ()
                  (interactive)
                  (find-file-other-window "~/.emacs.d/init.el")))

(global-set-key (kbd "C-c o c")
                (lambda () (interactive) (find-file "~/.emacs.d/config.org")))
(global-set-key (kbd "C-c o C-c")
                (lambda ()
                  (interactive)
                  (find-file-other-window "~/.emacs.d/config.org")))

(global-set-key (kbd "C-c o m")
                (lambda () (interactive) (switch-to-buffer "*Messages*")))
(global-set-key (kbd "C-c o s")
                (lambda () (interactive) (switch-to-buffer "*Scratch*")))
```

6.2 General Appearance

6.2.1 Fringe APPEARANCE:GENERAL:WINDOW

The fringe is the area on the left and right edge of Emacs windows.

```
(setq fringe-mode '(4 . 4)) ;; set default fringe width to be 4 pixels on both sides
```

6.2.2 Cursor CURSOR:IDLE

Change cursor when idle or in a different buffer. I like the vertical bar cursor, so this is nice when I need to find my cursor after not looking at the screen for minute.

```
(use-package cursor-chg
  :init
  (curchg-toggle-cursor-type-when-idle)
  (setq curch-idle-interval 0.5))
```

6.3 Finding files

`get-personal-file-binding` is used to binding a key to a file. Results in a function, named appropriately, that is called to find `file` when `key` is pressed. The advantage of this is that `guide-key` will display the binding nicely.

```
(defun make-get-personal-file-key (file key)
  "Doesn't handle duplicate filenames very well.
  Does handle files and directorys with same basename, though"
  (let ((function-symbol (make-symbol
                          (concat "get-personal-"
                                  (if (file-directory-p file)
                                      (concat "dir:" (file-name-nondirectory file))
                                      (concat"file:" (file-name-base file)))
                                  ))))
    `(progn
       (defun ,function-symbol ()
         (interactive)
         (find-file ,file))
       (global-set-key (kbd ,key) ',function-symbol))))

(defmacro get-personal-file-binding (file key)
```

```
  (make-get-personal-file-key (file-truename (directory-file-name
                                              (if (stringp file)
                                                  file
                                                (symbol-value file))))
                              key))
```

6.3.1 dired

```
(use-package dired-details
  :config
  (setq-default dired-details-hidden-string "--- ")
  (setq dired-dwim-target t)
  (dired-details-install)
  :bind
  ("C-c C-p" . dired-up-directory))
(use-package dired-subtree
  :config
  (define-key dired-mode-map "i" 'dired-subtree-insert)
  (define-key dired-mode-map ";" 'dired-subtree-remove))
```

6.4 Selection

6.4.1 ace-window JUMP

Jump quickly between windows and frames using just two key strokes with ace-window. Essential package:

```
(defun my/other-window ()
  (other-window 1))

(use-package ace-window
  :commands ace-window
  :bind*
  (("C-o" . ace-window))
  :config
  (setq aw-scope 'frame
        aw-background t
        aw-keys '(?j ?k ?l ?\; ?s ?d ?f ?g)
        aw-dispatch-alist '((?x aw-delete-window " Ace - Delete Window")
                            (?m aw-swap-window " Ace - Swap Window")
                            (?n aw-flip-window)
```

```
                                (?v aw-split-window-vert " Ace - Split Vert Window")
                                (?b aw-split-window-horz " Ace - Split Horz Window")
                                (?i delete-other-windows " Ace - Maximize Window")
                                (?o my/other-window " Ace - Other window")))
  :diminish 'ace-window)
```

6.4.2 windmove DIRECTION

Navigate windows directionally with wind-move:

```
(use-package windmove
  :commands
  ;; Here because alternative commands (key chords) do not trigger package autoload.
  (windmove-left windmove-right windmove-up windmove-down)
  :init
  (bind-keys
   ("C-x w j" . windmove-left)
   ("C-x w l" . windmove-right)
   ("C-x w i" . windmove-up)
   ("C-x w k" . windmove-down)))
```

6.4.3 ace-popup-menu SELECT

```
(use-package ace-popup-menu
  :diminish 'ace-popup-menu)
```

6.5 Saving/Restoring/Killing

6.5.1 workspaces

1. perspeen

   ```
   (use-package perspeen
     :ensure t
     :init
     (setq perspeen-use-tab t)
     :config
     (perspeen-mode))
   ```

   Redefine `perspeen-create-ws` to create a new workspace with only one window.

```
(defun perspeen-create-ws ()
  "Create a new workspace."
  (interactive)
  (perspeen-new-ws-internal)
  (delete-other-windows)
  (perspeen-update-mode-string))
```

6.5.2 desktop-save SAVE

Currently turned off

```
(setq desktop-basefilename "emacs.desktop"
      desktop-path `(,saveplace-dir))
```

6.5.3 volatile-kill-buffers

```
(defun volatile-kill-buffers ()
   "Kill current buffer unconditionally."
   (interactive)
   (let ((buffer-modified-p nil))
     (kill-buffer (current-buffer))))

(global-set-key (kbd "C-x M-K") 'volatile-kill-buffers)
```

6.6 Behavior

Force horizontal window splitting. No clue why anyone would want to split code windows vertically, you just waste window space past column 80.

```
(setq split-width-threshold 80)
(setq split-height-threshold nil)
```

6.6.1 zygospore

```
(use-package zygospore
  :bind ("C-x 1" . zygospore-toggle-delete-other-windows))
```

6.6.2 shackle POPUPS

Great package. It allows you to configure how popup messages are handled. For instance, please stop creating the magit status buffer in another window.

```
(use-package shackle
  :config
  (shackle-mode 1)
  (setq
   shackle-default-rule '(:inhibit-window-quit t :other t :align right)
   shackle-rules
   `(;; Util
     ("^\\*.+-Profiler-Report .+\\*$"
      :align below :size 0.3 :regexp t)
     ("*Ido Completions*"
      :align right :size 0.3)
     ("*esup*"
      :align below :size 0.4 :noselect t :inhibit-window-quit nil)
     ("*minor-modes*"
      :align below :size 0.5 :noselect t :inhibit-window-quit nil)
     ("*eval*"
      :align below :size 16  :noselect t :inhibit-window-quit nil)
     (dired-mode :ignore t)
     (helm-mode :ignore t)
     ("*Deletions*" :inhibit-window-quit nil :same t)
     ("\\`\\*helm.*?\\*\\'" :regexp t :align  :size 0.4)
     ;; Emacs
     ("*Pp Eval Output*"
      :align below :size 0.3 :inhibit-window-quit nil)
     ("*Apropos*"
      :align below :size 0.3 :inhibit-window-quit t)
     ("*Backtrace*"
      :align below :size 25  :noselect t :inhibit-window-quit nil)
     ("*Help*"
      :align right :size 80 :select t)
     ("\\*[hH]elp\[R\]*\\*" :regexp t :align right :size 80 :select t)
     ("\\*help\[R\]*\\*" :regexp t :align right :size 80 :select t)
     ("\\**Python Doc**\\*" :regexp t :align right :size 80 :select t)
     ("\\**magit*"
      :regexp t :same t :select t :inhibit-window-quit nil)
     ("*Messages*"
      :align below :size 15  :select t :inhibit-window-quit nil)
     ("*Warnings*"
      :align below :size 10  :noselect t :inhibit-window-quit nil)
     (compilation-mode
```

```
     :align below :size 15  :noselect t :inhibit-window-quit nil)
    (eww-mode
     :align below :size 30  :select t :inhibit-window-quit nil)
    ("*command-log*"
     :align right :size 28  :noselect t :inhibit-window-quit nil)
    ;; vcs
    ("*vc-diff*"
     :align below :size 15  :noselect t :inhibit-window-quit nil)
    ("*vc-change-log*"
     :align below :size 15  :select t :inhibit-window-quit nil)
    (vc-annotate-mode    :same t :inhibit-window-quit nil)
    ("\\*Org Agenda\\*" :select t :inhibit-window-quit t))))
```

Emacs 25.1+ properly shows the completion window at the bottom of the current frame.

```
(unless (version< emacs-version "25.1")
  (push '("*Completions*"     :align below :size 30  :noselect t)
        shackle-rules))
```

7 DONE Editing [0/0]

```
(let ((inhibit-message nil))
  (message "Configuring editing settings..."))
```

Use my goodies repository for some stuff I like to use, for instance, `helm-insert-color-name`, for selecting colors and inserting their name at point.

```
(add-to-list 'load-path (concat lisp-dir "goodies"))
(unless (require 'dwc-goodies nil t)
  (warn "Could not load dwc-goodies"))
```

7.1 Basic Settings

```
(global-set-key (kbd "C-M-z") 'zap-to-char)
```

```
(setq global-mark-ring-max 5000      ; increase mark ring to contains 5000 entries
      mark-ring-max 10000            ; increase kill ring to contains 10000 entries
      mode-require-final-newline t   ; add a newline to end of file
      tab-width 4                    ; default to 4 visible spaces to display a tab
```

```
      kill-ring-max 10000            ; increase kill-ring capacity
      kill-whole-line t)             ; if NIL, kill whole line and move the next line
```

`electric-indent-mode` provides on-the-fly re-indentation

```
(setq electric-indent-mode nil)
```

DISABLED *what does this do?*

```
;  (put 'downcase-region 'disabled nil)
;  (put 'upcase-region 'disabled nil)
```

Delete tabs with backspace

```
(setq backward-delete-char-untabify-method 'hungry)
```

Use space to indent by default

```
(setq-default indent-tabs-mode nil)
```

Set appearance of a tab that is represented by 4 spaces

```
(setq-default tab-width 4)
```

Remap `C-z` to `just-one-space`.

```
(global-set-key (kbd "C-z") 'just-one-space)
```

Remap `M-z` to

```
(defun delete-space ()
  "Kill the whitespace between two non-whitespace characters"
  (interactive "*")
  (if (and (region-active-p) (interactive-p))
      (let* ((beg (region-beginning))
             (end (region-end))
             (num-lines (count-lines beg end)))
        (save-excursion
          (goto-char beg)
          (cl-loop for i from 1 to num-lines
                   do (progn (beginning-of-line)
                             (delete-space)
```

```
                                (forward-line)))))
    (save-excursion
    (save-restriction
      (save-match-data
        (progn
          (re-search-backward "[^ \t\r]" nil t)
          (re-search-forward "[ \t\r]+" nil t)
          (replace-match "" nil nil)))))))
(global-set-key (kbd "C-M-S-z") 'delete-space)
```

Remap `M-k` to `kill-whole-line`

```
(global-set-key (kbd "M-k") 'kill-whole-line)
```

Use `subword-mode`, which treats camelcase components as word, so, for example, when backwards deleting a word, it will stop at the nearest capital letter.

```
(global-subword-mode 1)
```

7.1.1 Parentheses

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

1. `smartparens`

```
(use-package smartparens
  :bind*
  (:map smartparens-mode-map
        ("M-[" . sp-unwrap-sexp)
        ("C-M-d" . sp-delete-sexp)
        ("C-M-k" . sp-kill-sexp))
  :config
  (unbind-key "M-s")
  ;; These are loaded from a file because currently smartparens isn't
  ;; handling escaped quotations in org well. Funnily enough, smartparens
  ;; binds requires that I bind to a quotation, which requires an escaped
  ;; quotation :O
  (load-file (concat lisp-dir "sp-binds.el"))
  (defun sp-lisp-pair-pred (_ _ _)
    (interactive)
```

```
    (not (or (eq major-mode 'emacs-lisp-mode) (eq major-mode 'org-mode))))
  (sp-pair "'" nil :when '(sp-lisp-pair-pred) :actions '(insert wrap))
  (setq sp-base-key-bindings 'paredit)
  (setq sp-autoskip-closing-pair 'always)
  (setq sp-hybrid-kill-entire-symbol nil)
  (setq sp-backward-delete-char 'paredit-backward-delete)
  (global-set-key (kbd "C-M-w") 'sp-copy-sexp)
  (sp-use-paredit-bindings)
  (show-smartparens-global-mode +1)
  (smartparens-global-mode 1)
  (add-hook 'prog-mode-hook 'turn-on-smartparens-mode)
  (add-hook 'markdown-mode-hook 'turn-on-smartparens-strict-mode)
  :diminish 'smartparens-mode)
```

Precious binding with useless command.

```
(unbind-key "M-r" smartparens-mode-map)

(define-key smartparens-mode-map (kbd "M-[") 'sp-unwrap-sexp)
```

Delete sexp:

```
(defun sp-delete-sexp ()
  "Deletes sexp at point. Does not save to kill ring."
  (interactive)
  (sp-kill-sexp)
  (pop kill-ring))
```

Backward delete sexp:

```
(defun sp-backward-delete-sexp ()
  "Deletes sexp at point. Does not save to kill ring."
  (interactive)
  (sp-backward-kill-sexp)
  (pop kill-ring))
```

7.1.2 Useful Packages

1. `volatile-highlights`

```
(use-package volatile-highlights
  :config
  (volatile-highlights-mode t)
  :diminish 'volatile-highlights-mode)
```

2. clean-auto-indent-mode

```
(use-package clean-aindent-mode
  :commands clean-aindent-mode
  :init
  (add-hook 'prog-mode-hook 'clean-aindent-mode))
```

3. dtrt-indent

```
(use-package dtrt-indent
  :config
  (setq dtrt-indent-verbosity 0)
  (dtrt-indent-mode 1))
```

4. ws-butler

```
(use-package ws-butler
  :commands ws-butler
  :init
  (add-hook 'c-mode-common-hook 'ws-butler-mode)
  (add-hook 'text-mode 'ws-butler-mode)
  (add-hook 'fundamental-mode 'ws-butler-mode)
  (add-hook 'prog-mode-hook 'ws-butler-mode)
  :diminish 'ws-butler-mode)
```

5. anzu

```
(use-package anzu
  :commands
  (anzu-query-replace
   anzu-query-replace-regexp)
  :init
  ;; Bindings
  (bind-key "M-%" 'anzu-query-replace)
  (bind-key "C-M-%" 'anzu-query-replace-regexp)
  :config
```

```
  (global-anzu-mode)
  )
```

6. `iedit`

   When `iedit` mode is turned on, all the occurrences of the current region in the buffer (possibly narrowed) or a region are highlighted. If one occurrence is modified, the change are applied to all other occurrences simultaneously.

```
(use-package iedit
  :commands iedit-mode
  :init
  (bind-key "C-x ;" 'iedit-mode)
  :config
  (setq iedit-toggle-key-default nil))
```

7. `expand-region`

```
(use-package expand-region
  :commands er/expand-region
  :init
  )
```

8. `duplicate-thing`

```
(use-package duplicate-thing
  :commands duplicate-thing
  :init
  (bind-key "M-c" 'duplicate-thing))
```

7.1.3 Key commands

1. General

   Automatically indent when pressing `return`.

```
(global-set-key (kbd "RET") 'newline-and-indent)
```

   Activate whitespace-mode to view all whitespace characters

```
(global-set-key (kbd "C-c w") 'whitespace-mode)
```

Delete region command is useful sometimes where <delete> doesnt work

```
(global-set-key (kbd "C-c <delete>") 'delete-region)
```

Great, simple package. Makes `C-w` and `M-w` act as `kill-whole-line` or copy whole line when no region is active, and normal `kill-region` or `yank` otherwise.

```
(use-package whole-line-or-region)
```

2. Narrowing

```
(global-set-key (kbd "C-x n d") 'narrow-to-defun)
(global-set-key (kbd "C-x n r") 'narrow-to-region)
(global-set-key (kbd "C-x n w") 'widen)
```

7.2 Common

capitalize do-what-I-mean.

```
(global-set-key (kbd "C-x L") 'capitalize-dwim)
```

7.3 Diffing

Show whitespace in diff-mode

```
(add-hook 'diff-mode-hook (lambda ()
                            (setq-local whitespace-style
                                        '(face tabs tab-mark
                                          spaces space-mark trailing
                                          indentation::space
                                          indentation::tab
                                          newline newline-mark))
                            (whitespace-mode 1)))
```

7.4 Undoing/Redoing

7.4.1 `undo-tree`

```
(use-package undo-tree
  :config
  (global-undo-tree-mode)
  :diminish 'undo-tree-mode)
```

7.5 Snippet expansion

7.5.1 yasnippet

yasnippet gives us snippet expansions. You can define your own, and they live in the ~/.emacs.d/snippets/ directory. Snippets are performed with the C-c k binding. So the string src might expand to a generic our source block markup, offer you some completion opportunities that you can fill out and tab through, then leave you between the begin_src_

```
(use-package yasnippet
  :commands
  (yas-exit-all-snippets
   yas/goto-end-of-active-field    ;; Defined below
   yas/goto-start-of-active-field  ;; Defined below
   yas-expand)
  :init
  (yas-global-mode 1)
  ;; Bindings
  (bind-key "<return>" 'yas-exit-all-snippets yas-keymap)
  (bind-key "C-e" 'yas-goto-end-of-active-field yas-keymap)
  (bind-key "C-a" 'yas-goto-start-of-active-field yas-keymap)
                                        ;       (bind-key [(tab)] 'nil yas-minor-mode-
                                        ;       (bind-key (kbd "<tab>") 'yas-expand ya
  :functions (yas/goto-end-of-active-field yas/goto-start-of-active-field)
  :config
  (progn
    (setq yas-verbosity 1) ;; No need to be so verbose
                                        ;         (setq yas-wrap-around-region nil) ;;
    (setq yas-prompt-functions '(yas/ido-prompt yas/completing-prompt))
    (defun my/yas-term-hook ()
      (setq yas-dont-activate t))
    (add-hook 'term-mode-hook 'my/yas-term-hook)
    (defun my/yas-before-hook ()
      (when (eq yas-minor-mode t) (expand-abbrev)))
    (add-hook 'yas-before-expand-snippet-hook 'my/yas-before-hook)
    (defun my/yas-after-hook ()
      (setq snippet-mode-abbrev-table local-abbrev-table))
    (add-hook 'yas-after-exit-snippet-hook 'my/yas-after-hook))
  :diminish 'yas-minor-mode)
```

Inter-field navigation:

```
;; Go to end of active field
(defun yas-goto-end-of-active-field ()
  (interactive)
  (let* ((snippet (car (yas--snippets-at-point)))
         (position (yas--field-end (yas--snippet-active-field snippet))))
    (if (= (point) position)
        (move-end-of-line 1)
      (goto-char position))))

;; Go to start of active field
(defun yas-goto-start-of-active-field ()
  (interactive)
  (let* ((snippet (car (yas--snippets-at-point)))
         (position (yas--field-start (yas--snippet-active-field snippet))))
    (if (= (point) position)
        (move-beginning-of-line 1)
      (goto-char position))))

(define-key yas-keymap (kbd "C-a") 'yas/goto-start-of-active-field)
(define-key yas-keymap (kbd "C-e") 'yas/goto-end-of-active-field)
```

7.5.2 hippie-expand

```
;; Hippie expand-file-name
(global-set-key (kbd "M-/") 'hippie-expand)
;; Lisp-friendly hippie expand
(setq hippie-expand-try-functions-list
      '(try-expand-dabbrev
        try-expand-dabbrev-all-buffers
        try-expand-dabbrev-from-kill
        try-complete-lisp-symbol-partially
        try-complete-lisp-symbol))
```

7.6 Non-ASCII symbols

7.6.1 Encoding defaults

```
(set-terminal-coding-system 'utf-8)
(set-keyboard-coding-system 'utf-8)
(set-language-environment "UTF-8")
```

```
(prefer-coding-system 'utf-8)
(set-default-coding-systems 'utf-8)

(setq-default indent-tabs-mode nil)
(delete-selection-mode)
(global-set-key (kbd "RET") 'newline-and-indent)
```

7.6.2 Key translations

These are some keyboard translations for symbol assertions. Mostly this is just for English → Greek symbol translation. Need to insert an `alpha`? Type `C-c u` and then the `a` key.

```
(define-key key-translation-map (kbd "C-c u p")       (kbd ""))
(define-key key-translation-map (kbd "C-c u \"")       (kbd "\""))
(define-key key-translation-map (kbd "C-c u '")       (kbd "'"))
(define-key key-translation-map (kbd "C-c u x")       (kbd ""))
(define-key key-translation-map (kbd "C-c u i")       (kbd ""))
(define-key key-translation-map (kbd "C-c u l")       (kbd ""))
(define-key key-translation-map (kbd "C-c u a")       (kbd ""))
(define-key key-translation-map (kbd "C-c u b")       (kbd ""))
(define-key key-translation-map (kbd "C-c u e")       (kbd ""))
(define-key key-translation-map (kbd "C-c u d")       (kbd ""))
(define-key key-translation-map (kbd "C-c u z")       (kbd ""))
(define-key key-translation-map (kbd "C-c u s")       (kbd ""))
(define-key key-translation-map (kbd "C-c u <right>") (kbd "→"))
(define-key key-translation-map (kbd "C-c u <left>")  (kbd "←"))
(define-key key-translation-map (kbd "C-c u <up>")    (kbd "↑"))
(define-key key-translation-map (kbd "C-c u <down>")  (kbd "↓"))
```

7.6.3 `abbrev-mode`

```
(define-abbrev-table 'global-abbrev-table '(("alpha" "")
                                            ("inf" "")
                                            ("ar" "→")
                                            ("lambda" "")))
(abbrev-mode 1)
```

7.6.4 `char-menu`

`char-menu` is a useful package that presents a pop-up buffer for selecting non-ascii symbols. They can be categorized, and in that way easily organized,

filtered and selected. Bound to `C-c u <RET>` by default.

```
(use-package char-menu
  :commands char-menu
  :bind
  (("C-c u <RET>" . char-menu))
  :config
  (setq char-menu '((" Basic"      "-" "‘’" "“”" "..." "«»" "-")
                    (" Typography" "•" "©" "†" "‡" "°" "·" "§" "№" "")
                    (" Math"       "" "" "" "" "×" "±" "" "÷" "" "" "")
                    (" Arrows"     "←" "→" "↑" "↓" "" "" "" "")
                    (" Greek"      "" "" "Y" "" "" "" "" "" "" "" "" ""
                                   "" "" "" "" "" "" "" "" "" "" "" "")))))
```

7.6.5 math-symbols

`math-symbols` is a package of mine for using helm to select unicode symbols.

```
(use-package math-symbols
  :bind*
  ("C-c u TAB" . helm-math-sym-get-symbols))
```

7.7 Functions

7.7.1 just-one-space

```
(defun just-one-space-in-region (beg end)
  "Replace all whitespace in the region with single spaces"
  (interactive "r")
  (save-excursion
    (save-restriction
      (narrow-to-region beg end)
      (goto-char (point-min))
      (while (re-search-forward "\\s-+" nil t)
        (replace-match " ")))))
```

7.7.2 unfill-paragraph

```
;;; Stefan Monnier <foo at acm.org>. It is the opposite of fill-paragraph
(defun unfill-paragraph (&optional region)
  "Takes a multi-line paragraph and makes it into a single line of text."
  (interactive (progn (barf-if-buffer-read-only) '(t)))
```

```
  (let ((fill-column (point-max)))
    (fill-paragraph nil region)))

(define-key global-map (kbd "M-Q") 'unfill-paragraph)
```

7.7.3 die-tabs

```
(defun die-tabs ()
"use 2 spaces for tabs"
  (interactive)
  (set-variable 'tab-width 2)
  (mark-whole-buffer)
  (untabify (region-beginning) (region-end))
  (keyboard-quit))
```

7.7.4 prelude-move-beginning-of-line

```
;; Customized functions
(defun prelude-move-beginning-of-line (arg)
  "Move point back to indentation of beginning of line.

Move point to the first non-whitespace character on this line.
If point is already there, move to the beginning of the line.
Effectively toggle between the first non-whitespace character and
the beginning of the line.

If ARG is not nil or 1, move forward ARG - 1 lines first. If
point reaches the beginning or end of the buffer, stop there."
  (interactive "^p")
  (setq arg (or arg 1))

  ;; Move lines first
  (when (/= arg 1)
    (let ((line-move-visual nil))
      (forward-line (1- arg))))

  (let ((orig-point (point)))
    (back-to-indentation)
    (when (= orig-point (point))
      (move-beginning-of-line 1))))
```

```
(global-set-key (kbd "C-a") 'prelude-move-beginning-of-line)
```

7.7.5 defadvice kill-ring-save

```
(defadvice kill-ring-save (before slick-copy activate compile)
  "When called interactively with no active region, copy a single
line instead."
  (interactive
   (if mark-active (list (region-beginning) (region-end))
     (message "Copied line")
     (list (line-beginning-position)
           (line-beginning-position 2)))))
```

7.7.6 defadvice kill-region

```
(defadvice kill-region (before slick-cut activate compile)
  "When called interactively with no active region, kill a single
  line instead."
  (interactive
   (if mark-active (list (region-beginning) (region-end))
     (list (line-beginning-position)
           (line-beginning-position 2)))))
```

7.7.7 defadvice kill-line

```
;; kill a line, including whitespace characters until next non-whiepsace character
;; of next line
(defadvice kill-line (before check-position activate)
  (if (member major-mode
              '(emacs-lisp-mode scheme-mode lisp-mode
                                c-mode c++-mode objc-mode
                                latex-mode plain-tex-mode))
      (if (and (eolp) (not (bolp)))
          (progn (forward-char 1)
                 (just-one-space 0)
                 (backward-char 1)))))
```

7.7.8 yank-advised-indent-function

yank-indent-modes

```
;; taken from prelude-editor.el
;; automatically indenting yanked text if in programming-modes
(defvar yank-indent-modes
  '(LaTeX-mode TeX-mode)
  "Modes in which to indent regions that are yanked (or yank-popped).
Only modes that don't derive from `prog-mode' should be listed here.")
```

yank-indent-blacklisted-modes

```
(defvar yank-indent-blacklisted-modes
  '(python-mode slim-mode haml-mode)
  "Modes for which auto-indenting is suppressed.")
```

yank-advised-indent-threshol

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

yank-advised-indent-function

```
(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)))
```

7.7.9 defadvice yank

```
(defadvice yank (after yank-indent activate)
  "If current mode is one of 'yank-indent-modes,
indent yanked text (with prefix arg don't indent)."
  (if (and (not (ad-get-arg 0))
           (not (member major-mode yank-indent-blacklisted-modes))
           (or (derived-mode-p 'prog-mode)
               (member major-mode yank-indent-modes)))
      (let ((transient-mark-mode nil))
        (yank-advised-indent-function (region-beginning) (region-end)))))
```

7.7.10 defadvice yank-pop

```
(defadvice yank-pop (after yank-pop-indent activate)
  "If current mode is one of `yank-indent-modes',
```

```
indent yanked text (with prefix arg don't indent)."
  (when (and (not (ad-get-arg 0))
             (not (member major-mode yank-indent-blacklisted-modes))
             (or (derived-mode-p 'prog-mode)
                 (member major-mode yank-indent-modes)))
    (let ((transient-mark-mode nil))
      (yank-advised-indent-function (region-beginning) (region-end)))))
```

7.7.11 indent-buffer

```
;; prelude-core.el
(defun indent-buffer ()
  "Indent the currently visited buffer."
  (interactive)
  (indent-region (point-min) (point-max)))
```

7.7.12 prelude-indent-sensitive-modes

```
;; prelude-editing.el
(defcustom prelude-indent-sensitive-modes
  '(coffee-mode python-mode slim-mode haml-mode yaml-mode)
  "Modes for which auto-indenting is suppressed."
  :type 'list)
```

7.7.13 indent-region-or-buffer

```
(defun indent-region-or-buffer ()
  "Indent a region if selected, otherwise the whole buffer."
  (interactive)
  (unless (member major-mode prelude-indent-sensitive-modes)
    (save-excursion
      (if (region-active-p)
          (progn
            (indent-region (region-beginning) (region-end))
            (message "Indented selected region."))
        (progn
          (indent-buffer)
          (message "Indented buffer.")))
      (whitespace-cleanup))))

(global-set-key (kbd "C-c i") 'indent-region-or-buffer)
```

7.7.14 prelude-get-positions-of-line-or-region

```
;; add duplicate line function from Prelude. taken from prelude-core.el.
(defun prelude-get-positions-of-line-or-region ()
  "Return positions (beg . end) of the current line
or region."
  (let (beg end)
    (if (and mark-active (> (point) (mark)))
        (exchange-point-and-mark))
    (setq beg (line-beginning-position))
    (if mark-active
        (exchange-point-and-mark))
    (setq end (line-end-position))
    (cons beg end)))
```

7.7.15 prelude-smart-open-line

smart openline

```
(defun prelude-smart-open-line (arg)
  "Insert an empty line after the current line.
Position the cursor at its beginning, according to the current mode.
With a prefix ARG open line above the current line."
  (interactive "P")
  (if arg
      (prelude-smart-open-line-above)
    (progn
      (move-end-of-line nil)
      (newline-and-indent))))
```

7.7.16 prelude-smart-open-line-above

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

```
(global-set-key (kbd "M-o") 'prelude-smart-open-line)
(global-set-key (kbd "M-o") 'open-line)


(add-hook 'emacs-lisp-mode-hook
            (lambda ()
              (set (make-local-variable 'company-backends) '(company-elisp))))
```

7.7.17 toggle-comment-on-line

Comment out a line:

```
(defun toggle-comment-on-line ()
  "comment or uncomment current line"
  (interactive)
  (comment-or-uncomment-region (line-beginning-position) (line-end-position)))
```

7.7.18 add-file-name-to-clipboard

```
(defun add-file-name-to-clipboard ()
  "Put the current file name on the clipboard"
  (interactive)
  (let ((filename (if (equal major-mode 'dired,-mode)
                      default-directory
                    (buffer-file-name))))
    (message
     (if filename
         (with-temp-buffer
           (insert filename)
           (clipboard-kill-region (point-min) (point-max))))
       (kill-new default-directory))))

(global-set-key (kbd "C-x f") 'add-file-name-to-clipboard)
```

7.7.19 duplicate-line

```
(defun duplicate-line(arg)
  (interactive "P")
  (let* ((arg (if arg arg 1))
         (beg (save-excursion (beginning-of-line) (point)))
         (end (save-excursion (end-of-line) (point)))
```

```
        (line (buffer-substring-no-properties beg end)))
    (save-excursion
      (end-of-line)
      (print arg)
      (open-line arg)
      (setq num 0)
      (while (< num arg)
        (setq num (1+ num))
        (forward-line 1)
        (insert-string line)))))

(global-set-key (kbd "C-S-o") 'duplicate-line)
```

8 TODO Software Development [0/1]

```
(let ((inhibit-message nil))
  (message "Configuring development environments..."))
```

8.1 General Settings

8.1.1 General

Highlight current line when coding

```
(add-hook 'prog-mode-hook 'hl-line-mode)
```

Use helm-dash-install-docset to install docsets. Nice package!!

```
(use-package helm-dash)
```

Basic navigation

```
(define-key prog-mode-map (kbd "C-c e") 'end-of-defun)
(define-key prog-mode-map (kbd "C-c a") 'beginning-of-defun)
```

Code searching with `imenu`

```
(global-set-key (kbd "C-c i") 'imenu)
```

`hs-minor-mode` gives us the ability to fold and unfold code and comments

```
(add-hook 'prog-mode 'hs-minor-mode)

(global-set-key (kbd "C-c f t") 'hs-toggle-hiding)
(global-set-key (kbd "C-c f h") 'hs-hide-block)
(global-set-key (kbd "C-c f s") 'hs-show-block)
(global-set-key (kbd "C-c f a h") 'hs-hide-all)
(global-set-key (kbd "C-c f a s") 'hs-show-all)
```

`dired` should have a key for launching `magit`

```
(add-hook 'dired-mode-hook (lambda () (define-key dired-mode-map (kbd "z") 'magit-stat
```

8.1.2 Editing

```
(define-key prog-mode-map (kbd "<return>") 'newline-and-indent)
```

In programming modes, replace `backwards-kill-word` command with `backward-kill-sexp`.

```
(define-key prog-mode-map (kbd "M-S-<backspace>") 'backward-kill-sexp)
(define-key smartparens-mode-map (kbd "M-S-<backspace>") 'backward-kill-sexp)
(define-key prog-mode-map (kbd "M-S-d") 'kill-sexp)
(define-key smartparens-mode-map (kbd "M-S-d") 'kill-sexp)
```

1. clean-up-buffer-or-region

   Untabifies, indents and deletes trailing whitespace from buffer or region.

   ```
   (defun clean-up-buffer-or-region ()
     "Untabifies, indents and deletes trailing whitespace from buffer or region."
     (interactive)
     (save-excursion
       (unless (region-active-p)
         (mark-whole-buffer))
       (untabify (region-beginning) (region-end))
       (indent-region (region-beginning) (region-end))
       (save-restriction
         (narrow-to-region (region-beginning) (region-end))
         (delete-trailing-whitespace))))
   ```

2. cut-region

   Helper function.

```
(defun cut-region (beg end)
  (let ((str (copy-region-as-kill beg end)))
    (delete-region beg end)
    str))
```

8.1.3 Navigation

```
(use-package dumb-jump
 :config
(setq dumb-jump-selector 'helm))

(define-key prog-mode-map (kbd "C-S-j b") 'beginning-of-defun)
(define-key prog-mode-map (kbd "C-S-j f") 'end-of-defun)
```

1. Projectile

```
(use-package projectile
  :config
  (projectile-global-mode)
  (setq-default projectile-enable-caching t
                projectile-cache-file (concat
                                       saveplace-dir
                                       "projectile.cache")
                projectile-known-projects-file (concat
                                                saveplace-dir
                                                "projectile-bookmarks.eld"))
  :diminish 'projectile-mode)

(use-package helm-projectile
  :config
  (helm-projectile-on)
  (setq projectile-completion-system 'helm)
  (setq projectile-indexing-method 'alien)
  :diminish 'helm-projectile)

(setq tramp-default-method "ssh")
```

2. gtags

```
(use-package helm-gtags
  :commands helm-gtags-mode
```

```
:bind
(("C-c g a" . helm-gtags-tags-in-this-function)
 ("C-j" . helm-gtags-select)
 ("M-." . helm-gtags-dwim)
 ("M-," . helm-gtags-pop-stack)
 ("C-c <" . helm-gtags-previous-history)
 ("C-c >" . helm-gtags-next-history))
:init
; Enable helm-gtags-mode in Eshell for the same reason as above:
(add-hook 'dired-mode-hook 'helm-gtags-mode)
; Enable helm-gtags-mode in languages that GNU Global supports:
(add-hook 'eshell-mode-hook 'helm-gtags-mode)
; Enable helm-gtags-mode in Dired so you can jump to any tag when navigating pr
(add-hook 'c-mode-hook 'helm-gtags-mode)
(add-hook 'c++-mode-hook 'helm-gtags-mode)
(add-hook 'java-mode-hook 'helm-gtags-mode)
:config
(setq
 helm-gtags-ignore-case t
 helm-gtags-auto-update t
 helm-gtags-use-input-at-cursor t
 helm-gtags-pulse-at-cursor t
 helm-gtags-prefix-key "\C-cg"
 helm-gtags-suggested-key-mapping t)
)
```

8.1.4 Folding

Folding code, vim-style

```
(use-package vimish-fold
  :config
  (defun vimish-fold-defun-fold ()
    (interactive)
    (vimish-fold (save-excursion (beginning-of-defun) (point))
                 (save-excursion (end-of-defun) (point))))
  (defun vimish-fold-python-block-fold ()
    (interactive)
    (vimish-fold (save-excursion (python-nav-beginning-of-block) (point))
                 (save-excursion (python-nav-end-of-block) (point)))
```

```
        :bind*
        (("C-c 5 RET" . vimish-fold)
         ("C-c 5 s" . vimish-fold-unfold)
         ("C-c 5 S" . vimish-fold-unfold-all)
         ("C-c 5 d" . vimish-fold-delete)
         ("C-c 5 D" . vimish-fold-delete-all)
         ("C-c 5 TAB" . vimish-fold-toggle)
         ("C-c 5 <C-tab>" . vimish-fold-toggle-all)
         ("C-c 5 h" . vimish-fold-refold)
         ("C-c 5 H" . vimish-fold-refold-all)
         ("C-c 5 m" . vimish-fold-avy)
         ("C-c 5 f" . vimish-fold-defun-fold)
;         :map python-mode-map
;         ("C-c 5 b" . vimish-fold-python-block-fold)
))
```

8.1.5 Semantic

```
(semantic-mode 1)

(global-semanticdb-minor-mode 1)

(global-semantic-idle-scheduler-mode 1)

(global-semantic-stickyfunc-mode 1)
```

8.1.6 Compilation

```
(global-set-key (kbd "<f5>") (lambda ()
                               (interactive)
                               (setq-local compilation-read-command nil)
                               (call-interactively 'compile)))
```

8.1.7 Debugging

```
;; Setup GDB
(setq gdb-many-windows t
 ;; Non-nil means display source file containing the main routine at startup
 gdb-show-main t)
```

8.1.8 EMR

```
(use-package emr)
```

8.1.9 JSON

```
(use-package json-snatcher)
(use-package json-reformat)
(use-package json-mode)
```

8.1.10 REPL/Command Line

```
(use-package eval-in-repl
  :init
  (require 'eval-in-repl-ielm)
  (define-key emacs-lisp-mode-map (kbd "C-x <return>") 'eir-eval-in-ielm)
  (define-key lisp-interaction-mode-map (kbd "C-x <return>") 'eir-eval-in-ielm)
  (setq eir-ielm-eval-in-current-buffer t)
  (define-key Info-mode-map (kbd "<C-return>") 'eir-eval-in-ielm)

  (when (require 'cider nil t)
    (require 'eval-in-repl-cider)
      (define-key cider-mode-map (kbd "<C-return>") 'eir-eval-in-cider))

  (when (require 'slime nil t)
    (require 'eval-in-repl-slime)
    (add-hook 'lisp-mode-hook
              '(lambda ()
                 (local-set-key (kbd "<C-return>") 'eir-eval-in-slime))))

  (require 'eval-in-repl-shell)
  (add-hook 'sh-mode-hook
            '(lambda()
               (local-set-key (kbd "C-<return>") 'eir-eval-in-shell)))
  :config
  (setq eir-repl-placement 'left))
```

1. comint

```
(define-key comint-mode-map (kbd "M-r") 'comint-history-isearch-backward)
```

```
(defvar python-comint-output-limit 7000)

(defun python-truncate-comint-output (string)
  "Does not truncate STRING currently, in order to avoid dangling quotes and paren
  (let* ((line-lengths (mapcar 'length (split-string string "\n")))
         (line-max-size 120)
         ;; Avoid treating mulitline things like DataFrame output as junk to be r
         (size (apply '+ (mapcar
                          (lambda (l) (if (< l line-max-size) (* 0.4 l) l))
                          line-lengths))))
    (if (and (>  size python-comint-output-limit)
             (equal major-mode 'inferior-python-mode))
        (format-message
         "\nOutput character length (%s) exceeds character limit `python-comint-ou
         (length string)
         python-comint-output-limit)
      string)))

(add-hook 'comint-preoutput-filter-functions 'python-truncate-comint-output)
```

8.2 Python

8.2.1 General

Add line numbers in python buffers

```
(add-hook 'python-mode-hook 'linum-mode)
```

Use python3

```
(setq py-python-command "/usr/bin/python3")
```

8.2.2 python-mode

```
(use-package python-mode
  :bind
  ("C-c M-f" .   python-nav-forward-defun)
  ("C-c M-b" .   python-nav-backward-defun)
  ("C-c C-f" .   python-nav-forward-sexp)
  ("C-c C-b" .   python-nav-backward-sexp)
  ("C-c F"   .   python-nav-forward-block)
```

```
("C-c B"   .   python-nav-backward-block)
("C-c M-e" .   python-nav-end-of-defun)
("C-c M-a" .   python-nav-beginning-of-defun)
("C-c C-e" .   python-nav-end-of-statement)
("C-c C-a" .   python-nav-beginning-of-statement)
("C-c E"   .   python-nav-end-of-block)
("C-c A"   .   python-nav-beginning-of-block)
:config
(require 'eval-in-repl-python)
(add-hook 'python-mode-hook
          '(lambda ()
             (local-set-key (kbd "C-x <return>") 'eir-eval-in-python)))
(setq-local eir-jump-after-eval nil)
(define-key python-mode-map (kbd "C-c C-c") 'python-shell-send-code-and-step)
(define-key prog-mode-map (kbd "C-c C-e") 'python-nav-end-of-block)
(define-key python-mode-map (kbd "<return>") 'py-newline-and-indent)
(define-key prog-mode-map (kbd "C-c C-a") 'python-nav-beginning-of-block)
(define-key python-mode-map (kbd "C-c SPC") 'py-switch-to-shell)
(define-key python-mode-map (kbd "C-c C-SPC") 'python-switch-to-shell-same-window)
(setq python-shell-send-code-step-on-function t)
(add-hook 'python-mode-hook
          '(lambda ()
             (setq-local completion-at-point-functions nil))))
```

1. `python-send-code-and-step`

   Some functions that will help make dealing with .py/shell workflow more intuitive. Based on `ESS`.

   ```
   (defvar python-shell-send-code-step-on-function nil
     "If non-nil, step sending a function to the python interpreter.")

   (defun python-shell-step-after-defun ()
     (forward-paragraph)
     ;; when at last paragraph, don't step to beginning of "next" paragraph
     (unless (equal (line-number-at-pos (point))
                    (progn (forward-paragraph) (line-number-at-pos (point))))
       (backward-paragraph)
       (forward-line 1)))

   (defun python-shell-send-paragraph-and-step ()
     "Send current paragraph of code and move point to the beginning of next paragra
   ```

```
  (interactive)
  (save-excursion
    (let* ((beg (progn (forward-paragraph)
                       (backward-paragraph)
                       (unless (equal (line-number-at-pos) 1)
                         (forward-line)) (point)))
           (end (progn (forward-paragraph)
                       (unless (equal (line-number-at-pos (point))
                                      (line-number-at-pos (point-max)))
                         (forward-line -1))
                       (end-of-line)
                       (point)))
           (num-lines (1+ (- (line-number-at-pos end)
                             (line-number-at-pos beg)))))
      (python-shell-send-region beg end)
      (message (concat
                (format "Sent %d line%s "
                        num-lines
                        (if (equal num-lines 1) "" "s"))
                (if (equal num-lines 1)
                    (format "(line %d)" (line-number-at-pos end))
                  (format "(lines %d-%d)"
                          (line-number-at-pos beg)
                          (line-number-at-pos end)))
                " to Python interpreter")
               (line-number-at-pos beg)
               (line-number-at-pos end))))
  (python-shell-step-after-defun))

(defun python-current-defun ()
  "‘python-info-current-defun’ doesn't work when on a blank line, for some reason
  (interactive)
  (save-excursion (forward-paragraph)
                  (backward-paragraph)
                  (forward-line)
                  (python-info-current-defun)))

(defun python-shell-send-code-and-step (arg)
  (interactive "p")
  ;; xemacs doesn't have use-region-p
```

```
(let ((python-shell-send-code-step-on-function
       (if (equal arg 4) t nil)))
  (unless (python-shell-get-process)
    (run-python))
  (cond ((use-region-p)
         (let ((end (region-end))
               (beg (region-beginning)))
           (python-shell-send-region beg end)
           (message "Sent region between lines %d and %d to python interpreter"
                    (line-number-at-pos beg)
                    (line-number-at-pos end))
           (goto-char end)))
        ;; send function if in a function, else send block and iterate
        ((python-current-defun)
         (progn
           (python-shell-send-defun)
           (message "Sent '%s' function to python interpreter"
                    (propertize (python-current-defun) 'face
                                '(:foreground "#66D9EF")))
           (when python-shell-send-code-step-on-function
             (python-nav-end-of-defun)
             (forward-line -1)
             (python-shell-step-after-defun))))
        (t
         (python-shell-send-paragraph-and-step)))
  (when (not (equal arg 4))
    (python-shell-switch-to-shell))))
```

Replace the normal binding (which is set to `python-shell-send-file`) with this improved function.

8.2.3 `py-autopep8`

```
(use-package py-autopep8
  :init
  (add-hook 'elpy-mode-hook 'py-autopep8-enable-on-save))
```

8.2.4 `elpy-mode` DISABLED

```
;    (use-package elpy
;      :commands (elpy-mode elpy-enable)
```

```
;      :init
;      (setq elpy-rpc-backend "jedi"
;            elpy-syntax-check-command "pylint")
;      (defun elpy-on-python-mode ()
;        (elpy-mode)
;        (elpy-enable))
;      (add-hook 'python-mode-hook 'elpy-on-python-mode)
;      :bind
;      (("C-c ?" . elpy-doc))
;      :config
;      ;; (elpy-use-ipython)
;      (when (require 'flycheck nil t)
;        (setq elpy-modules (delq 'elpy-module-flymake elpy-modules))
;        (add-hook 'elpy-mode-hook 'flycheck-mode))
;      (add-hook 'elpy-mode-hook 'flycheck-mode))
```

8.2.5 anaconda-mode

`anaconda` (not to be confused with the Python package manager) is an alternative to `elpy`.

```
(use-package anaconda-mode
  :init
  (defun my-python-mode-hook ()
    (anaconda-mode)
    (anaconda-eldoc-mode))
  (add-hook 'python-mode-hook 'my-python-mode-hook)
  (add-hook 'inferior-python-mode (lambda () (add-to-list 'company-backends 'company-c
                                       ;    (add-hook 'inferior-python-mode (lambda (
  (eval-after-load "company" '(add-to-list 'company-backends 'company-anaconda))
  :bind
  ("C-c C-" . run-python)
  :config
  (use-package company-anaconda
    :init
    :after anaconda-mode
    :config
    (semantic-idle-summary-mode -1)
    (mapc (lambda (x)
            (let ((command-name (car x))
```

```
                (title (cadr x))
                (region-p (caddr x))
                predicate)
            (setq predicate (lambda ()
                              (and (anaconda-mode-running-p)
                                   (not (use-region-p))
                                   (not (sp-point-in-string-or-comment)))))
            ;; (emr-declare-command (intern
            ;;                       (format "anaconda-mode-%s"
            ;;                               (symbol-name command-name)))
            ;;   :title title :modes 'python-mode :predicate predicate)
            ))
        '((show-doc          "view documentation" t)
          (find-assignments  "find assignments"  t)
          (find-definitions  "find definitions"  t)
          (find-file         "find assignments"  t)
          (find-references "show usages" nil))))
  :diminish 'anaconda-mode)
```

8.2.6 ein

Ein is an Emacs front end for `jupyter notebook`.

```
(use-package ein
  ;; :bind*
  ;; ("C-c v n" . ein:worksheet-insert-cell-below)
  ;; ("C-c k" . ein:worksheet-kill-cell)
  ;; ("C-c v p" . ein:worksheet-insert-cell-above)
  ;; ("C-c v d" . ein:worksheet-split-cell-at-point)
  :config
  (setq url-proxy-services '(("no_proxy" . "127.0.0.1")))
  (add-hook 'ein:connect-mode-hook 'ein:jedi-setup)
  (setq ein:use-auto-complete-supe t
        ein:console-executable "/user/local/bin/ipython")
  (add-hook 'ein:notebook-multilang-mode (lambda ()
                                           (company-mode -1)
                                           (auto-complete-mode)))
  (add-hook 'ein:notebook-python-mode (lambda ()
                                        (company-mode -1)
                                        (auto-complete-mode)))
```

```
(custom-set-faces
 '(mumamo-background-chunk-major
   ((((class color) (min-colors 88) (background dark)) nil))))
(defun switch-color ()
  (interactive)
  "Switch default bg for ipython notebook."
  (face-remap-add-relative 'default '((:background "gray15")))
  (face-remap-add-relative ' '((:foreground "red"))))
(add-hook 'ein:notebook-multilang-mode-hook 'switch-color))
```

8.2.7 ob-ipython

Currently DO NOT attempt to fetch completions from jupyter. It does not yet support emacs due to a jupyter bug.

```
; none
```

8.2.8 company-anaconda

company-anaconda

8.2.9 Python Shell

```
(defun python-switch-to-shell-same-window ()
  (interactive)
  (let ((python-buffer (python-shell-get-buffer)))
    (if python-buffer
        (switch-to-buffer python-buffer)
      (message "No python process running"))))
```

1. IPython **DISABLED**

   see here for info on IPython 5 is not working properly in Emacs.

   ```
   ;  (setq python-shell-prompt-detect-failure-warning nil
   ;        py-shell-interpreter "ipython"
   ;        ;; use the wx backend, for both mayavi and matplotlib
   ;        python-shell-interpreter-args "--matplotlib=wx --pylab=wx --colors=DarkB
   ;        py-shell-switch-buffers-on-execute-p t
   ;        py-smart-indentation t
   ;        python-shell-completion-native-disabled-interpreters '("jupyter" "pypy"),
   ```

2. CPython

   I currently prefer using the `CPython` interpreter.

```
(setq python-shell-prompt-detect-failure-warning nil
      py-shell-interpreter "python"
      ;; setup the matplotlib backend and import pyplot.
      python-shell-interpreter-args "-i -c \"import matplotlib; matplotlib.use('qt
      py-shell-switch-buffers-on-execute-p t
      python-shell-enable-font-lock t
      py-smart-indentation t
      python-shell-completion-native-disabled-interpreters '("python" "pypy"))

;; subtly gray commmands
(set-face-attribute 'comint-highlight-input nil
                    :foreground "gray63" :weight 'bold)

(eval-after-load "python"
  '(setq python-shell-setup-code '((python-shell-completion-native-turn-off))))
```

8.3 JavaScript

8.3.1 js-comint

```
(use-package js-comint
  :config)
```

1. Code stepping

```
  (defvar js-comint-send-code-step-on-function nil
    "If non-nil, step sending a function to the js interpreter.")

  (defun js-comint-step-after-defun ()
    (forward-paragraph)
    ;; when at last paragraph, don't step to beginning of "next" paragraph
    (unless (equal (line-number-at-pos (point))
                   (progn (forward-paragraph) (line-number-at-pos (point))))
      (backward-paragraph)
      (forward-line 1)))

  (defun js-comint-send-paragraph-and-step ()
    "Send current paragraph of code and move point to the beginning of next paragr
```

```
  (interactive)
  (save-excursion
    (let* ((beg (progn (forward-paragraph)
                       (backward-paragraph)
                       (unless (equal (line-number-at-pos) 1)
                         (forward-line)) (point)))
           (end (progn (forward-paragraph)
                       (unless (equal (line-number-at-pos (point))
                                      (line-number-at-pos (point-max)))
                         (forward-line -1))
                       (end-of-line)
                       (point)))
           (num-lines (1+ (- (line-number-at-pos end)
                             (line-number-at-pos beg))))
           (str (buffer-substring-no-properties beg end)))
      (js-comint-send-string str)
      (message (concat
                (format "Sent %d line%s "
                        num-lines
                        (if (equal num-lines 1) "" "s"))
                (if (equal num-lines 1)
                    (format "(line %d)" (line-number-at-pos end))
                  (format "(lines %d-%d)"
                          (line-number-at-pos beg)
                          (line-number-at-pos end)))
                " to JavaScript interpreter")
               (line-number-at-pos beg)
               (line-number-at-pos end))))
  (js-comint-step-after-defun))

(defun js-comint-send-code-and-step ()
  (interactive)
  ;; xemacs doesn't have use-region-p
  (unless (js-comint-get-process)
    (js-comint-start-or-switch-to-repl))
  (cond ((use-region-p)
         (let* ((end (region-end))
                (beg (region-beginning))
                (str (buffer-substring-no-properties beg end)))
           (js-comint-send-string str)
```

```
                (message "Sent region between lines %d and %d to python interpreter"
                         (line-number-at-pos beg)
                         (line-number-at-pos end))
                (goto-char end)))
              (t
               (js-comint-send-paragraph-and-step))))

  (define-key js-mode-map (kbd "C-c C-c") 'js-comint-send-code-and-step)
```

8.3.2 js2-mode

```
(use-package js2-mode)
(add-to-list 'auto-mode-alist '("\\.js\\'" . js2-mode))
```

Better JavaScript `imenu`.

```
(add-hook 'js2-mode-hook #'js2-imenu-extras-mode)
```

1. js2-refactor

   ```
   (use-package js2-refactor
     :bind
     (:map
      js2-mode-map
      ("C-k" . js2r-kill)))
   ```

2. xref-js2

   ```
   (use-package xref-js2)
   ```

8.3.3 company-tern

```
(use-package company-tern)
(add-to-list 'company-backends 'company-tern)
(add-hook 'js2-mode-hook (lambda ()
                           (tern-mode)
                           (company-mode)))
;; Disable completion keybindings, use xref-js2 instead
(define-key tern-mode-keymap (kbd "M-.") nil)
(define-key tern-mode-keymap (kbd "M-,") nil)
```

8.4 C/C++

Use line numbers in C/C++ buffers

```
(add-hook 'c++-mode-hook 'linum-mode)
(add-hook 'c-mode-hook 'linum-mode)

(use-package rtags)
(use-package company-rtags
  :config
  (setq rtags-completions-enabled t)
  (add-to-list 'company-backends 'company-rtags)
  (setq rtags-autostart-diagnostics t)
  (rtags-enable-standard-keybindings))

(use-package helm-rtags
  :config
  (setq rtags-use-helm t))
```

Jump between header and source files

```
(global-set-key (kbd "C-M-,") 'ff-find-other-file)
```

Company-C-Headers enables the completion of C/C++ header file names using company-mode:

```
(use-package company-c-headers
   :commands (c++-mode c-mode)
   :config
   (use-package company-irony
     :config
     (eval-after-load 'company
       '(add-to-list 'company-backends 'company-irony))
     (add-hook 'irony-mode-hook 'company-irony-setup-begin-commands))
   (require 'cc-mode)
   (require 'semantic)

   (add-to-list 'company-backends 'company-c-headers)
   ;(add-to-list 'company-c-headers-path-system "/usr/include/c++/4.8/")

   ;; ********** Available C style: ***************
   ;; "gnu":    The default style for GNU projects
```

```
    ;; "k&r":    What Kernighan and Ritchie, the authors of C used in their book
    ;; "bsd":    What BSD developers use, aka "Allman style" after Eric Allman.
    ;; "whitesmith": Popularized by the examples that came with Whitesmiths C, an early
    ;; "stroustrup": What Stroustrup, the author of C++ used in his book
    ;; "ellemtel":  Popular C++ coding standards as defined by "Programming in C++, Rul
    ;;              Erik Nyquist and Mats Henricson, Ellemtel
    ;; "linux":  What the Linux developers use for kernel development
    ;; "python":What Python developers use for extension modules
    ;; "java":  The default style for java-mode (see below)
    ;; "user":  When you want to define your own style
    ;; ********************************************
    ;; set style to "linux"
    (setq c-default-style "linux"))

(use-package company-irony-c-headers)
;; Load with `irony-mode` as a grouped backend
(eval-after-load 'company
  '(add-to-list
    'company-backends '(company-irony-c-headers company-irony)))

(set-default 'semantic-case-fold t)

(add-to-list 'auto-mode-alist '("\\.h\\'" . c++-mode))

(defun my/cedet-hook ()
  (local-set-key "\C-c\C-j" 'semantic-ia-fast-jump)
  (local-set-key "\C-c\C-s" 'semantic-ia-show-summary))

(add-hook 'c-mode-common-hook 'my/cedet-hook)
(add-hook 'c-mode-hook 'my/cedet-hook)
(add-hook 'c++-mode-hook 'my/cedet-hook)

(add-hook 'c-mode-common-hook 'hs-minor-mode)

(use-package function-args
  :commands (c++-mode c-mode)
  :config
  (require cc-mode)
  (fa-config-default)
  (bind-key  [("C-c C-f C-h")] 'moo-complete c-mode-map)
```

```
  (bind-key [(control tab)] 'moo-complete c++-mode-map)
  (bind-key "C-c M-o s" 'fa-show c-mode-map)
  (bind-key "C-c M-o s" 'fa-show c++-mode-map))
```

Don't ask if I really want to compile:

```
(global-set-key (kbd "<f5>") (lambda ()
                               (interactive)
                               (setq-local compilation-read-command nil)
                               (call-interactively 'compile)))
```

Enable Emacs Development Environment (EDE) only in C/C++:

```
(require 'ede)

(global-ede-mode)
```

Configure clang executable

```
(setq company-clang-executable "clang++-3.5")
```

8.5 Lisp

`lispy` is a core lisp-editing package developed by abo-abo.

```
(use-package lispy
  :init
  (add-hook 'emacs-lisp-mode-hook (lambda () (lispy-mode 1)))
  ;(add-hook 'ielm-mode-hook (lambda () (lispy-mode 1)))
  (defun conditionally-enable-lispy ()
    (when (eq this-command 'eval-expression)
      (lispy-mode 1)))
  (add-hook 'minibuffer-setup-hook 'conditionally-enable-lispy)
  :config
  (lispy-define-key lispy-mode-map "u" 'lispy-backward)
  (lispy-define-key lispy-mode-map "o" 'lispy-forward)
  (lispy-define-key lispy-mode-map "*" 'lispy-wrap-round)
  (define-key lispy-mode-map (kbd "C-c (") (lambda () (interactive) (insert "(")))
  (define-key lispy-mode-map (kbd "C-c )") (lambda () (interactive) (insert ")")))
  (when (require 'smartparens nil t)
    (smartparens-mode -1)))
```

`lispy` has really annoying behavior for beginning of line movement. It will go to beginning of line, unless it's already there, in which case it will `backward-to-indentation`. This should be the other way around.

```
(defun lispy-move-beginning-of-line ()
  "Rewrite lispy-move-beginning-of-line to replace unintuitive
behavior"
  (interactive)
  (lispy--ensure-visible)
  (if (equal (point) (save-excursion (back-to-indentation) (point)))
      (move-beginning-of-line 1)
    (back-to-indentation)))
```

8.5.1 General Lisp Settings

Define hooks:

```
(defun my/general-lisp-hook ()
  (rainbow-delimiters-mode-enable))
```

Emacs Lisp hook:

```
(defun my/emacs-lisp-hook ()
      (my/general-lisp-hook)
      (turn-on-eldoc-mode))
```

Add hooks:

```
(add-hook 'emacs-lisp-mode-hook 'my/emacs-lisp-hook)
(add-hook 'ielm-mode-hook 'turn-on-eldoc-mode)
```

Enable rainbow-delimiters for lisp modes

```
(add-hook 'eval-expression-minibuffer-setup-hook 'my/general-lisp-hook)
(add-hook 'ielm-mode-hook             'my/general-lisp-hook)
(add-hook 'lisp-mode-hook            'my/general-lisp-hook)
(add-hook 'scheme-mode-hook           'my/general-lisp-hook)
```

Enable eldoc-mode in appropriate emacs lisp hooks

```
;; eldoc-mode shows documentation in the minibuffer when writing code
;; http://www.emacswiki.org/emacs/ElDoc
(add-hook 'lisp-interaction-mode-hook 'turn-on-eldoc-mode)
```

8.5.2 Emacs Lisp

```
(define-prefix-command 'Apropos-Prefix nil "Apropos (a,c,d,i,l,v,C-v)")
(global-set-key (kbd "C-h C-a") 'Apropos-Prefix)
(define-key Apropos-Prefix (kbd "a")   'apropos)
(define-key Apropos-Prefix (kbd "C-a") 'apropos)
(define-key Apropos-Prefix (kbd "c")   'apropos-command)
(define-key Apropos-Prefix (kbd "d")   'apropos-documentation)
(define-key Apropos-Prefix (kbd "i")   'info-apropos)
(define-key Apropos-Prefix (kbd "l")   'apropos-library)
(define-key Apropos-Prefix (kbd "v")   'apropos-variable)
(define-key Apropos-Prefix (kbd "C-v") 'apropos-value)
```

Turn on emacs lisp documentation

```
(eldoc-mode 1)
```

1. persistent-scratch

```
(use-package persistent-scratch
  :init
  ;; careful about keeping this order. Wouldn't want to save before
  ;; you restore :O
  (setq persistent-scratch-save-file (concat saveplace-dir "persistent-scratch"))
  (unless (file-exists-p persistent-scratch-save-file)
    (message "%s: %s" "Creating persistent-scratch file for first time at" persist
    (with-temp-buffer (write-file persistent-scratch-save-file))
    (persistent-scratch-save))
  (persistent-scratch-restore)
  (persistent-scratch-autosave-mode))

(defun my/bindkey-ielm-other-window ()
  (local-set-key (kbd "<f9>") (lambda ()
                                (let ((ielm-buffer (get-buffer "*ielm*")))
                                  (if (equal ielm-buffer nil)
                                      (ielm)
                                    (switch-to-buffer-other-window ielm-buffer)))
                                )))

(add-hook 'emacs-lisp-mode-hook 'my/bindkey-ielm-other-window)
```

Display possible symbol completions in a helm buffer:

```
(define-key global-map (kbd "C-c l c") 'helm-lisp-completion-at-point)
```

8.5.3 Clojure

1. clojure-mode

   My clojure-mode hook

   ```
   (defun my/clojure-mode-hook ()
     (my/general-lisp-hook)
     (subword-mode)
     (setq inferior-lisp-program "lein repl")
     (font-lock-add-keywords
      nil
      '(("(\\(facts?\\)"
         (1 font-lock-keyword-face))
        ("(\\(background?\\)"
         (1 font-lock-keyword-face))
        ))
     (define-clojure-indent (fact 1))
     (define-clojure-indent (facts 1))
     )

   (use-package clojure-mode
     :mode "\\.clj\\'"
     :init
     ;; Use clojure-mode for other file-name extensions
     (add-to-list 'auto-mode-alist '("\\.edn$" . clojure-mode))
     (add-to-list 'auto-mode-alist '("\\.boot$" . clojure-mode))
     (add-to-list 'auto-mode-alist '("\\.cljs.*$" . clojure-mode))
     (add-to-list 'auto-mode-alist '("lein-env" . enh-ruby-mode))
     ;; Define the clojure-mode-map prefix
     :config
       (use-package clojure-mode-extra-font-locking)
       (use-package flycheck-clojure)
       ;; A little more syntax highlighting
       (require 'clojure-mode-extra-font-locking)
                                              ;(use-package clj-refactor)
       (add-hook 'clojure-mode-hook 'my/clojure-mode-hook)
     )
   ```

2. cider-mode

```
(use-package cider
  :commands cider-mode
  :functions (cider-start-http-server cider-refresh cider-user-ns)
  :config
  (require 'clojure-mode-extra-font-locking)
  (progn
    (bind-keys
     ("C-' 1" . cider-visit-error-buffer))
    (bind-keys
     :map clojure-mode-map
     ("C" . cider-start-http-server)
     ( "C-c r" . cider-refresh)
     ("C-c u" . cider-user-ns))
    (bind-keys
     :map cider-mode-map
     ("C-c u" . cider-user-ns)
     ("C-‘" . cider-jack-in)))
  ;; Provides minibuffer documentation for the code you're typing into the repl
  (add-hook 'cider-mode-hook 'cider-turn-on-eldoc-mode)
  (setq cider-repl-pop-to-buffer-on-connect t) ;; Go right to the REPL buffer whe
  ;; it's finished connecting
  (setq cider-show-error-buffer nil) ;; When there's a cider error, show its buff
  ;; and switch to it
  (setq cider-auto-select-error-buffer t)
  (setq cider-repl-history-file "~/.emacs.d/cider-history") ;; Where to store the
  ;; cider history.
  (setq cider-repl-wrap-history t) ;; Wrap when navigating history.
  ;; CIDER and clojure-mode specific bindings:
  )

(defun cider-start-http-server ()
  (interactive)
  (cider-load-current-buffer)
  (let ((ns (cider-current-ns)))
    (cider-repl-set-ns ns)
    (cider-interactive-eval (format "(println '(def server (%s/start))) (println
    (cider-interactive-eval (format "(def server (%s/start)) (println server)" ns)

(defun cider-refresh ()
  (interactive)
```

```
    (cider-interactive-eval (format "(user/reset)")))

  (defun cider-user-ns ()
    (interactive)
    (cider-repl-set-ns "user"))
```

8.6 R

This is not downloaded via use-package and the package manager because I was having some problems with it throwing errors.

```
(add-to-list 'load-path "~/ess/lisp/")
(when (load "ess-site" t)
  (defun my/ess-mode-hook-funct ()
    (smartparens-mode t))
  (add-hook 'ess-mode-hook 'my/ess-mode-hook-funct))
```

Don't use that annoying indentation style with the comments, please

```
(setq ess-fancy-comments nil)

(defun dwc-ess-hook ()
  (smartparens-mode t))

(add-hook 'ess-mode-hook 'dwc-ess-hook)
```

8.7 Bash/shell

```
(add-hook 'sh-mode-hook (lambda () (setq tab-width 4)))
```

8.8 HTML

```
(defadvice sgml-delete-tag (after reindent-buffer activate) (clean-up-buffer-or-region

(advice-add 'sgml-close-tag :after 'clean-up-buffer-or-region)
```

8.8.1 sgml-copy-or-kill-tag

```
  (defun sgml-copy-or-kill-element (arg)
    (interactive "P")
    (let ((cut-tag (lambda ()
                     (let ((beg (point))
```

```
                        (end (save-excursion (sgml-skip-tag-forward 1)
                                             (point))))
                   (kill-ring-save beg end)
                   (when arg
                     (delete-region beg end))))))
      (save-excursion (if (sgml-beginning-of-tag)
                          (funcall cut-tag)
                        (if (progn (forward-char) (sgml-beginning-of-tag))
                            (funcall cut-tag)
                          (message "Not in an HTML tag."))))
      ))

(defun sgml-kill-element ()
  (interactive)
  (let ((current-prefix-arg 4))
    (call-interactively 'sgml-copy-or-kill-element))
  )

(define-key html-mode-map (kbd "C-c C-w") 'sgml-copy-or-kill-element)
(define-key html-mode-map (kbd "C-c C-k") 'sgml-kill-element)
```

8.9 Git [2/3]

8.9.1 magit

```
(use-package magit
  :commands magit-status
  :init
  (bind-key "C-c m s" 'magit-status))
```

Backup function to target when called.

```
(defun my/backup-specifics (file target)
"Copy file to target and apply function"
    (if (not (file-exists-p file))
      (write-region "" nil file)) ; create file
    (copy-file file target t))
```

Advise `magit-push` to backup `specifics.el` to a backup file in home.

```
(advice-add 'magit-push :around (lambda (push &rest args)
    (my/backup-specifics "~/.emacs.d/specifics.el" "~/.emacsSpecificsBackup.el")
    (apply push args)))
```

8.9.2 gist

```
(use-package gist)
```

8.9.3 TODO git-gutter disabled

```
(use-package git-gutter+
  :commands
  git-gutter-mode+
  :init
  ;; uncomment to enable git-gutter-mode
  ; (global-git-gutter+-mode)
  :config
  (setq git-gutter+-disabled-modes '(org-mode org))
  :diminish 'git-gutter+-mode)
```

9 DONE Org Mode [0/0]

```
(let ((inhibit-message nil))
  (message "Configuring org-mode..."))

(defun outline-down-heading ()
  (interactive)
  (org-up-heading-safe)
  (outline-forward-same-level 1))
(define-key org-mode-map (kbd "C-c C-.") 'outline-down-heading)
(define-key org-mode-map (kbd "C-c C-,") 'outline-up-heading)
```

9.1 General

```
(diminish 'eldoc-mode)
```

Modify org syntax table to treat the following characters as word constituents. This way, commands like forward-word will not treat them as word separators, which is annoying in org-mode.

```
(modify-syntax-entry ?_ "w" org-mode-syntax-table)
(modify-syntax-entry ?- "w" org-mode-syntax-table)
(modify-syntax-entry ?= "w" org-mode-syntax-table)
(modify-syntax-entry ?| "w" org-mode-syntax-table)
(modify-syntax-entry ?$ "w" org-mode-syntax-table)
```

```
(modify-syntax-entry ?# "w" org-mode-syntax-table)
(modify-syntax-entry ?* "w" org-mode-syntax-table)
(modify-syntax-entry '(133 . 140) "w" org-mode-syntax-table) ; ascii characters with c
```

`htmlize` is needed for pretty html syntax highlighting from org exports

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

Allow conversion to markdown. Useful for working with project readmes, etc., in org-mode.

```
(eval-after-load "org"
  '(require 'ox-md nil t))
```

Add my `org-notes` package. This package allows you to link two notes together.

```
(add-to-list 'load-path (concat lisp-dir "org-notes"))
(require 'org-notes)

(define-key org-mode-map (kbd "C-c L") 'org-notes-helm-link-notes)
```

Now can use the :ignore: tag to ignore just headings (not their content) on export.

```
(require 'ox-extra)
(ox-extras-activate '(ignore-headlines))
```

Some general defaults:

```
(setq org-deadline-warning-days 2
      ;; Paths to various locations in my personal organization workflow
      org-directory               personal-dir
      org-default-notes-file      (concat org-directory "todo.org")
      org-journal-file            (concat org-directory "journal.org")
      org-extend-today-until      4
      org-footnote-section        nil
      org-from-is-user-regexp     "\\<Dodge Coates\\>"
      org-goto-interface          'outline-path-completion
      org-goto-max-level          10
      org-scheduled-delay-days    0
```

```
      org-src-fontify-natively    t)
(set-face-attribute 'org-agenda-done nil
                    :foreground "olive drab"
                    :weight 'ultra-light
                    )
```

Some bindings for getting to org files faster

```
(global-set-key (kbd "C-c o e")
                (lambda () (interactive) (find-file (concat org-directory "everything.
```

```
(add-to-list 'load-path (concat lisp-dir "org-misc"))
(unless (require 'ob-misc nil t)
  (warn "Cannot load ob-misc"))
```

```
(global-set-key (kbd "C-c C-v ;") 'org-edit-src-headers)
(global-set-key (kbd "C-c C-v :") 'org-print-src-headers)
```

```
; (defun org-show-children-advice (funct &optional N)
;   (funcall funct 3))
; (advice-add 'org-show-children :around 'org-show-children-advice)
```

Use my `yas-org-pretty-symbols` package for making snippet expansion more manageable

```
(add-to-list 'load-path (concat lisp-dir "yas-org-pretty-symbols"))
(require 'yas-org-pretty-symbols)
```

Log notes into a drawer for a cleaner, less ambiguous presentation

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

Log notes with active headings

```
(setcdr (assoc 'note org-log-note-headings) "_Note taken on_ %T")
(setcdr (assoc 'state org-log-note-headings) "_State changed from_ *%S* $\\rightarrow$
```

9.2 Appearance

Org header appearance

```
(setq org-bullets-bullet-list '("•" "•" "•" "•"))
```

Load `atchka-org-theme`

```
(add-to-list 'load-path (concat lisp-dir "atchka-org"))
(add-to-list 'load-path (concat lisp-dir "atchka-org-blocks"))

(require 'atchka-org-blocks)
(atchka-org-minor-mode t)

(when (require 'atchka-org-theme nil t)
  (load-theme 'atchka-org))

(use-package org-agenda-property
:config)
```

Org warnings:

```
(set-face-attribute 'org-warning nil :foreground "red3" :weight 'bold)
```

Please truncate lines in org-mode. The last thing I want to see is a wrapped table, and everything should be paragraphed, anyway (M-q).

```
(setq org-startup-truncated t)
```

Install org bullets for pretty org headers

```
(use-package org-bullets)
```

Org-mode hook

```
(defun dwc-org-mode-appearance-hook ()
  (toggle-word-wrap)
  (flyspell-mode)
  (org-bullets-mode t)
  (hl-line-mode -1))

(add-hook 'org-mode-hook 'dwc-org-mode-appearance-hook)
```

Org will automatically convert the LaTex names for symbols to their utf-8 encodings (so long as they are preceded by a backslash.) For example, Γ is entered by typing "Gamma" preceded by a slash:

```
(setq-default org-pretty-entities-include-sub-superscripts t
              org-use-sub-superscripts nil)
```

Make Org latex fragments bigger:

```
(setq org-format-latex-options (plist-put org-format-latex-options :scale 1.4))
```

Hide certain symbols and markup

```
(setq org-hide-emphasis-markers t ;; Font lock should hide the emphasis marker
                                  ;; characters (inline code, bold, etc).
      org-hidden-keywords '(title)
      org-hide-leading-stars t)
```

Hide all Org source blocks by default.

```
(add-hook 'org-mode-hook 'org-babel-result-hide-all)
```

I add this advice to make the highlights on org agenda disappear when typing is going on in another buffer. Very distracting for me when a line an agenda buffer is highlighted in the other frame because I forgot my mouse cursor on it.

```
;; (defun org-agenda-highlight-entry-advice (func &rest args)
;;   (when (frame-pointer-visible-p)
;;     (funcall func args)))

;; (add-hook 'after-init-hook
;;           (lambda () (advice-add 'org-agenda-highlight-todo :around 'org-agenda-hig
```

Display inline images with a certain width to prevent overly large images

```
(setq org-image-actual-width '(600))
```

9.2.1 Abbrevs

Abbreviations

```
(defun org-create-org-abbrev-table (latex-keywords)
  "Produce an `abbrev-table' for org mode using LATEX-KEYWORDS.
LATEX-KEYWORDS is a list of latex keywords without backslashes
that orgmode also recognizes as the corresponding UTF-8 symbol.
For example, '('alpha' 'beta') will return (('alphaa'
'\\alpha') ('Alphaa' '\\Alpha') ('betaa' '\\beta') ('Betaa'
'\\Beta')), the idea being that one types the symbol name with an
```

```
extra character on the end, and abbrev will translate it to the
corresponding latex keyword, which org-mdoe will render as the
corresponding Unicode symbol."
  (apply 'append
         (mapcar
          (lambda (word-pair)
            (mapcar
             (lambda (wp)
               (list (cdr wp) (concat "\\" (car wp))))
             (list word-pair (cons (capitalize (car word-pair))
                                   (capitalize (cdr word-pair))))))
          latex-keywords)))
```

9.3 Commands

Edit source code in org-mode:

```
(bind-keys
   :map org-src-mode-map
   ("<C-return>" . org-edit-src-exit))
```

9.3.1 Automatically wrap org-mode markup

```
(defun insert-char-with-wrap (wrap-char-beg wrap-char-end &optional beg end)
  "Wrap region with wrap"
  (if (and beg end)
      (let ((beg-marker (set-marker (make-marker) beg))
            (end-marker (set-marker (make-marker) end)))
        (save-excursion
          (goto-char (marker-position beg-marker))
          (insert wrap-char-beg)
          (goto-char (marker-position end-marker))
          (insert (or wrap-char-end wrap-char-beg)))
        (goto-char (marker-position end-marker))
        (forward-char) t)
    (self-insert-command 1)))

(defun -insert-symbol (wrap-char-beg wrap-char-end)
  `(lambda (beg end)
     (interactive (if (region-active-p)
                      (list (region-beginning)
```

```
                                (region-end))
                        (list nil nil)))
      (insert-char-with-wrap ,wrap-char-beg ,wrap-char-end beg end)))

(defmacro insert-symbol (wrap-char-beg &optional wrap-char-end)
  (-insert-symbol wrap-char-beg wrap-char-end))
```

Bind the previously defined commands to their corresponding keys:

```
(define-key org-mode-map (kbd "_") (insert-symbol "_"))
(define-key org-mode-map (kbd "-") (insert-symbol "+"))
(define-key org-mode-map (kbd "/") (insert-symbol "/"))
(define-key org-mode-map (kbd "=") (insert-symbol "="))
(define-key org-mode-map (kbd "*") (insert-symbol "*"))
(define-key org-mode-map (kbd "~") (insert-symbol "~"))
```

9.4 Drill

```
(require 'org-drill)
```

9.5 Tasks and States

Tasks have multiple possible states. Below defines them and their transitions

```
(setq org-todo-keywords
      (quote ((sequence "TODO(t)" "NEXT(n)" "|" "DONE(d)")
              (sequence "LEARN(l)" "IMPROVE(r@/@)" "IDEA(e)" "VERIFY(v@/@)" "|" "NOTE(
              (sequence "BUG(b!)" "ISSUE(i!)" "FEATURE(f!)" "|" "FIXED(x!/@)")
              (sequence "WAITING(w@/!)" "HOLD(h@/!)" "|" "CANCELLED(c@/!)" "PHONE" "ME

(setq org-todo-keyword-faces
      (quote (("TODO" :foreground "#cd2626" :weight bold)
              ("NEXT" :foreground "#ff8c00" :weight bold)
              ("DONE" :foreground "forest green" :weight bold)
              ("FIXED" :foreground "#20b2aa" :weight bold)

              ("IDEA" :foreground "#ff6347" :weight bold)
              ("NOTE" :foreground "#1c86ee" :weight bold)
              ("CLARIFY" :foreground "#551a8b" :weight bold)
              ("LEARN" :foreground "#8a2be2" :weight bold)
              ("IMPROVE" :foreground "#a020f0" :weight bold)
```

```
              ("VERIFY" :foreground "#8b0000" :weight bold)
              ("UNDERSTOOD" :foreground "forest green" :weight bold)

              ("WAITING" :foreground "orange" :weight bold)
              ("HOLD" :foreground "magenta" :weight bold)
              ("CANCELLED" :foreground "gray" :weight bold)
              ;;
              ("BUG" :foreground "red" :weight bold)
              ("ISSUE" :foreground "Brown" :weight bold)
              ("FEATURE" :foreground "SaddleBrown" :weight bold)
              ;;
              ("MEETING" :foreground "forest green" :weight bold)
              ("PHONE" :foreground "forest green" :weight bold))))
```

Task triggers

```
(setq org-todo-state-tags-triggers
      (quote (("CANCELLED" ("CANCELLED" . t))
              ("WAITING" ("WAITING" . t))
              ("HOLD" ("WAITING") ("HOLD" . t))
              (done ("WAITING") ("HOLD"))
              ("TODO" ("WAITING") ("CANCELLED") ("HOLD"))
              ("NEXT" ("WAITING") ("CANCELLED") ("HOLD"))
              ("DONE" ("WAITING") ("CANCELLED") ("HOLD")))))
```

9.6 Agenda

```
(add-hook 'org-agenda-after-show-hook 'recenter)
```

```
(setq org-lowest-priority 69) ; make lowest priority possible 'E'
(setq org-default-priority 67) ; make default priority 'C'
(setq org-priority-faces
      '((65 . (:foreground "gray100" :weight ultrabold)) ; A
        (66 . (:foreground "gray80" :weight bold))       ; B
        (67 . (:foreground "gray70" :weight light))      ; C
        (68 . (:foreground "gray42" :slant italic))      ; D
        (69 . (:foreground "gray33" :slant italic))))    ; E
```

9.6.1 Appearance

```
(defface org-agenda-small-font-face
  '((t :height 95))
```

```
  "Temporary buffer-local face")

(defconst org-agenda-standard-height (face-attribute 'default :height))

(defvar org-agenda-window-width-threshold 80
  "Window width at which org agenda shrinks its font.")

(defun org-agenda-text-rescale ()
  (when (eq major-mode 'org-mode)
    (if (< (window-width) org-agenda-window-width-threshold)
        (buffer-face-set 'org-agenda-small-font-face)
      (buffer-face-set 'default))))

(defun toggle-agenda-text-rescale ()
  (interactive)
  (if (or
       (member 'org-agenda-text-rescale 'window-configuration-change-hook)
       (member 'org-agenda-text-rescale 'org-agenda-mode-hook))
      (progn
        (message "Turning OFF agenda text rescaling...")
        (remove-hook 'org-agenda-mode-hook 'org-agenda-text-rescale)
        (remove-hook 'window-configuration-change-hook 'org-agenda-text-rescale))
    (message "Turning ON agenda text rescaling...")
    (add-hook 'org-agenda-mode-hook 'org-agenda-text-rescale)
    (add-hook 'window-configuration-change-hook 'org-agenda-text-rescale))
  (cl-loop for buf in (buffer-list) do
           (with-current-buffer buf (org-agenda-text-rescale))))
```

9.6.2 Interface for agenda

```
(setq org-agenda-window-setup 'other-window)

(require 'ag-misc)
```

Don't show me tags in the agenda view.

```
(setq org-agenda-hide-tags-regexp "\\|*")
```

Do not dim blocked tasks

```
(setq org-agenda-dim-blocked-tasks nil)
```

Compact the block agenda view

```
(setq org-agenda-compact-blocks t)
```

Pressing `Q` in org agenda shouldn't kill the buffer and window. Usually I just want to glance at the agenda, then go back to what I was doing.

```
(define-key org-agenda-mode-map (kbd "q") 'quit-window)
```

Don't delete agenda buffer when quitting, just bury it

```
(setq org-agenda-sticky t)
(define-key org-agenda-mode-map (kbd "C-q") 'org-agenda-quit)

(require 'org-agenda)
(define-key org-agenda-mode-map "T" 'org-tags-view)
(define-key org-agenda-mode-map (kbd "C-c C-k") 'org-agenda-kill-entries)


;; these functions are in org-misc
(global-set-key (kbd "C-c C-j") 'org-switch-to-agenda-other-window)
(global-set-key (kbd "C-c C-S-j") 'org-get-agenda-other-window)
(global-set-key (kbd "C-c j") 'org-switch-to-agenda)
(global-set-key (kbd "C-c J") 'org-get-agenda)
```

Agenda custom commands

```
;; Custom agenda command definitions
(setq org-agenda-custom-commands
      (quote (("N" "Notes" tags "NOTE"
               ((org-agenda-overriding-header "Notes")
                (org-tags-match-list-sublevels t)))
              ("h" "Habits" tags-todo "STYLE=\"habit\""
               ((org-agenda-overriding-header "Habits")
                (org-agenda-sorting-strategy
                 '(todo-state-down effort-up category-keep))))
              (" " "Agenda"
               ((agenda "" nil)
                (tags "REFILE"
                      ((org-agenda-overriding-header "Tasks to Refile")
                       (org-tags-match-list-sublevels nil)))
```

```
(tags-todo "-CANCELLED/!"
           ((org-agenda-overriding-header "Stuck Projects")
            (org-agenda-skip-function 'bh/skip-non-stuck-projects)
            (org-agenda-sorting-strategy
             '(category-keep))))
(tags-todo "-HOLD-CANCELLED/!"
           ((org-agenda-overriding-header "Projects")
            (org-agenda-skip-function 'bh/skip-non-projects)
            (org-tags-match-list-sublevels 'indented)
            (org-agenda-sorting-strategy
             '(category-keep))))
(tags-todo "-CANCELLED/!NEXT"
           ((org-agenda-overriding-header (concat "Project Next Tasks"
                                                  (if bh/hide-schedule
                                                      ""
                                                    " (including WAITI
            (org-agenda-skip-function 'bh/skip-projects-and-habits-and
            (org-tags-match-list-sublevels t)
            (org-agenda-todo-ignore-scheduled bh/hide-scheduled-and-wa
            (org-agenda-todo-ignore-deadlines bh/hide-scheduled-and-wa
            (org-agenda-todo-ignore-with-date bh/hide-scheduled-and-wa
            (org-agenda-sorting-strategy
             '(todo-state-down effort-up category-keep))))
(tags-todo "-REFILE-CANCELLED-WAITING-HOLD/!"
           ((org-agenda-overriding-header (concat "Project Subtasks"
                                                  (if bh/hide-schedule
                                                      ""
                                                    " (including WAITI
            (org-agenda-skip-function 'bh/skip-non-project-tasks)
            (org-agenda-todo-ignore-scheduled bh/hide-scheduled-and-wa
            (org-agenda-todo-ignore-deadlines bh/hide-scheduled-and-wa
            (org-agenda-todo-ignore-with-date bh/hide-scheduled-and-wa
            (org-agenda-sorting-strategy
             '(category-keep))))
(tags-todo "-REFILE-CANCELLED-WAITING-HOLD/!"
           ((org-agenda-overriding-header (concat "Standalone Tasks"
                                                  (if bh/hide-schedule
                                                      ""
                                                    " (including WAITI
            (org-agenda-skip-function 'bh/skip-project-tasks)
```

```
                    (org-agenda-todo-ignore-scheduled bh/hide-scheduled-and-wa
                    (org-agenda-todo-ignore-deadlines bh/hide-scheduled-and-wa
                    (org-agenda-todo-ignore-with-date bh/hide-scheduled-and-wa
                    (org-agenda-sorting-strategy
                     '(category-keep))))
             (tags-todo "-CANCELLED+WAITING|HOLD/!"
                        ((org-agenda-overriding-header (concat "Waiting and Postpon
                                                               (if bh/hide-schedule
                                                                   ""
                                                                 " (including WAITI
                         (org-agenda-skip-function 'bh/skip-non-tasks)
                         (org-tags-match-list-sublevels nil)
                         (org-agenda-todo-ignore-scheduled bh/hide-scheduled-and-wa
                         (org-agenda-todo-ignore-deadlines bh/hide-scheduled-and-wa
             (tags "-REFILE/"
                   ((org-agenda-overriding-header "Tasks to Archive")
                    (org-agenda-skip-function 'bh/skip-non-archivable-tasks)
                    (org-tags-match-list-sublevels nil))))
            nil))))
```

1. Searching

```
(define-key org-agenda-mode-map (kbd "y") 'org-search-view)
```

9.6.3 Delimit prioirites DISABLED

Function to visually delimit priorities in the org-agenda-list buffer. **NOTE: Currently disabled**

```
(defun my-custom-agenda-fn ()
  (interactive)
  (save-excursion
    (let ((delimit "------------------------"))
      (org-agenda-goto-today)
      (dolist
          (priority '("\\[#A\\]" "\\[#B\\]" "\\[#C\\]" "\\[#D\\]" "\\[#E\\]"))
        (when (re-search-forward priority nil t)
          (goto-char (point-at-bol)) (insert (concat delimit "\n"))))
      (org-agenda-goto-today)
      (when (re-search-forward delimit nil t)
        (delete-region
```

```
            (progn (forward-visible-line 0) (point))
            (progn (forward-visible-line 1) (point))))
        )))

;  (add-hook 'org-agenda-finalize-hook 'my-custom-agenda-fn)
```

9.6.4 Update org-agenda-files

```
(setq org-agenda-file-regexp "\\`[^.].*\\.org\\'") ; default value
(defvar l '())
(defun load-org-agenda-files-recursively (dir) "Find all directories in DIR."
       (unless (file-directory-p dir) (error "Not a directory `%s'" dir))
       (unless (equal (directory-files dir nil org-agenda-file-regexp t) nil)
         (add-to-list 'l dir))
       (dolist (file (directory-files dir nil nil t))
         (unless (member file '("." ".."))
           (let ((file (concat dir file "/")))
             (when (file-directory-p file)
               (load-org-agenda-files-recursively file))))))

(load-org-agenda-files-recursively org-directory)
(org-store-new-agenda-file-list l)
```

9.7 Clock

```
;(define-key org-mode-map (kbd "C-c C-x C-o") 'org-clock-in)
```

9.7.1 tea-time

Tea time will play a sound of your choosing on a delay. It's a timer.

Sound command is set to `play` which is acquired on Ubuntu Linux with `sudo apt install sox`.

```
(use-package tea-time
  :init
  (setq org-clock-sound t)
  (global-set-key (kbd "C-c C-x s") 'tea-show-remaining-time)
  (global-set-key (kbd "C-c C-x n") 'tea-time)
  (setq tea-time-sound (concat user-emacs-directory "bowshootalert.wav"))
  (setq tea-time-sound-command "play %s"))
```

9.8 Babel

This should be waaaay easier to load, but for some reason I can't get these packages to play nicely and I'm tired and need to go to bed.

```
(use-package dash-functional)
(use-package f)
(use-package s)
(use-package ob-ipython)

(org-babel-do-load-languages
 'org-babel-load-languages
 '((ipython . t)))
```

Stop asking me if I want to evaluate code

```
(setq org-confirm-babel-evaluate nil)
```

Display images in the results section of the org buffer (like a normal notebook)

```
(add-hook 'org-babel-after-execute-hook 'org-display-inline-images 'append)

(org-babel-do-load-languages
 'org-babel-load-languages
 '((shell . t)
   (js . t)
   (clojure . t)
   (python . t)
   (org . t)
   (R . t)))
```

Editing source code

```
(define-key org-mode-map (kbd "C-c RET") 'org-edit-special)
(define-key org-src-mode-map (kbd "C-c k") 'org-exit-src-code)
(define-key org-src-mode-map (kbd "C-c RET") 'org-edit-src-exit)

(require 'ox-jupyter)
(require 'ox-ipynb)
```

Advice to fix annoying org bug that messes up font-locking after editing source blocks.

```
(defun org-edit-source-advice (funct &rest args)
  (funcall funct)
  (org-restart-font-lock))

(advice-add 'org-edit-src-exit :around 'org-edit-source-advice)
```

9.9 Capture

```
(defun my-org-capture-hook ()
  (abbrev-mode t))

(add-hook 'org-capture-mode-hook 'my-org-capture-hook)
```

9.9.1 Template Components

org-capture-template is set in a private file. Look up the documention for org-capture-template (and org-capture in general) to see how it might look. Some components I made for building templates

9.9.2 Captures

I use my org-capture-builder convenience package

```
(add-to-list 'load-path (concat lisp-dir "org-capture-builder"))
(require 'org-capture-builder)

(defvar general-org-file "~/personal/general.org")
```

Basic org capture set. I set many more in my `post.el` file in ~/personal.

```
(setq org-capture-templates
      (org-make-project-templates
       "g"
       nil
       "General"
       `(file general-org-file)
       :study t
       :project nil))
```

9.10 Footnotes

Prompt for footnote label instead of automatically producing one.

```
(setq org-footnote-auto-label 'confirm)

(define-key org-mode-map (kbd "C-c C-SPC") 'org-footnote-new)
```

9.11 Exports

9.11.1 ox-latex

```
(require 'ox-latex)
(add-to-list 'org-latex-packages-alist '("" "minted"))
(setq org-latex-listings 'minted)

(setq org-latex-pdf-process
      '("pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"
        "pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"
        "pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"))
```

9.11.2 html

```
(setq org-html-table-default-attributes
      '(:border "2" :cellspacing "0" :cellpadding "6" :rules "all" :frame
                "all"))

;; Style from http://gongzhitaao.org/orgcss/

;; (setq org-html-htmlize-output-type 'inline-css) ;; default
(setq org-html-htmlize-output-type 'css)
;; (setq org-html-htmlize-font-prefix "") ;; default
(setq org-html-htmlize-font-prefix "org-")
```

9.11.3 Lots of html

```
(setq org-html-style-default
"<style type=\"text/css\">
  html{font-family:sans-serif;
       -ms-text-size-adjust:100%;
       -webkit-text-size-adjust:100%}
  body{margin:0}
```

```
article,aside,details,figcaption,figure,footer,header,main,menu,nav,section,summary{
audio,canvas,progress,video{display:inline-block}
audio:not([controls]){display:none;
                      height:0}
progress{vertical-align:baseline}
[hidden],template{display:none}
a{background-color:transparent;
  -webkit-text-decoration-skip:objects}
a:active,a:hover{outline-width:0}
abbr[title]{border-bottom:none;
            text-decoration:underline;
            text-decoration:underline dotted}
b,strong{font-weight:inherit;
         font-weight:bolder}
dfn{font-style:italic}
h1{font-size:2em;
   margin:.67em 0}
mark{background-color:#ff0;
     color:#000}
small{font-size:80%}
sub,sup{font-size:75%;
        line-height:0;
        position:relative;
        vertical-align:baseline}
sub{bottom:-.25em}
sup{top:-.5em}
img{border-style:none}
svg:not(:root){overflow:hidden}
code,kbd,pre,samp{font-family:monospace;
                  font-size:1em}
figure{margin:1em 40px}
hr{box-sizing:content-box;
   height:0;
   overflow:visible}
button,input,select,textarea{font:inherit;
                             margin:0}
optgroup{font-weight:700}
button,input{overflow:visible}
button,select{text-transform:none}
[type=reset],[type=submit],button,html [type=button]{-webkit-appearance:button}
```

```
[type=button]::-moz-focus-inner,[type=reset]::-moz-focus-inner,[type=submit]::-moz-f

[type=button]:-moz-focusring,[type=reset]:-moz-focusring,[type=submit]:-moz-focusring
fieldset{border:1px solid silver;
         margin:0 2px;
         padding:.35em .625em .75em}
legend{box-sizing:border-box;
       color:inherit;
       display:table;
       max-width:100%;
       padding:0;
       white-space:normal}
textarea{overflow:auto}
[type=checkbox],[type=radio]{box-sizing:border-box;
                             padding:0}
[type=number]::-webkit-inner-spin-button,[type=number]::-webkit-outer-spin-button{he
[type=search]{-webkit-appearance:textfield;
              outline-offset:-2px}
[type=search]::-webkit-search-cancel-button,[type=search]::-webkit-search-decoration
::-webkit-input-placeholder{color:inherit;
                            opacity:.54}
::-webkit-file-upload-button{-webkit-appearance:button;
                             font:inherit}
body{width:95%;
     margin:2%;
     font:normal normal normal 17px/1.6em Helvetica,sans-serif;
     color:#333}
@media (min-width:769px){body{width:700px;
                              margin-left:5vw}
}
.title{margin:auto;
       color:#000}
.subtitle,.title{text-align:center}
.subtitle{font-size:medium;
          font-weight:700}
.abstract{margin:auto;
          width:80%;
          font-style:italic}
.abstract p:last-of-type:before{content:\"    \";
                                white-space:pre}
```

```
.status{font-size:90%;
        margin:2em auto}
[class^=section-number-]{margin-right:.5em}
#footnotes{font-size:90%}
.footpara{display:inline;
          margin:.2em auto}
.footdef{margin-bottom:1em}
.footdef sup{padding-right:.5em}
a{color:#527d9a;
  text-decoration:none}
a:hover{color:#035;
        border-bottom:1px dotted}
figure{padding:0;
       margin:0;
       text-align:center}
img{max-width:100%;
    vertical-align:middle}
@media (min-width:769px){img{max-width:85vw;
                             margin:auto}
}
.MathJax_Display{margin:0!important;
                 width:90%!important}
h1,h2,h3,h4,h5,h6{color:#a5573e;
                  line-height:1.6em;
                  font-family:Georgia,serif}
h4,h5,h6{font-size:1em}
dt{font-weight:700}
table{margin:auto;
      border-top:2px solid;
      border-collapse:collapse}
table,thead{border-bottom:2px solid}
table td+td,table th+th{border-left:1px solid gray}
table tr{border-top:1px solid #d3d3d3}
td,th{padding:5px 10px;
      vertical-align:middle}
caption.t-above{caption-side:top}
caption.t-bottom{caption-side:bottom}
th.org-center,th.org-left,th.org-right{text-align:center}
td.org-right{text-align:right}
td.org-left{text-align:left}
```

```
td.org-center{text-align:center}
code{padding:2px 5px;
     margin:auto 1px;
     border:1px solid #ddd;
     border-radius:3px;
     background-clip:padding-box;
     color:#333;
     font-size:80%}
blockquote{margin:1em 2em;
           padding-left:1em;
           border-left:3px solid #ccc}
kbd{background-color:#f7f7f7;
    font-size:80%;
    margin:0 .1em;
    padding:.1em .6em}
.todo{color:red}
.done,.todo{font-family:Lucida Console,monospace}
.done{color:green}
.priority{color:orange}
.priority,.tag{font-family:Lucida Console,monospace}
.tag{background-color:#eee;
     font-size:80%;
     font-weight:400;
     padding:2px}
.timestamp{color:#bebebe}
.timestamp-kwd{color:#5f9ea0}
.org-right{margin-left:auto;
           margin-right:0;
           text-align:right}
.org-left{margin-left:0;
          margin-right:auto;
          text-align:left}
.org-center{margin-left:auto;
            margin-right:auto;
            text-align:center}
.underline{text-decoration:underline}
#postamble p,#preamble p{font-size:90%;
                         margin:.2em}
p.verse{margin-left:3%}
pre{border:1px solid #ccc;
```

```
    box-shadow:3px 3px 3px #eee;
    font-family:Lucida Console,monospace;
    margin:1.2em;
    padding:8pt}
pre.src{overflow:auto;
        padding-top:1.2em;
        position:relative;
        font-size:80%}
pre.src:before{background-color:#fff;
               border:1px solid #000;
               display:none;
               padding:3px;
               position:absolute;
               right:10px;
               top:.6em}
pre.src:hover:before{display:inline}
pre.src-sh:before{content:'sh'}
pre.src-bash:before{content:'bash'}
pre.src-emacs-lisp:before{content:'Emacs Lisp'}
pre.src-R:before{content:'R'}
pre.src-org:before{content:'Org'}
pre.src-c+:before{content:'C++'}
pre.src-c:before{content:'C'}
pre.src-html:before{content:'HTML'}
pre.example{overflow:auto;
            padding-top:1.2em;
            position:relative;
            font-size:80%}
.inlinetask{background:#ffc;
            border:2px solid gray;
            margin:10px;
            padding:10px}
#org-div-home-and-up{font-size:70%;
                     text-align:right;
                     white-space:nowrap}
.linenr{font-size:smaller}
.code-highlighted{background-color:#ff0}
#bibliography{font-size:90%}
#bibliography table{width:100%}
```

```
  .creator{display:block}@media (min-width:769px){.creator{display:inline;
                                                           float:right}}</style>")
```

9.12 Functions

```
(define-key org-mode-map (kbd "C-M-<return>") 'org-insert-subheading)
```

9.12.1 org-path-completion

```
(setq org-goto-interface 'outline-path-completion
      org-goto-max-level 10)
```

9.12.2 org-capture

org-capture allows you to take a note anywhere, which it will write to the org-default-notes-file.

```
;; this is set in ~/.emacs.d/custom/setup-specifics.el
(global-set-key (kbd "C-c c") 'org-capture)
```

9.12.3 org-back-to-top-level-heading

```
;; move point to top-level heading
(defun org-back-to-top-level-heading ()
  "Go back to the current top level heading."
  (interactive)
  (or (re-search-backward "^\* " nil t)
      (goto-char (point-min))))

;; make todo's check recursively when determining the number of todo's under it
(setq org-hierarchical-todo-statistics nil)
```

9.12.4 org-summary-todo

This is for making sure that the top-level todo automatically is marked DONE if all sub-levels are DONE. TODO otherwise.

```
(defun org-summary-todo (n-done n-not-done)
  "Switch entry to DONE when all subentries are done, to TODO otherwise."
  (let (org-log-done org-log-states)   ; turn off logging
    (org-todo (if (= n-not-done 0) "DONE" "TODO"))))

(add-hook 'org-after-todo-statistics-hook 'org-summary-todo)
```

9.12.5 jump-to-org-agenda

This is a snippet from John Wiegley. It shows org agenda after emacs has been idle for a certain amount of time.

```
(defun jump-to-org-agenda ()
  (interactive)
  (unless (> (apply '+ (mapcar
                        (lambda (buf) (if (string-match "Org Agenda*" (buffer-name buf
                        (buffer-list))) 0)
    (let ((buf (get-buffer "*Org Agenda*"))
          wind)
      (if buf
          (if (setq wind (get-buffer-window buf))
              (select-window wind)
            (if (called-interactively-p)
                (progn
                  (select-window (display-buffer buf t t))
                  (org-fit-window-to-buffer)
                  ;; (org-agenda-redo)
                  )
              (with-selected-window (display-buffer buf)
                (org-fit-window-to-buffer)
                ;; (org-agenda-redo)
                )))
        (call-interactively 'org-agenda-list))))
  ;;(let ((buf (get-buffer "*Calendar*")))
  ;;  (unless (get-buffer-window buf)
  ;;    (org-agenda-goto-calendar)))
  )

; (run-with-idle-timer 500 t 'jump-to-org-agenda)
```

9.12.6 create-tasks-heading

Insert a new Task heading. These are used by org-capture. ID is saved in kill ring.

```
(defun create-tasks-heading ()
  (interactive)
  (save-excursion
```

```
    (org-insert-subheading nil)
    (insert "TODO [/]")
    (backward-char)
    (org-ctrl-c-ctrl-c)
    (kill-new (org-id-get-create))
    )
  )
```

10 DONE Emacs as a Auxiliary Interface [0/0]

```
(message "Configuring Emacs as an auxiliary interface...")
```

10.1 Terminal

Track eshell sessions:

```
(defun eshell-default-prompt-function ()
  (interactive)
  (concat
   (system-name) ":"
   (propertize (car (last (split-string (pwd-repl-home (eshell/pwd)) "/"))) 'face
               `(:foreground "#cdcd00"))
   (or (curr-dir-git-branch-string (eshell/pwd)))
   (propertize "# " 'face 'default)))

(defun eshell-short-prompt-function ()
  (interactive)
  (concat (format-time-string "#" (current-time))
          (if (= (user-uid) 0) " # " " ")))

(defun eshell-swap-prompt ()
  (interactive)
  (setq eshell-prompt-function
        (if (eq eshell-prompt-function
                'eshell-default-prompt-function)
          'eshell-short-prompt-function
        'eshell-default-prompt-function))
  (eshell-send-input))

(add-hook 'eshell-mode-hook
```

```
          (lambda () (define-key eshell-mode-map (kbd "C-c C-;")
                  'eshell-swap-prompt)))

(defvar eshell-previous-buffer nil)
(defvar eshell-session-alist nil)

(defun dwc-switch-to-terminal (arg)
  (interactive "p")
  (if (eq arg 4)
      (progn
        (let ((sesh (alist-get (current-buffer) eshell-session-alist)))
          (if sesh
              (progn
                (setq eshell-previous-buffer (current-buffer))
                (eshell sesh))
            (let ((len (length eshell-session-alist)))
              (push (cons (current-buffer) len) eshell-session-alist)
              (eshell len)))))
    (if (string-match-p (regexp-quote eshell-buffer-name)
                        (buffer-name (current-buffer)))
        (if eshell-previous-buffer
            (switch-to-buffer eshell-previous-buffer)
          (previous-buffer))
      (setq eshell-previous-buffer (current-buffer))
      (eshell))))

(define-key global-map (kbd "C-x j") 'dwc-switch-to-terminal)
```

`eshell` is useful for quick shell stuff. It's not a full-fledged terminal, so it's usage is limited. The following code snippet is taken from here, and is by *Liang Zan*.

```
(setq eshell-history-size 1024)
(setq eshell-prompt-regexp "^[^#$]*[#$] ")

(load "em-hist")           ; So the history vars are defined
(if (boundp 'eshell-save-history-on-exit)
    (setq eshell-save-history-on-exit t)) ; Don't ask, just save
;(message "eshell-ask-to-save-history is %s" eshell-ask-to-save-history)
(if (boundp 'eshell-ask-to-save-history)
```

```
    (setq eshell-ask-to-save-history 'always)) ; For older(?) version
;(message "eshell-ask-to-save-history is %s" eshell-ask-to-save-history)

(defun eshell/ef (fname-regexp &rest dir) (ef fname-regexp default-directory))


;;; ---- path manipulation

(defun pwd-repl-home (pwd)
  (interactive)
  (let* ((home (expand-file-name (getenv "HOME")))
   (home-len (length home)))
    (if (and
   (>= (length pwd) home-len)
   (equal home (substring pwd 0 home-len)))
  (concat "~" (substring pwd home-len))
      pwd)))

(defun curr-dir-git-branch-string (pwd)
  "Returns current git branch as a string, or the empty string if
PWD is not in a git repo (or the git command is not found)."
  (interactive)
  (when (and (eshell-search-path "git")
             (locate-dominating-file pwd ".git"))
    (let ((git-output (shell-command-to-string (concat "cd " pwd " && git branch | gre
      (propertize (concat "["
              (if (> (length git-output) 0)
                  (substring git-output 0 -1)
                "(no branch)")
              "]") 'face `(:foreground "#2e8b57"))
      )))


(setq eshell-prompt-function
      (lambda ()
        (concat
         (system-name) ":"
         (propertize (car (last (split-string (pwd-repl-home (eshell/pwd)) "/"))) 'fac
                     `(:foreground "#cdcd00"))
         (or (curr-dir-git-branch-string (eshell/pwd)))
```

```
        (propertize "# " 'face 'default))))

(setq eshell-highlight-prompt nil)

(defun eshell-next-prompt (n)
  "Move to end of Nth next prompt in the buffer.
See `eshell-prompt-regexp'."
  (interactive "p")
  (re-search-forward eshell-prompt-regexp nil t (or n -1)))
```

10.2 Edit With Emacs (Google Chrome extension)

```
;; Edit With Emacs (Google Chrome Extension)
(add-to-list 'load-path (concat lisp-dir "edit-with-emacs"))
(use-package edit-server
  :config
  (edit-server-start))
```

10.3 Gghat

```
(setq jabber-account-list
      '(("dodge.w.coates@gmail.com"
         (:network-server . "talk.google.com")
         (:connection-type . ssl))))
```

10.4 IRC

10.4.1 ERC

```
(use-package erc
  :defer t
  :init
  (defun my/erc-mode-hook ()
    (if (equal word-wrap nil)
     (toggle-word-wrap)))
  :config
  (add-hook 'erc-mode-hook 'my/erc-mode-hook)
  (setq erc-hide-list '("PART" "QUIT" "JOIN"))
  (setq erc-autojoin-channels-alist '(("freenode.net"
                                       "#emacs"))
        erc-server "irc.freenode.net"
```

```
        erc-nick "dwc1")
  (defun erc-cmd-DEOPME ()
    "Deop myself from current channel."
    (erc-cmd-DEOP (format "%s" (erc-current-nick))))
  )
```

10.5 SSH

10.5.1 Tramp

I edit ~/.ssh/config to add something like the following
Host my-repo HostName mygitserverdomain.com Port 22 User repository
`C-x C-f /ssh:repo/some/path/to/a/file` will ssh via `tramp-mode` corresponding the path in user@host (my-repo@mygitserverdomain.com:22/some/path/to/a/file).

```
(setq tramp-default-method "ssh")
```

Get autocompletions in tramp:

```
(setq my-tramp-ssh-completions
      '((tramp-parse-sconfig "~/.ssh/config")
        (tramp-parse-shosts "~/.ssh/known_hosts")))
(mapc (lambda (method)
        (tramp-set-completion-function method my-tramp-ssh-completions))
      '("fcp" "rsync" "scp" "scpc" "scpx" "sftp" "ssh"))
```

10.6 StackExchange

```
(use-package sx)
```

10.7 Email

```
;; (require 'nnir)

;; ;; @see http://www.emacswiki.org/emacs/GnusGmail#toc1
;; (setq gnus-select-method '(nntp "news.gmane.org")) ;; if you read news groups

;; ;; ask encryption password once
;; (setq epa-file-cache-passphrase-for-symmetric-encryption t)

;; ;; @see http://gnus.org/manual/gnus_397.html
;; (add-to-list 'gnus-secondary-select-methods
```

```
;;              '(nnimap "gmail"
;;                       (nnimap-address "imap.gmail.com")
;;                       (nnimap-server-port 993)
;;                       (nnimap-stream ssl)
;;                       (nnir-search-engine imap)
;;                       ; @see http://www.gnu.org/software/emacs/manual/html_node/gnu
;;                       ;; press 'E' to expire email
;;                       (nnmail-expiry-target "nnimap+gmail:[Gmail]/Trash")
;;                       (nnmail-expiry-wait 90)))

;; ;; OPTIONAL, the setup for Microsoft Hotmail
;; (add-to-list 'gnus-secondary-select-methods
;;              '(nnimap "hotmail"
;;                       (nnimap-address "imap-mail.outlook.com")
;;                       (nnimap-server-port 993)
;;                       (nnimap-stream ssl)
;;                       (nnir-search-engine imap)
;;                       (nnmail-expiry-wait 90)))

;; (setq gnus-thread-sort-functions
;;       '(gnus-thread-sort-by-most-recent-date
;;         (not gnus-thread-sort-by-number)))

;; ; NO 'passive
;; (setq gnus-use-cache t)

;; ;; BBDB: Address list
;; ;(add-to-list 'load-path "/where/you/place/bbdb/")
;; ;(require 'bbdb)
;; ;(bbdb-initialize 'message 'gnus 'sendmail)
;; ;(add-hook 'gnus-startup-hook 'bbdb-insinuate-gnus)
;; ;(setq bbdb/mail-auto-create-p t
;; ;      bbdb/news-auto-create-p t)

;; ;; auto-complete emacs address using bbdb UI
;; ;(add-hook 'message-mode-hook
;; ;          '(lambda ()
;; ;             (flyspell-mode t)
;; ;             (local-set-key "<TAB>" 'bbdb-complete-name)))
```

```
;; ;; Fetch only part of the article if we can.
;; ;; I saw this in someone's .gnus
;; (setq gnus-read-active-file 'some)

;; ;; open attachment
;; (eval-after-load 'mailcap
;;   '(progn
;;      (cond
;;       ;; on OSX, maybe change mailcap-mime-data?
;;       ((eq system-type 'darwin))
;;       ;; on Windows, maybe change mailcap-mime-data?
;;       ((eq system-type 'windows-nt))
;;       (t
;;        ;; Linux, read ~/.mailcap
;;        (mailcap-parse-mailcaps)))))

;; ;; Tree view for groups.
;; (add-hook 'gnus-group-mode-hook 'gnus-topic-mode)

;; ;; Threads!  I hate reading un-threaded email -- especially mailing
;; ;; lists.  This helps a ton!
;; (setq gnus-summary-thread-gathering-function 'gnus-gather-threads-by-subject)

;; ;; Also, I prefer to see only the top level message.  If a message has
;; ;; several replies or is part of a thread, only show the first message.
;; ;; `gnus-thread-ignore-subject' will ignore the subject and
;; ;; look at 'In-Reply-To:' and 'References:' headers.
;; (setq gnus-thread-hide-subtree t)
;; (setq gnus-thread-ignore-subject t)

;; ;; Read HTML mail
;; ;; You need install the command line web browser 'w3m' and Emacs plugin 'w3m'
;; // (use-package w3m)
;; (setq mm-text-html-renderer 'w3m)

;; ;; Setup to send email through SMTP
;; (setq message-send-mail-function 'smtpmail-send-it
;;       smtpmail-default-smtp-server "smtp.gmail.com"
;;       smtpmail-smtp-service 587
```

```
;;       smtpmail-local-domain "homepc")
;; ;; http://www.gnu.org/software/emacs/manual/html_node/gnus/_005b9_002e2_005d.html
;; (setq gnus-use-correct-string-widths nil)

;; (eval-after-load 'gnus-topic
;;   '(progn
;;      (setq gnus-message-archive-group '((format-time-string "sent.%Y")))
;;      (setq gnus-server-alist '(("archive" nnfolder "archive" (nnfolder-directory "~
;;                                 (nnfolder-active-file "~/Mail/archive/active")
;;                                 (nnfolder-get-new-mail nil)
;;                                 (nnfolder-inhibit-expiry t))))

;;      (setq gnus-topic-topology '(("Gnus" visible)
;;                                  (("misc" visible))
;;                                  (("hotmail" visible nil nil))
;;                                  (("gmail" visible nil nil))))

;;      (setq gnus-topic-alist '(("hotmail" ; the key of topic
;;                                "nnimap+hotmail:Inbox"
;;                                "nnimap+hotmail:Drafts"
;;                                "nnimap+hotmail:Sent"
;;                                "nnimap+hotmail:Junk"
;;                                "nnimap+hotmail:Deleted")
;;                               ("gmail" ; the key of topic
;;                                "INBOX"
;;                                "[Gmail]/Sent Mail"
;;                                "[Gmail]/Trash"
;;                                "Sent Messages"
;;                                "Drafts")
;;                               ("misc" ; the key of topic
;;                                "nnfolder+archive:sent.2015-12"
;;                                "nnfolder+archive:sent.2016"
;;                                "nnfolder+archive:sent.2017"
;;                                "nndraft:drafts")
;;                               ("Gnus")))))
```

10.8 PDF viewing

pdf-tools is an improvment on Emacs' doc-view mode. Lets us search, follow links, and more.

```
;; (use-package pdf-tools
;;   :init
;;   (pdf-tools-install t t t))
```

10.9 Google

10.9.1 gnugol

An interface for Google. Very nice. See here for installation instructions.

```
(require 'gnugol)
```

10.10 Chess USER SPECIFIC SETTING TO SET

```
(use-package chess
  :config
  (setq chess-images-directory
        (concat user-emacs-directory
                "dependencies/chess-2.0.4/pieces/xboard")
        chess-images-default-size 58
        ;; I like it in current frame.
        chess-images-separate-frame nil
        ))
```

The following code I've added to fix the undersizing of new chess buffers.

```
(defun dwc-chess-images-popup ()
  (unless chess-images-size
    (chess-error 'no-images))

  (let* ((size (float (+ (* (or chess-images-border-width 0) 8)
                         (* chess-images-size 8))))
         (max-char-height (ceiling (/ size (frame-char-height))))
         (max-char-width  (ceiling (/ size (frame-char-width)))))
    ;; create the frame whenever necessary
    (if chess-images-separate-frame
        (chess-display-popup-in-frame
         (+ max-char-height 2)
         max-char-width
         (cdr (assq 'font (frame-parameters))))
      (dwc-chess-display-popup-in-window nil (+ max-char-height 1)))))
(setq chess-images-popup-function 'dwc-chess-images-popup)
```

```
(defun dwc-chess-display-popup-in-window (&optional max-h min-h max-w min-w)
  "Popup the given DISPLAY, so that it's visible to the user."
  (unless (get-buffer-window (current-buffer))
    (if (> (length (window-list)) 1)
        (fit-window-to-buffer (display-buffer (current-buffer))
                              max-h min-h max-w min-w)
      (display-buffer (current-buffer)))))
```

10.11 Calculator

Emacs calculator is very cool once you get used to it. Access it with `C-x * *`, view the rest of the options with `C-x *` prefix.

```
(global-set-key (kbd "C-c u =") 'calc) ;; directly access calculator
```

10.11.1 Usage

Calc normally uses RPN notation. You may be familiar with the RPN system from Hewlett-Packard calculators, FORTH, or PostScript. (Reverse Polish Notation, RPN, is named after the Polish mathematician Jan Lukasiewicz.)

The central component of an RPN calculator is the stack. A calculator stack is like a stack of dishes. New dishes (numbers) are added at the top of the stack, and numbers are normally only removed from the top of the stack.

In an operation like '2+3', the 2 and 3 are called the operands and the '+' is the operator. In an RPN calculator you always enter the operands first, then the operator. Each time you type a number, Calc adds or pushes it onto the top of the Stack. When you press an operator key like +, Calc pops the appropriate number of operands from the stack and pushes back the result.

More here.

10.11.2 Algebraic Notation (infix)

If you are not used to RPN notation, you may prefer to operate the Calculator in Algebraic mode, which is closer to the way non-RPN calculators work. In Algebraic mode, you enter formulas in traditional '2+3' notation.

NOTE: Calc gives '/' lower precedence than '*', so that 'a/b*c' is interpreted as 'a/(b*c)'; this is not standard across all computer languages. See below for details.

You don't really need any special "mode" to enter algebraic formulas. You can enter a formula at any time by pressing the apostrophe (') key. Answer the prompt with the desired formula, then press RET. The formula is evaluated and the result is pushed onto the RPN stack. If you don't want to think in RPN at all, you can enter your whole computation as a formula, read the result from the stack, then press DEL to delete it from the stack.

Try pressing the apostrophe key, then 2+3+4, then RET. The result should be the number 9.

More here.

10.11.3 Undo

If you make a mistake, you can usually correct it by pressing `U`, the "undo" command. First, clear the stack (`M-0 DEL`) and exit and restart Calc to make sure things start off with a clean slate.

11 DONE Final/Cleanup [0/0]

Display start-up time and play a notification message.

```
(defun display-startup-echo-area-message ()
  "Redefine the startup message to be more informative."
  (interactive)
  (tea-time-play-sound)
  (message (concat
            "Emacs initialization complete after "
            (message "'%s'" (propertize "%.2f" 'face '(:foreground "#66D9EF")))
            " seconds.")
           (- (float-time) emacs-start-time)))
```

Remove any buffers spawned during start-up

```
(volatile-kill-buffers)

(let ((inhibit-message nil))
    (message "Finished loading base Emacs config."))
```