Emacs Initialization Script

Andrew Apollonsky

December 28, 2014

Contents

1 Basic Settings

1.1 Identification

```
(setf user-full-name "Andrew Apollonsky")
(setf user-mail-address "Anapollonsky@gmail.com")
```

1.2 Diminish

Allows for hiding of modes from the mode line.

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

1.3 Key Chord

Allows for more complicated key combinations for commands.

```
(use-package key-chord
  :ensure t
  :diminish key-chord-mode
  :config
  (key-chord-mode t))
```

1.4 Theme

Add my theme directory and load one!

```
(add-to-list 'custom-theme-load-path "~/.emacs.d/themes")
(load-theme 'apples-gray t)
```

1.5 Backups

Make lots of backups, and store them in a centralized directory.

```
(setq backup-directory-alist '(("." . "~/.emacs.d/backups")))
(setq delete-old-versions -1)
(setq version-control t)
(setq vc-make-backup-files t)
(setq auto-save-file-name-transforms '((".*" "~/.emacs.d/auto-save-list/" t)))
```

1.6 History

Save history files!

```
(setq savehist-file "~/.emacs.d/savehist")
(savehist-mode 1)
(setq history-length t)
(setq history-delete-duplicates t)
(setq savehist-save-minibuffer-history 1)
(setq savehist-additional-variables
      '(kill-ring
        search-ring
        regexp-search-ring))
```

And save your last place in files when editing!

```
(use-package saveplace
  :ensure t
  :config
  (progn (setq save-place-file "~/.emacs.d/saveplace")
         (setq-default save-place t)))
```

1.7 Basic GUI

Turn off unnecessary GUI elements.

```
(when window-system
  (tooltip-mode -1)
  (tool-bar-mode -1)
  (menu-bar-mode -1)
  (scroll-bar-mode -1))
(setq inhibit-startup-screen t)
```

1.8 Encoding

UTF-8 is nice.

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

1.9 Writing

By default, sentences in emacs are expected to end with double spaces. Weird.

```
(setq sentence-end-double-space nil)
```

1.10 Other

Buffers that should appear over the current buffer.

```
(add-to-list 'same-window-buffer-names '*undo-tree*)
```

Integrate copy/paste with that of other X clients.

```
(setq x-select-enable-clipboard t
      interprogram-paste-function
      'x-cut-buffer-or-selection-value)
```

Replace yes-or-no with y-or-n

```
(setq search-highlight t
      query-replace-highlight t)
(fset 'yes-or-no-p 'y-or-n-p)
```

Case-insensitive name completion.

```
(setq completion-ignore-case t
      read-file-name-completion-ignore-case t)
```

These are fairly self-explanatory…

```
(global-unset-key (kbd "C-x c"))
(global-unset-key (kbd "C-c h"))
(global-set-key (kbd "RET")             'newline-and-indent)
(global-set-key (kbd "C-x C-r") 'comment-or-uncomment-region)
(global-set-key (kbd "<delete>")        'delete-char)
(global-set-key (kbd "M-g")             'goto-line)
(global-set-key (kbd "M-G")             'goto-char)
(global-set-key (kbd "C-x q")   'pop-to-mark-command) ;Pop to last mark
(global-set-key (kbd "C-c j")           'eval-region)
(key-chord-define-global "qw"           'other-window)
(key-chord-define-global "fw"           'fixup-whitespace)
```

1.11 Generic Custom Functions

Copies current file name to clipboard.

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

(global-set-key (kbd "C-c s h") 'prelude-copy-file-name-to-clipboard)
```

Show matching parentheses

```
(defadvice show-paren-function
  (after show-matching-paren-offscreen activate)
  "If the matching paren is offscreen, show the matching line in the
    echo area. Has no effect if the character before point is not of
    the syntax class ')'."
  (interactive)
  (let* ((cb (char-before (point)))
         (matching-text (and cb
                             (char-equal (char-syntax cb) ?\) )
                             (blink-matching-open))))
    (when matching-text (message matching-text))))
```

Jump to matching parentheses, forward or back.

```
(defun goto-match-paren (arg)
  "Go to the matching  if on (){}[], similar to vi style of % "
  (interactive "p")
  ;; first, check for "outside of bracket" positions expected by forward-sexp, etc.
  (cond ((looking-at "[\[\(\{]") (forward-sexp))
        ((looking-back "[\]\)\}]" 1) (backward-sexp))
        ;; now, try to succeed from inside of a bracket
        ((looking-at "[\]\)\}]") (forward-char) (backward-sexp))
        ((looking-back "[\[\(\{]" 1) (backward-char) (forward-sexp))
        (t nil)))
```

When called interactively with no active region, kill a single line instead.

```
(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)))))
```

2 Interface

2.1 Helm

Helm is a powerful package framework allowing for rapid text-based narrowing of choices. Pretty much conflicts with ido.

```
(use-package helm
  :ensure t
  :diminish helm-mode
  :init
  (progn
    (require 'helm-config)

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

    (setq helm-quick-update                      t ; do not display invisible candidates
          helm-split-window-in-side-p            t ; open helm buffer inside current window, not occupy whole other window
          helm-buffers-fuzzy-matching            t ; fuzzy matching buffer names when non--nil
          helm-move-to-line-cycle-in-source      t ; move to end or beginning of source when reaching top or bottom of source.
          helm-ff-search-library-in-sexp         t ; search for library in `require' and `declare-function' sexp.
          helm-scroll-amount                     8 ; scroll 8 lines other window using M-<next>/M-<prior>
          helm-ff-file-name-history-use-recentf t
          helm-ff-skip-boring-files t)
    (helm-mode))

  :config
  (progn
    (global-set-key (kbd "M-x")         'helm-M-x)
    (global-set-key (kbd "M-y")         'helm-show-kill-ring)
    (global-set-key (kbd "C-x b")       'helm-mini)
    (global-set-key (kbd "C-x C-f")     'helm-find-files)
    (global-set-key (kbd "C-c m")       'helm-man-woman)
    (global-set-key (kbd "C-c f")       'helm-find)
    (global-set-key (kbd "C-c u")       'helm-locate)
    (global-set-key (kbd "C-c o")       'helm-occur)))
```

Helm-swoop allows for fast navigation of lines in a document

```
(use-package helm-swoop
 :ensure t
 :config
 (progn
   (define-key isearch-mode-map (kbd "M-i")     'helm-swoop-from-isearch)
   (define-key helm-swoop-map (kbd "M-i")       'helm-multi-swoop-all-from-helm-swoop))
   ;; (global-set-key (kbd "C-c y q")            'helm-swoop))

)
```

2.2 Winner-mode

Go back and forth in buffer and window configuration history. A bit messy with helm, but worth it.

```
(when (fboundp 'winner-mode)
  (winner-mode 1))
(global-set-key (kbd "C-c C-<left>")    'winner-undo)
(global-set-key (kbd "C-c C-<right>")   'winner-redo)
```

2.3 Perspective

Control workspaces within emacs. Allows for multiple concurrent windows, switching between them, etc.

```
(use-package perspective
  :ensure t
  :config
  (progn
    (persp-mode 1)
    (key-chord-define-global "xx"               'persp-switch)))

(use-package persp-projectile
  :ensure t)
```

2.4 Windmove/Framemove

Allows for SHIFT + arrow keys to navigate between open buffers. Framemove extends this functionality to frames. In org-mode, this only works when shift + arrows are not over headers.

```
(use-package windmove
  :ensure t
  :config
  (progn
    (windmove-default-keybindings)
    (add-hook 'org-shiftup-final-hook 'windmove-up)
    (add-hook 'org-shiftleft-final-hook 'windmove-left)
    (add-hook 'org-shiftdown-final-hook 'windmove-down)
    (add-hook 'org-shiftright-final-hook 'windmove-right)))

(use-package framemove
  :ensure t
  :config
  (setq framemove-hook-into-windmove t))
```

2.5 Buffer-move

Easily swap the contents of two nearby buffer windows.

```
(use-package buffer-move
  :ensure t
  :config
  (progn (global-set-key (kbd "C-x <up>")       'buf-move-up)
         (global-set-key (kbd "C-x <down>")     'buf-move-down)
         (global-set-key (kbd "C-x <left>")     'buf-move-left)
         (global-set-key (kbd "C-x <right>")    'buf-move-right)))
```

2.6 God Mode

When activated, all commands treated as C- commands.

```
(use-package god-mode
  :ensure t
  :config
  (key-chord-define-global "qq"         'god-local-mode))
```

2.7 Projectile

Used for navigating and otherwise controlling projects.

```
(use-package projectile
  :ensure t
  :config
  (progn
    (projectile-global-mode)
    (setq projectile-completion-system 'helm)
    (setq projectile-enable-caching t)))

(use-package helm-projectile
  :ensure t
  :config
  (progn
    (helm-projectile-on)
    (global-set-key (kbd "C-c e")       'helm-projectile)))
```

2.8 Mode Line

Makes my mode line pretty.

```
(use-package powerline
  :ensure t
  :config
  (powerline-default-theme))
```

2.9 Guide-Key

Pops up with keyboard shortcut suggestions

```
(use-package guide-key
  :ensure t
  :diminish guide-key-mode
  :init
  (progn
  (setq guide-key/guide-key-sequence '("C-x" "C-c s" "C-c"))
  (guide-key-mode 1)))
```

3 Visual

3.1 Volatile Highlights

Highlight a recently-pasted region.

```
(use-package volatile-highlights
  :ensure t
  :diminish volatile-highlights-mode
  :config
  (volatile-highlights-mode t))
```

3.2 Whitespace

Configure whitespace display.

```
(use-package whitespace
  :ensure t
  :diminish global-whitespace-mode
  :init
  (progn (setq whitespace-style
               '(face tabs spaces newline space-mark tab-mark newline-mark indentation space-after-tab space-before-tab))
         (setq whitespace-display-mappings
               '(
                 (space-mark 32 [183] [46]) ; normal space
                 (newline-mark 10 [182 10]) ; newlne
                 (tab-mark 9 [9655 9] [92 9]) ; tab
                 )))
  :config (global-whitespace-mode))
```

3.3 Hi-Lock

Turn on hi-lock, allowing highlighting of symbols.

```
(global-hi-lock-mode 1)
```

3.4 Uniquify

```
(require 'uniquify)
(setq
 uniquify-buffer-name-style 'post-forward
 uniquify-separator ":"
 uniquify-after-kill-buffer-p t
 uniquify-ignore-buffers-re "^\\*")
```

3.5 Rainbow Delimiters

Color different matching sets of parentheses in different colors.

```
(use-package rainbow-delimiters
  :ensure t
  :init
  (add-hook 'prog-mode-hook 'rainbow-delimiters-mode))
```

3.6 Line Numbers

Only show line numbers in code-related major modes.

```
(use-package nlinum
  :ensure t
  :init
  (progn
    (add-hook 'haskell-mode-hook 'nlinum-mode)
    (add-hook 'emacs-lisp-mode-hook 'nlinum-mode)
    (add-hook 'c-mode-hook 'nlinum-mode)
    (add-hook 'c++-mode-hook 'nlinum-mode)
    (add-hook 'python-mode-hook 'nlinum-mode)))
```

3.7 Visual Line Mode

Causes line-wrapping, and remaps C-a and C-e to jump to visual lines, not logical lines.

```
(global-visual-line-mode t)
(diminish 'visual-line-mode)
```

4 Tool Major Modes

4.1 Ztree

Tools for navigating and comparing directories in tree-form.

```
(use-package ztree-dir
  :ensure ztree)

(use-package ztree-diff
  :ensure ztree)
```

4.2 Org-Mode

A lot of important stuff here.

```
  (use-package org
    :config
    (progn
      (unless (boundp 'org-export-latex-classes)
        (setq org-export-latex-classes nil))
      (setq org-log-done t)
      (setq org-src-fontify-natively t)

      ;; active Babel languages
      (org-babel-do-load-languages
       'org-babel-load-languages
       '((C . t)
         (python . t)
         (lisp . t)
         (latex . t)
         (sh . t)
         ))
      (key-chord-define-global "zq"             'org-capture)))

  ;; (setq org-export-html-style-include-scripts nil
  ;;        org-export-html-style-include-default nil)
  ;;  (setq org-export-html-style
  ;;        "<link rel=\"stylesheet\" type=\"text/css\" href=\"/home/aapollon/.emacs.d/themes/solarized-dark.css\" />")

  ;; Include the latex-exporter
  (use-package ox-latex
    :config
    (progn
      ;; Add minted to the defaults packages to include when exporting.
      (add-to-list 'org-latex-packages-alist '("" "minted"))
      ;; Tell the latex export to use the minted package for source
      ;; code coloration.
      (setq org-latex-listings 'minted)
      ;; Let the exporter use the -shell-escape option to let latex
      ;; execute external programs.
      (setq org-latex-pdf-process
            '("pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"))

      (setq org-export-latex-listings t)
      (add-to-list 'org-latex-classes
                   '("code-article"
                     "\\documentclass{article}"
                     ("\\section{%s}" . "\\section*{%s}")
                     ("\\subsection{%s}" . "\\subsection*{%s}")
                     ("\\subsubsection{%s}" . "\\subsubsection*{%s}")))


      ;; org-capture
      (setq org-directory "~/org")
      (setq org-default-notes-file (concat org-directory "/notes.org"))
      ;; Bind Org Capture to C-c r

      (setq org-capture-templates
            '(("t" "Todo" entry (file+headline (concat org-directory "/notes.org") "Tasks")
               "** TODO %?\n %i\n")
              ("l" "Link" plain (file+headline (concat org-directory "/notes.org") "Links")
               "- %?\n %x\n")
              ("q" "Quick Note" plain (file+headline (concat org-directory "/notes.org") "Quick Notes")
               "+ %?\n %i\n")))

      (setq org-agenda-files (list "~/org/agenda.org"))))

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

4.3 Speedbar

IDE-like tab can display directory or semantic code information.

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

(use-package semantic/sb)
```

4.4 Multi-term

Allows for multiple separate terminals to be open.

```
(use-package multi-term
  :ensure t
  :init
  (progn
    (setq multi-term-program "/bin/zsh")
    (setq term-buffer-maximum-size 0)))
```

4.5 ibuffer

Major mode for manipulating buffers. Create custom categories, save them.

```
(use-package ibuffer
  :ensure t
  :config
  (progn
    (setq ibuffer-saved-filter-groups
          (quote (("default"
                   ("Org"
                    (mode . org-mode))
                   ("Mail"
                    (or
                     (mode . message-mode)
                     (mode . mail-mode)
                     ))
                   ("Helm"
                    (name . "Helm"))
                   ("Vobs"
                    (filename . "/vobs/"))
                   ("Scripts"
                    (filename . "/home/aapollon/scripts"))
                   ("Manpages"
                    (mode . Man))))))

    (add-hook 'ibuffer-mode-hook
              (lambda ()
                (ibuffer-switch-to-saved-filter-groups "default")))
    (define-key global-map (kbd "C-x C-b")                  'ibuffer)))
```

4.6 IRC

IRC major-mode.

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

Mail client. Doesn't qyute work yet.

4.7 mu4e

```
(use-package mu4e
  :ensure mu4e-maildirs-extension
  :init
  (progn
    ;; default
    (setq mu4e-maildir (expand-file-name "~/.mail/gmail"))

    (setq mu4e-drafts-folder "Drafts")
    (setq mu4e-sent-folder   "Sent_Mail")
    (setq mu4e-trash-folder  "Trash")

    ;; don't save message to Sent Messages, Gmail/IMAP takes care of this
    (setq mu4e-sent-messages-behavior 'delete)

    ;; (See the documentation for `mu4e-sent-messages-behavior' if you have
    ;; additional non-Gmail addresses and want assign them different
    ;; behavior.)

    ;; setup some handy shortcuts
    ;; you can quickly switch to your Inbox -- press ``ji''
    ;; then, when you want archive some messages, move them to
    ;; the 'All Mail' folder by pressing ``ma''.
    ;; (setq mu4e-maildir-shortcuts
    ;;     '( ("/INBOX"              . ?i)
    ;;        ("/[Gmail].Sent Mail"   . ?s)
    ;;        ("/[Gmail].Trash"       . ?t)
    ;;        ("/[Gmail].All Mail"    . ?a)))

    ;; allow for updating mail using 'U' in the main view:
    (setq mu4e-get-mail-command "mbsync gmail"
          mu4e-html2text-command "w3m -T text/html"
          mu4e-update-interval 120
          mu4e-headers-auto-update t
          mu4e-compose-signature-auto-include nil)
    ;; something about ourselves
    (setq
     user-mail-address "Anapollonsky@gmail.com"
     user-full-name  "Andrew Apollonsky"
     mu4e-compose-signature
     (concat
      "Andrew Apollonsky"
      ""))))

;; sending mail -- replace USERNAME with your gmail username
;; also, make sure the gnutls command line utils are installed
;; package 'gnutls-bin' in Debian/Ubuntu
(use-package smtpmail
  :ensure t
  :init
  (progn
    ;; (setq message-send-mail-function 'smtpmail-send-it
    ;;    starttls-use-gnutls t
    ;;    smtpmail-starttls-credentials '(("smtp.gmail.com" 587 nil nil))
    ;;    smtpmail-auth-credentials
    ;;      '(("smtp.gmail.com" 587 "Anapollonsky@gmail.com" nil))
    ;;    smtpmail-default-smtp-server "smtp.gmail.com"
    ;;    smtpmail-smtp-server "smtp.gmail.com"
    ;;    smtpmail-smtp-service 587)

    ;; alternatively, for emacs-24 you can use:
    (setq message-send-mail-function 'smtpmail-send-it
          smtpmail-stream-type 'starttls
          smtpmail-default-smtp-server "smtp.gmail.com"
          smtpmail-smtp-server "smtp.gmail.com"
          smtpmail-smtp-service 587)

    ;; don't keep message buffers around
    (setq message-kill-buffer-on-exit t)))
```

4.8 Undo Tree

```
(use-package undo-tree
  :ensure t
  :diminish undo-tree-mode
  :config
  (progn
    (global-undo-tree-mode)
    (setq undo-tree-auto-save-history t)
    (setq undo-tree-visualizer-timestamps t)
    (setq undo-tree-visualizer-diff t)))
```

4.9 Dired

Dired! An emacs file manager.

```
(use-package dired+
  :ensure t
  :config
  (setq dired-dwim-target t)) ;; Copy to other dired buffer by default
```

5 Multi-Language Tools

5.1 Semantic

```
(use-package semantic
 :ensure t
 :config
 (progn
   (add-to-list 'semantic-default-submodes 'global-semanticdb-minor-mode)
   (add-to-list 'semantic-default-submodes 'global-semantic-idle-scheduler-mode)
   (add-to-list 'semantic-default-submodes 'global-semantic-idle-summary-mode)
   (add-to-list 'semantic-default-submodes 'global-semantic-decoration-mode)
   (add-to-list 'semantic-default-submodes 'global-semantic-highlight-func-mode)
   (add-to-list 'semantic-default-submodes 'global-semantic-stickyfunc-mode)
   (add-to-list 'semantic-default-submodes 'global-semantic-mru-bookmark-mode)
   (add-to-list 'semantic-default-submodes 'global-semantic-idle-breadcrumbs-mode)
   (semantic-mode 1)

   (defvar semantic-tags-location-ring (make-ring 20))
   (defun semantic-goto-definition (point)
     "Goto definition using semantic-ia-fast-jump, save the pointer marker if tag is found"
     (interactive "d")
     (condition-case err
         (progn
           (ring-insert semantic-tags-location-ring (point-marker))
           (semantic-ia-fast-jump point))
       (error
        ;;if not found remove the tag saved in the ring
        (set-marker (ring-remove semantic-tags-location-ring 0) nil nil)
        (signal (car err) (cdr err)))))

   (defun semantic-pop-tag-mark ()
     "popup the tag save by semantic-goto-definition"
     (interactive)
     (if (ring-empty-p semantic-tags-location-ring)
         (message "%s" "No more tags available")
       (let* ((marker (ring-remove semantic-tags-location-ring 0))
              (buff (marker-buffer marker))
              (pos (marker-position marker)))
         (if (not buff)
             (message "Buffer has been deleted")
           (switch-to-buffer buff)
           (goto-char pos))
         (set-marker marker nil nil))))

   (define-key semantic-mode-map (kbd "C-c w d")               'semantic-goto-definition)
   (define-key semantic-mode-map (kbd "C-c w q")               'semantic-pop-tag-mark)
   (define-key semantic-mode-map (kbd "C-c w e")               'senator-go-up-reference)
   (define-key semantic-mode-map (kbd "C-c w s")               'semantic-symref)
   (define-key semantic-mode-map (kbd "C-c w z")               'senator-previous-tag)
   (define-key semantic-mode-map (kbd "C-c w x")               'senator-next-tag)))

(use-package semantic/bovine/gcc
 :ensure semantic)
```

5.2 Xcscope

Cscope indexes and then allows for fast navigation of C projects, with limited support for other languages. This is my favorite such tool, because despite being dull and only working well with C, it has a nice interface.

```
(use-package xcscope
 :ensure t
 :init
 (progn
   (add-hook 'c-mode-hook 'cscope-minor-mode)
   (add-hook 'c++-mode-hook 'cscope-minor-mode)
   (setq cscope-initial-directory "/vobs/"))
 :config
 (define-key cscope-minor-mode-keymap (kbd "C-c s q")    'cscope-pop-mark))
```

5.3 Ggtags

Gtags, or GNU Global, is a more comprehensive tagging program, theoretically supporting more languages and whatnot. Ggtags is an emacs package for interfacing with it.

```
(use-package ggtags
 :ensure t
 :init
 (progn
   (defun create-tags (dir-name)
     "Create tags file."
     (interactive "DDirectory: ")
```

```
    (eshell-command
     (format "find %s -type f -name \"*.[ch]\" | etags -" dir-name))))
 :config
 (progn
   (define-key ggtags-mode-map (kbd "C-c s g")            'ggtags-find-tag-dwim)
   (define-key ggtags-mode-map (kbd "C-c s d")            'ggtags-find-definition)
   (define-key ggtags-mode-map (kbd "C-c s f")            'ggtags-find-file)
   (define-key ggtags-mode-map (kbd "C-c s s")            'ggtags-find-reference)
   (define-key ggtags-mode-map (kbd "C-c s q")            'ggtags-prev-mark)
   (define-key ggtags-mode-map (kbd "C-c s w")            'ggtags-next-mark)
   (define-key ggtags-mode-map (kbd "C-c s t")            'ggtags-grep)))
```

5.4 Helm-Gtags

Helm interface for gtags. I think I prefer ggtags.

```
(use-package helm-gtags
 :ensure t
 :init
 (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
  helm-gtatgs-path-style 'relative
  )
 :config
 (progn
   ;; helm-gtags
   (define-key helm-gtags-mode-map (kbd "C-c s v") 'helm-gtags-select)
   (define-key helm-gtags-mode-map (kbd "C-c s g") 'helm-gtags-dwim)
   (define-key helm-gtags-mode-map (kbd "C-c <")          'helm-gtags-previous-history)
   (define-key helm-gtags-mode-map (kbd "C-c >")          'helm-gtags-next-history)
   (define-key helm-gtags-mode-map (kbd "C-c s t") 'helm-gtags-find-tag)
   (define-key helm-gtags-mode-map (kbd "C-c s s") 'helm-gtags-find-symbol)
   (define-key helm-gtags-mode-map (kbd "C-c s f") 'helm-gtags-find-files)
   ;;(define-key helm-gtags-mode-map (kbd "C-c s q")       'helm-gtags-pop-stack)
   (define-key helm-gtags-mode-map (kbd "C-c s r") 'helm-gtags-resume)
   (define-key helm-gtags-mode-map (kbd "C-c s w") 'helm-gtags-next-history)
   (define-key helm-gtags-mode-map (kbd "C-c s q") 'helm-gtags-previous-history)))
```

5.5 Syntax Checking

Performs syntac checking and complains if errors occur.

```
(use-package flycheck-tip
  :ensure t)

(use-package flycheck
 :diminish company-mode
 :ensure t
 :init
 (progn
   (add-hook 'after-init-hook 'global-flycheck-mode)
   (flycheck-tip-use-timer 'verbose)))
```

5.6 Silver-Searcher

Interfaces to Ag, a non-indexing code searcher, like grep or awk but faster. Requires that Ag be installed.

```
(use-package helm-ag
 :ensure t
 :config
 (progn
   (setq helm-ag-insert-at-point 'symbol)
   (global-set-key (kbd "C-c y a") 'helm-ag)
   (global-set-key (kbd "C-c y q") 'helm-ag-pop-stack)))
```

5.7 Company Completion

Company Code completion framework. I dislike automatic completion, so it's instead bound to a key chord.

```
(use-package company
  :ensure t
  :diminish company-mode
  :config
  (progn
    (global-company-mode)
    (setq company-idle-delay nil)
    (key-chord-define-global "zx"               'company-complete)))
```

5.8 Aggressive Indent

Keeps your code continuously indented as you type. Visually distracting, but useful.

```
(use-package aggressive-indent
  :ensure t
  :config
  (global-set-key (kbd "C-c n")           'aggressive-indent-mode))
```

6 Language-Specific Tools

6.1 C

Set indentation size to 4 by default

```
(setq c-default-style "linux"
      c-basic-offset 4)
```

6.2 Haskell

Basic mode, with indentation and whatnot.

```
(use-package haskell-mode
  :ensure t
  :config
  (add-hook 'haskell-mode-hook 'turn-on-haskell-indentation))
```

Compiler integration

```
(use-package ghc
  :ensure t
  :config
  (progn
    (let ((my-cabal-path (expand-file-name "~/.cabal/bin")))
      (setenv "PATH" (concat my-cabal-path ":" (getenv "PATH")))
      (add-to-list 'exec-path my-cabal-path))

    (autoload 'ghc-init "ghc" nil t)
    (autoload 'ghc-debug "ghc" nil t)
    (add-hook 'haskell-mode-hook (lambda () (ghc-init)))))
```

6.3 Lisp

SLIME

```
(use-package slime-autoloads
  :ensure slime
  :config
  (setq inferior-lisp-program "/usr/local/bin/sbcl"))
```

Paredit allows for weird, parentheses-based editing. Don't quite have the hang of it yet.

```
(use-package paredit
  :ensure t
  :config
  (progn
    (add-hook 'slime-repl-mode-hook (lambda () (paredit-mode +1)))
    ;; Stop SLIME's REPL from grabbing DEL,
    ;; which is annoying when backspacing over a '('
    (defun override-slime-repl-bindings-with-paredit ()
      (define-key slime-repl-mode-map
        (read-kbd-macro paredit-backward-delete-key) nil))
    (add-hook 'slime-repl-mode-hook 'override-slime-repl-bindings-with-paredit)
    (add-hook 'emacs-lisp-mode-hook 'slime-mode)))
```

6.4 Python

```
(use-package jedi
  :ensure t
  :config
  (add-hook 'python-mode-hook 'jedi:setup))
```

6.5 Latex

Auctex is a latex minor mode. Preview allows latex snippets to be shown in-line.

```
(use-package tex
  :ensure auctex)
```

6.6 Generic Language Highlighting

```
(use-package generic-x)
```

7 Text Processing Tools

7.1 Visual-Regexp on Steroids

This package does two things: provides visual feedback to regexp-tools like replace, and replaces the built-in emacs regexp engine with another, the default being Python. This is nice, because the emacs regexp engine treats '[' as the raw character, and '\[' as the regexp grouping special character (and similarly with '('). This messes with your head once in a while.

```
(use-package visual-regexp-steroids
  :ensure t
  :config
  (progn
    (global-set-key (kbd "C-c r r")     'vr/replace)
    (global-set-key (kbd "C-c r q")     'vr/query-replace)
    (global-set-key (kbd "C-s")         'vr/isearch-forward)
    (global-set-key (kbd "C-r")         'vr/isearch-backward)))
```

7.2 Semantic Expand Region

Allows for the semantic expansion and contraction of the region.

```
(use-package expand-region
  :ensure t
  :config
  (progn
    (global-set-key (kbd "C-.")         'er/expand-region)
    (global-set-key (kbd "C-,")         'er/contract-region)))
```

7.3 Clean aident

Remove whitespace after RETing 2nd time in a row

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

7.4 yasnippet

Snippeting!

```
(use-package yasnippet
  :ensure t
  :diminish yas-minor-mode
  :init
  (yas-global-mode 1))
```

7.5 Abbreviations

Abbreviations!

```
(use-package abbrev
  :diminish abbrev-mode
  :config
  (progn
    (setq-default abbrev-mode t)
    (or (file-exists-p "~/.emacs.d/abbrev_defs") (write-region "" nil "~/.emacs.d/abbrev_defs"))
    (read-abbrev-file "~/.emacs.d/abbrev_defs")
    (setq save-address t)
    (setq save-abbrevs t)))
```

7.6 Hippie Expansion

Searches for similar phrases in history, open buffers, abbreviations, etc, and cycles through them.

```
(use-package hippie-exp
  :ensure t
  :config
  (key-chord-define-global "zx"         'hippie-expand))
```

7.7 Ace Jump

Allows for jumping around based on the first letter of characters, words and lines.

```
(use-package ace-jump-mode
  :ensure t
  :config
  (progn
    (eval-after-load "ace-jump-mode"
      '(ace-jump-mode-enable-mark-sync))
    (key-chord-define-global "jk"          'ace-jump-word-mode)
    (key-chord-define-global "jl"          'ace-jump-line-mode)
    (key-chord-define-global "jj"          'ace-jump-char-mode)))
```

7.8 Ace Zap

Zap up to a character, ace-jump style.

```
(use-package ace-jump-zap
  :ensure ace-jump-zap
  :config
  (progn
    (key-chord-define-global "jz"          'ace-jump-zap-up-to-char-dwim)))
```

7.9 Multiple Cursors

Powerful plugin that generates multiple cursors, allowing for easy rectangular editing, application of functions at several places at once, etc.

```
(use-package multiple-cursors
  :ensure t
  :config
  (progn
    (global-set-key (kbd "C-x c e") 'mc/edit-lines)
    (global-set-key (kbd "C-x c d") 'mc/mark-all-dwim)
    (global-set-key (kbd "C-x c a") 'mc/mark-all-like-this)
    (global-set-key (kbd "C-x c n") 'mc/mark-next-like-this)
    (global-set-key (kbd "C-x c p") 'mc/mark-previous-like-this)
    (global-set-key (kbd "C-x c w") 'mc/mark-more-like-this-extended)
    (global-set-key (kbd "C-x c t") 'mc/mark-sgml-tag-pair)
    (global-set-key (kbd "C-x c c") 'mc/insert-numbers)
    (global-set-key (kbd "C-x c r") 'mc/reverse-regions)
    (global-set-key (kbd "C-x c s") 'set-rectangular-region-anchor)))
```

7.10 Iedit

Fits a similar niche to multiple cursors, but slightly less powerful. Faster, though. Why not?

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

7.11 Version Control

Magit - the best interface for git. When it works.

```
(use-package magit
  :ensure t
  :diminish magit-auto-revert-mode
  :init
  (progn
    (autoload 'magit-status "magit" nil t)
    (define-key global-map (kbd "C-c g")                         'magit-status)))
```