config.org

Andy Pierz

August 3, 2016

1 Emacs Configuration

This is my personal Emacs configuration. I use the Railwaycat Emacs for Mac port. It has two main advantages over the Emacs for Mac OSX port for me: 1. it renders colors in my Powerline better, and 2. It lets me use 2 finger swiping to navigate through buffers, which is very handy when I am looking at Emacs while working on something else.

1.0.1 Screenshot of my config

`https://github.com/apierz/dotfiles/blob/master/screenshot.png?raw=true`

2 Personal Information

```
(setq user-full-name "Andy Pierz"
      user-mail-address "andy@andypierz.com")
```

3 Packages and Paths

The first thing that gets loaded is `package`, Emacs' package manager. I then add some common repositories for Emacs packages and initialize package.

```
(require 'package)
(add-to-list 'package-archives '("org" . "http://orgmode.org/elpa/"))
(add-to-list 'package-archives '("melpa" . "http://melpa.org/packages/"))
(add-to-list 'package-archives '("melpa-stable" .
"http://stable.melpa.org/packages/"))

(setq package-enable-at-startup nil)
(package-initialize)
```

Next I want to set my load paths, it's mostly the standard folders, plus one for my working Dracula Theme Directory.

```
(add-to-list 'load-path (expand-file-name "snippets" user-emacs-directory))
(add-to-list 'load-path "~/.emacs.d/plugins")
(add-to-list 'load-path "~/.emacs.d/plugins/evil-org-mode")
(add-to-list 'load-path "/usr/local/Cellar/mu/HEAD/bin/mu")
(add-to-list 'exec-path "/usr/local/bin")
(add-to-list 'load-path "/Users/Andy/Documents/Programming_Projects/dracula-theme/emacs")
```

`use-package` is a `require` replacement that helps Emacs load faster. Might be a placebo.

```
(require 'use-package)
```

4 Essential Settings

4.1 Simple Configuration Settings

These are basic Emacs settings. They include inhibiting the startup screen and menu-bar and various small visual preferences.

```
(setq inhibit-splash-screen t
      inhibit-startup-message t
      inhibit-startup-echo-area-message t) (menu-bar-mode -1)
(tool-bar-mode -1)
(when (boundp 'scroll-bar-mode)
  (scroll-bar-mode -1))
(show-paren-mode nil)
(setq visual-line-fringe-indicators '(left-curly-arrow right-curly-arrow))
(setq-default left-fringe-width nil)
(setq-default indent-tabs-mode nil)
(eval-after-load "vc" '(setq vc-handled-backends nil))
(setq vc-follow-symlinks t)
(setq large-file-warning-threshold nil)
(setq split-width-threshold nil)
(setq custom-safe-themes t)
(put 'narrow-to-region 'disabled nil)
(setq global-visual-line-mode t)
(setq word-wrap t)
(setq initial-major-mode 'org-mode)
(setq initial-scratch-message "")
(setq-default fill-column 80)
(setq-default tab-width 2)
(put 'dired-find-alternate-file 'disabled nil)
```

4.2 Title Bar Configuration

Emacs@hostname is not very useful information for the title bar.

```
(setq frame-title-format
  '("" invocation-name ": "(:eval (if (buffer-file-name)
                (abbreviate-file-name (buffer-file-name))
                  "%b"))))
```

4.3 Backup Configuration

I prefer for Emacs to store automatic backups in a central location, rather than leaving them strewn about.

```
(setq backup-directory-alist '(("." . "~/Dropbox/emacs_backups"))
      backup-by-copying      t  ; Don't de-link hard links
      version-control        t  ; Use version numbers on backups
      delete-old-versions    t  ; Automatically delete excess backups:
      kept-new-versions      5 ; how many of the newest versions to keep
      kept-old-versions      5) ; and how many of the old
```

5 Utility Functions

Small utility functions, mostly collected from others.

5.1 From aaronbieber

```
(defun cycle-powerline-separators (&optional reverse)
  "Set Powerline separators in turn.  If REVERSE is not nil, go backwards."
 (interactive)
 (let* ((fn (if reverse 'reverse 'identity))
   (separators (funcall fn '("arrow" "arrow-fade" "slant"
                            "chamfer" "wave" "brace" "roundstub" "zigzag"
                            "butt" "rounded" "contour" "curve")))
   (found nil))
  (while (not found)
    (progn (setq separators (append (cdr separators) (list (car separators))))
    (when (string= (car separators) powerline-default-separator)
      (progn (setq powerline-default-separator (cadr separators))
         (setq found t)
          (redraw-display)))))))
```

```
(defun rename-this-file-and-buffer (new-name)
  "Renames both current buffer and file it's visiting to NEW-NAME."
  (interactive "sNew name: ")
  (let ((name (buffer-name))
        (filename (buffer-file-name)))
    (unless filename
      (error "Buffer '%s' is not visiting a file!" name))
    (if (get-buffer new-name)
        (message "A buffer named '%s' already exists!" new-name)
      (progn
        (when (file-exists-p filename)
         (rename-file filename new-name 1))
        (rename-buffer new-name)
(set-visited-file-name new-name)))))

(defun delete-this-file ()
  "Delete the current file, and kill the buffer."
  (interactive)
  (or (buffer-file-name) (error "No file is currently being edited"))
  (when (yes-or-no-p (format "Really delete '%s'?"
                             (file-name-nondirectory buffer-file-name)))
    (delete-file (buffer-file-name))
(kill-this-buffer)))


(require 'htmlfontify)
(defun fontify-and-browse ()
  "Fontify the current buffer into HTML, write it to a temp file, and open it in a browser.'
  (interactive)
  (let* ((fontified-buffer (hfy-fontify-buffer))
         (temp-file-name (make-temp-file "ff" nil ".html")))
    (with-current-buffer fontified-buffer
      (write-region (point-min) (point-max) temp-file-name))
    (browse-url (concat "file://" temp-file-name))))


(defun show-first-occurrence ()
  "Display the location of the word at point's first occurrence in the buffer."
  (interactive)
  (save-excursion
    (let ((search-word (thing-at-point 'symbol t)))
      (goto-char 1)
      (re-search-forward search-word)
      (message (concat
                "L" (number-to-string (line-number-at-pos)) ": "
                (replace-regexp-in-string
                 "[ \t\n]*\\'"
                 ""
                 (thing-at-point 'line t)
                 ))))))

(defun switch-to-previous-buffer ()
  "Switch to previously open buffer.
```

```
Repeated invocations toggle between the two most recently open buffers."
  (interactive)
  (switch-to-buffer (other-buffer (current-buffer) 1)))

(defun narrow-and-set-normal ()
  "Narrow to the region and, if in a visual mode, set normal mode."
  (interactive)
  (narrow-to-region (region-beginning) (region-end))
  (if (string= evil-state "visual")
      (progn (evil-normal-state nil)
(evil-goto-first-line))))
```

5.2 From hrs

```
(defmacro diminish-minor-mode (filename mode &optional abbrev)
  "Supply a FILENAME, to hide a minor MODE or replace with an ABBREV."
  `(eval-after-load (symbol-name ,filename)
     '(diminish ,mode ,abbrev)))

(defmacro diminish-major-mode (mode-hook abbrev)
  "Supply a MODE-HOOK, to hide a major MODE or replace with an ABBREV."
  `(add-hook ,mode-hook
             (lambda () (setq mode-name ,abbrev))))
```

5.3 Mine + Unknown

```
(defun search-my-notes (searchforthis)
  "Search for SEARCHFORTHIS."
  (interactive "sSearch Query: ")
  (rgrep searchforthis "*.txt"  "~/Dropbox/Notes"))

(eval-after-load "grep"
  '(grep-compute-defaults))

(defun minibuffer-keyboard-quit ()
  "Abort recursive edit.
In Delete Selection mode, if the mark is active, just deactivate it;
then it takes a second \\[keyboard-quit] to abort the minibuffer."
  (interactive)
  (if (and delete-selection-mode transient-mark-mode mark-active)
      (setq deactivate-mark  t)
    (when (get-buffer "*Completions*") (delete-windows-on "*Completions*"))
    (abort-recursive-edit)))

(defun andy-new-empty-buffer ()
  "Open a new empty buffer."
  (interactive)
  (let ((buf (generate-new-buffer "untitled")))
    (switch-to-buffer buf)
    (funcall (and initial-major-mode))
    (setq buffer-offer-save t)))
```

6 Visual Stuff

I use my own version of Dracua Theme, which includes some extra coloring for Helm, mu4e, some changes to the syntax highlighting and correcting the background color when using Emacs in the terminal. My version is availble at my Github page. I also use Hack as my font. Right now I'm trying out transparency to see how I like it.

```
(use-package dracula-theme)
(load-theme 'dracula t)

(set-face-attribute 'default nil
                    :family "Hack" :height 140)

(set-frame-parameter (selected-frame) 'alpha '(90 90))
(add-to-list 'default-frame-alist '(alpha 90 90))
```

Just to be double-dog sure it ends up as utf-8...

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

I use some diminsh functions I got from hrs. This lets me hide some minor modes and rename others as encircled unicode characters. I also rename some major modes to save a little space in my powerline.

```
(diminish-minor-mode 'auto-complete 'auto-complete-mode "  ")
(diminish-minor-mode 'flycheck 'flycheck-mode "  ")
(diminish-minor-mode 'projectile 'projectile-mode "  ")
(diminish-minor-mode 'robe 'robe-mode "  ")
(diminish-minor-mode 'flymake 'flymake-mode "  ")
(diminish-minor-mode 'evil-snipe 'evil-snipe-local-mode)
(diminish-minor-mode 'evil-surround 'evil-surround-mode )
(diminish-minor-mode 'evil-commentary 'evil-commentary-mode)
(diminish-minor-mode 'yasnippet 'yas-minor-mode)
(diminish-minor-mode 'autorevert 'auto-revert-mode)
(diminish-minor-mode 'flyspell 'flyspell-mode)
```

```
(diminish-minor-mode 'undo-tree 'undo-tree-mode)
(diminish-minor-mode 'evil-org 'evil-org-mode)

(diminish-major-mode 'emacs-lisp-mode-hook ".el")
(diminish-major-mode 'haskell-mode-hook "?=")
(diminish-major-mode 'lisp-interaction-mode-hook "?")
(diminish-major-mode 'python-mode-hook ".py")
(diminish-major-mode 'ruby-mode-hook ".rb")
(diminish-major-mode 'sh-mode-hook ".sh")
(diminish-major-mode 'markdown-mode-hook ".md")
```

7 evil-mode

I prefer the Vim keybindings and use them wherever possible in Emacs. I recently made a switch to HJKL from IJKL when I learned the arrow keys on my `Pok3r` keyboard could be reprogrammed to use HJKL everywhere so I'm currently tring to unlearn my old bad habits.

7.1 Main package:

```
(use-package evil)
(evil-mode t)
```

7.2 Addons, based on Vim plugins

`evil-surround` is based on tpope's plugin and makes it easy to change surrounding syntax luke ", ', (, {, etc. `evil-commentary` is also based on a tpope plugin that makes it easy to comment a line or lines. [g-c-c] will comment a line [g-c-4-k] will comment the next 4 lines, etc.

```
(use-package evil-leader)
(use-package evil-surround
  :config
  (global-evil-surround-mode 1))
(use-package evil-commentary
  :config
  (evil-commentary-mode))
(use-package evil-snipe
  :config
  (evil-snipe-mode 1)
  (evil-snipe-override-mode 1))
```

7.3 Controls

I make a few changes to the `dired` control map to make it more natural when using Vim style navigation. Also I use 'hh' as a quick shortcut to return to evil-normal-state.

```
(use-package key-chord
  :config
  (key-chord-mode 1))

(key-chord-define evil-insert-state-map "hh" 'evil-normal-state)
(key-chord-define evil-insert-state-map ",," "<")
(key-chord-define evil-insert-state-map ".." ">")
(key-chord-define evil-replace-state-map "hh" 'evil-normal-state)
(key-chord-define evil-visual-state-map "hh" 'evil-normal-state)
(key-chord-define evil-motion-state-map "hh" 'evil-normal-state)
(evil-define-key 'normal dired-mode-map "l" 'dired-find-alternate-file)
(evil-define-key 'normal dired-mode-map "v" 'dired-toggle-marks)
(evil-define-key 'normal dired-mode-map "m" 'dired-mark)
(evil-define-key 'normal dired-mode-map "u" 'dired-unmark)
(evil-define-key 'normal dired-mode-map "U" 'dired-unmark-all-marks)
(evil-define-key 'normal dired-mode-map "c" 'dired-create-directory)
(evil-define-key 'normal dired-mode-map "n" 'evil-search-next)
(evil-define-key 'normal dired-mode-map "N" 'evil-search-previous)
(evil-define-key 'normal dired-mode-map "q" 'kill-this-buffer)
(setq evil-shift-width 2)
```

I made some changes to the normal evil-org keybindings because I think these bindings are more intuitive.

```
(use-package evil-org)
(evil-define-key 'normal evil-org-mode-map (kbd "M-k") 'org-metaup)
(evil-define-key 'normal evil-org-mode-map (kbd "M-h") 'org-metaleft)
(evil-define-key 'normal evil-org-mode-map (kbd "M-j") 'org-metadown)
(evil-define-key 'normal evil-org-mode-map (kbd "M-l") 'org-metaright)
(evil-define-key 'normal evil-org-mode-map (kbd "M-K") 'org-shiftmetaup)
(evil-define-key 'normal evil-org-mode-map (kbd "M-H") 'org-shiftmetaleft)
(evil-define-key 'normal evil-org-mode-map (kbd "M-J") 'org-shiftmetadown)
(evil-define-key 'normal evil-org-mode-map (kbd "M-L") 'org-shiftmetaright)
(evil-define-key 'normal evil-org-mode-map (kbd "K") 'org-shiftup)
(evil-define-key 'normal evil-org-mode-map (kbd "H") 'org-shiftleft)
(evil-define-key 'normal evil-org-mode-map (kbd "J") 'org-shiftdown)
(evil-define-key 'normal evil-org-mode-map (kbd "L") 'org-shiftright)
```

7.4 Evil Leader

Evil Leader is a package that let's you do quick shortcuts in evil-mode. While in evil-normal-state you press and hold your leader key (for me its ,) and then press another key to trigger a function. It's very handy and great for triggering little utility functions you come accross.

```
(defun andy--config-evil-leader ()
  "Configure evil leader mode."
  (evil-leader/set-leader ",")
  (setq evil-leader/in-all-states 1)
  (evil-leader/set-key
    "k"  'switch-to-previous-buffer
    "m"  'previous-buffer
    "."  'next-buffer
    ":"  'eval-expression
    "b"  'helm-mini
    "d"  'kill-this-buffer
    "e"  'find-file
    "f"  'fontify-and-browse
    "p"  'cycle-powerline-separators
    "b"  'switch-to-buffer
    "l"  'whitespace-mode       ;; Show invisible characters
    "nn" 'narrow-and-set-normal ;; Narrow to region and enter normal mode
    "nw" 'widen
    "o"  'delete-other-windows  ;; C-w o
    "S"  'delete-trailing-whitespace
    "t"  'gtags-reindex
    "T"  'gtags-find-tag
    "w"  'save-buffer
    "x"  'helm-M-x))

(global-evil-leader-mode)
(andy--config-evil-leader)
```

7.5 macOS Specific Stuff

Use the standard OSX keys for cut/copy/paste.

```
(defun pbcopy ()
  "Use OSX' pasteboard for copying."
  (interactive)
  (call-process-region (point) (mark) "pbcopy")
  (setq deactivate-mark t))

(defun pbpaste ()
  "Use OSX' pasteboard for pasting."
  (interactive)
  (call-process-region (point) (if mark-active (mark) (point)) "pbpaste" t t))

(defun pbcut ()
  "Use OSX' pasteboard for cutting."
  (interactive)
  (pbcopy)
  (delete-region (region-beginning) (region-end)))

(global-set-key (kbd "M-c") 'pbcopy)
(global-set-key (kbd "C-c x") 'pbcut)
(global-set-key (kbd "M-v") 'pbpaste)
```

Switch the macOS `Command` button to be Emacs `Meta` key.

```
(defun mac-switch-meta nil
  "Switch meta between Option and Command."
  (interactive)
  (if (eq mac-option-modifier nil)
      (progn
  (setq mac-option-modifier 'meta)
  (setq mac-command-modifier 'hyper)
)
    (progn
      (setq mac-option-modifier nil)
      (setq mac-command-modifier 'meta))))
```

7.6 Minor evil Configurations

Stop that terrible cursor move back nonsense!

```
(setq evil-move-cursor-back nil)
```

Set some shortcuts to the function buttons.

```
(global-set-key [f1]  'mu4e)
(global-set-key [f2] 'andy-new-empty-buffer)

(global-set-key [f4] 'fci-mode)
(global-set-key [f5] 'search-my-notes)
(global-set-key [f6] 'linum-relative-mode)
```

Robe Mode Commands

```
(global-set-key (kbd "M-j") 'robe-jump)
```

Magit Commands

```
(global-set-key (kbd "C-x g") 'magit-status)
(global-set-key (kbd "C-x M-g") 'magit-dispatch-popup)
```

Use ESC to quit non-evil stuff

```
(define-key evil-normal-state-map [escape] 'keyboard-quit)
(define-key evil-motion-state-map [escape] 'keyboard-quit)
(define-key evil-visual-state-map [escape] 'keyboard-quit)
(define-key evil-emacs-state-map [escape] 'keyboard-quit)
(define-key minibuffer-local-map [escape] 'minibuffer-keyboard-quit)
(define-key minibuffer-local-ns-map [escape] 'minibuffer-keyboard-quit)
(define-key minibuffer-local-completion-map [escape] 'minibuffer-keyboard-quit)
(define-key minibuffer-local-must-match-map [escape]'minibuffer-keyboard-quit)
(define-key minibuffer-local-isearch-map [escape] 'minibuffer-keyboard-quit)
```

Use `evil` controls in `Dired` and other `motion-state` modes.

```
(setq evil-normal-state-modes (append evil-motion-state-modes
  evil-normal-state-modes))
```

8 Helm

Helm is useful for searching through Emacs. I prefer Helm for searching through my buffers, kill ring and other things.

```
(use-package helm)
(use-package helm-config)
(global-set-key (kbd "C-x b") 'helm-buffers-list)
(global-set-key (kbd "C-x r b") 'helm-bookmarks)
(global-set-key (kbd "C-X m") 'helm-M-x)
(global-set-key (kbd "M-y") 'helm-show-kill-ring)
(global-set-key (kbd "C-x C-f") 'helm-find-files)

(setq helm-split-window-in-side-p t)

(with-eval-after-load
  'helm (define-key helm-map (kbd "<tab>") 'helm-execute-persistent-action)
     (define-key helm-map (kbd "ESC") 'helm-keyboard-quit)
)
```

9 Org-mode

`Org` is Emacs famous markup language with all kinds of useful features. You can even write your Emacs config in `Org`, which is what I have done here.

```
(use-package org)
(use-package ox)
(use-package org-grep)
(use-package org-capture)
```

These are the basic bindings `Org` recommends you use.

```
(global-set-key "\C-cl" 'org-store-link)
(global-set-key "\C-ca" 'org-agenda)
(global-set-key "\C-cc" 'org-capture)
(global-set-key "\C-cb" 'org-iswitchb)
```

I want everything in my notes folder to open in `Org-mode` and for .txt files to open in `Org-mode`. It is very rare I want to edit a plain text file without Org.

```
(setq org-export-coding-system 'utf-8)
(setq org-agenda-files (list "~/Dropbox/Notes"))
(setq org-agenda-file-regexp "\\`[^.].*\\.txt\\|[0-9]\\{8\\}\\'")
(add-to-list 'auto-mode-alist '("\\.txt$" . org-mode))
(setq org-agenda-text-search-extra-files (list nil ))


(add-hook 'find-file-hooks
  (lambda ()
    (let ((file (buffer-file-name)))
    (when (and file (equal (file-name-directory file) "~/Dropbox/Notes"))
    (org-mode)))))
```

I prefer to turn line numbers off while in `Org-mode`.

```
(use-package linum-off
  :config
  (add-to-list 'linum-disabled-modes-list "org-mode"))

(add-to-list 'org-latex-classes
             '("article"
               "\\documentclass{article}"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")
               ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))

(add-to-list org-latex-classes
   '(("my-article" "
\\documentclass[10pt,a4paper]{article}
\\usepackage[margin=2cm]{geometry}
\\usepackage{fontspec}

\\XeTeXlinebreakskip = 0pt plus 1pt
\\linespread{1.36}

\\setcounter{tocdepth}{5}
\\usepackage{multicol}

\\usepackage{hyperref}
\\hypersetup{
  colorlinks=true,
  linkcolor=[rgb]{0,0.37,0.53},
  citecolor=[rgb]{0,0.47,0.68},
  filecolor=[rgb]{0,0.37,0.53},
  urlcolor=[rgb]{0,0.37,0.53},
  pagebackref=true,
  linktoc=all,}"
      ("\\section{%s}" . "\\section*{%s}")
      ("\\subsection{%s}" . "\\subsection*{%s}")
      ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
      ("\\paragraph{%s}" . "\\paragraph*{%s}")
      ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))))
  (with-temp-buffer
(insert
 (mapconcat 'identity
        '("#+LATEX_CLASS: my-article"
          "#+OPTIONS: h:5 num:3"
          "* section"
          "** subsection"
          "*** subsubsection"
          "**** paragraph"
          "***** subparagraph")
        "\n"))
(org-latex-export-as-latex nil nil nil t)
```

9.1 Keywords

My todo system is fairly simple. `TODO` = unsorted, `ONDECK` = could be done at anytime, `WAITING` = waiting on something out of my control, `SOMEDAY` = not urgent, `CURRENT` = the thing I am currently working on. I've given these keywords colors from Dracula theme.

```
(setq org-todo-keywords
  '((sequence "TODO(t)" "ONDECK(o)" "WAITING(w)" "SOMEDAY(s)" "CURRENT(c)" "|" "DONE(d)")))

 ;; For Dracula Theme
 (setq org-todo-keyword-faces
   '(("ONDECK" . (:foreground "#f1fa8c" :weight bold))
     ("WAITING" . (:foreground "#bd93f9" :weight bold))
     ("CANCELED" . (:foreground "#ff5555" :weight bold))
     ("CURRENT" . (:foreground "#50fa7b" :weight bold))
     ("DONE" . (:foreground "#ff5555" :weight bold))
```

```
    ("SOMEDAY" . (:foreground "#6272a4" :weight bold))))
```

9.2 Visual Styling

I prefer to use fancy bullets, rather than a row of *s. Though every now and then I like to go back to a simpler style with one font size and regular bullets.

```
(setq org-hide-leading-stars t)
(use-package org-bullets
  :ensure t
  :config
  (add-hook 'org-mode-hook (lambda () (org-bullets-mode 1))))
```

Dracula Theme has recently revamped their `Org-mode` colors so I have switched to their defaults. I use a single character ellipsis, though sometimes switch to an arrow or other *fancy* symbol.

```
(setq org-ellipsis " ...")
```

I like some whitespace between my headings.

```
(setq org-cycle-separator-lines 0)
```

I prefer my text to wrap.

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

9.3 Org Capture

`Org` allows for capturing, which allows you to create/edit Org files whereever you are in Emacs. I have three kinds of Org Captures:

- TODO: adds a todo item to my Inbox heading in my main todo.txt file
- New Note: creates a new note file and saves it to my notes folder
- Kill Ring Note: creates a new note with whatever is currently at the head of my kill ring to a new note. I am considering changing this to add to an ongoing file instead.

```
(defun capture-report-date-file (path)
  (let ((name (read-string "Name: ")))
    (expand-file-name (format "%s.txt" name) path)))

(setq org-capture-templates
  '(
    ("t" "TODO" entry (file+headline "~/Dropbox/Notes/todo.txt" "Inbox")
     "** TODO %^{prompt}\n%U\n")
    ("n" "New Note" entry (file (capture-report-date-file "~/Dropbox/Notes/"))
     "** %^{prompt}\n %a\n%U\n")
    ("k" "Kill Ring Note" entry (file (capture-report-date-file "~/Dropbox/Notes"))
     "** %c\n %? %a\n %U\n")))
```

9.4 Org-babel

Org-babel is a system that allows for source code blocks within an Org mode document. It is very nice for notes, or for literate progamming, like this config file.

```
(setq org-src-fontify-natively t)
(setq org-src-tab-acts-natively t)
(setq org-src-window-setup 'current-window)
(setq org-confirm-babel-evaluate nil)

(org-babel-do-load-languages
 'org-babel-load-languages
 '((emacs-lisp . t)
   (ruby . t)
   (dot . t)
   (gnuplot . t)))
```

9.5 Org-Toodledo

Something to work on, need to find a way to hide my password.

```
;; (push "<path-to-this-file>" load-path)
;; (require 'org-toodledo)
;; (setq org-toodledo-userid "<toodledo-userid>")      << *NOT* your email!
;; (setq org-toodledo-password "<toodled-password>")

;; ;; Useful key bindings for org-mode
;; (add-hook 'org-mode-hook
;;        (lambda ()
;;          (local-unset-key "\C-o")
;;          (local-set-key "\C-od" 'org-toodledo-mark-task-deleted)
;;          (local-set-key "\C-os" 'org-toodledo-sync)
;;          )
;;        )
;; (add-hook 'org-agenda-mode-hook
;;        (lambda ()
;;          (local-unset-key "\C-o")
;;          (local-set-key "\C-od" 'org-toodledo-agenda-mark-task-deleted)
;;          )
       ;; )
```

10 Programming Stuff

This section is for stuff that helps with programming and coding. (note to self, look into diff-hl)

10.1 General Stuff

I like yasnippet for snippets, but I generally rely on auto-complete to speed up my coding.

```
(use-package yasnippet
  :ensure t
  :defer t
  :config
  (yas-reload-all)
  (setq yas-snippet-dirs '("~/.emacs.d/snippets"
                           "~/.emacs.d/remote-snippets"))
  (setq tab-always-indent 'complete)
  (setq yas-prompt-functions '(yas-completing-prompt
                               yas-ido-prompt
                               yas-dropdown-prompt))
(define-key yas-minor-mode-map (kbd "<escape>") 'yas-exit-snippet))
(ac-config-default)
```

I use relative line numbers, which helps with the Vim bindings. I use a 0 offset, so if I want to delete to a line and it says its line 4 I can press `d-4-k`.

```
(require 'linum-relative)

(linum-mode)
(global-linum-mode)
(setq linum-format "%4d \u2502 ")
(set-face-attribute 'linum nil :slant 'normal)
(with-eval-after-load 'linum
(linum-relative-toggle))
(setq linum-relative-current-symbol "->")
(setq linum-relative-plusp-offset 0)
```

I use smooth scrolling, it might be a placebo.

```
(use-package smooth-scrolling
  :config
  (smooth-scrolling-mode 1))
```

A few other useful packages for coding.

```
(use-package fill-column-indicator)
(use-package unbound)
(use-package nnir)
(use-package dumb-jump
  :config
  (dumb-jump-mode))
```

10.2 Emacs-lisp

```
(add-hook 'emacs-lisp-mode-hook
          (lambda ()
            (rainbow-delimiters-mode)))
```

10.3 Python

```
(setq python-indent-offset 2)
```

10.4 Shell and bash scripting

```
(add-hook 'sh-mode-hook
          (lambda ()
            (rainbow-delimiters-mode)
            (setq sh-basic-offset 2
                  sh-indentation 2)))
```

10.5 Ruby

```
(add-hook 'ruby-mode-hook
  (lambda ()
    (setq ruby-insert-encoding-magic-comment nil)
      (yas-minor-mode)
      (robe-mode)
      (rainbow-delimiters-mode)
      (local-set-key "\r" 'newline-and-indent)
      (flymake-mode)
      (flymake-ruby-load)
      (define-key ruby-mode-map (kbd "C-c C-c") 'xmp)
      (define-key ruby-mode-map (kbd "C-c C-s") 'inf-ruby)
      (define-key ruby-mode-map (kbd "C-c C-r") 'ruby-send-region)
      (define-key ruby-mode-map (kbd "C-c C-z") 'ruby-switch-to-inf)
      (define-key ruby-mode-map (kbd "C-c C-l") 'ruby-load-file)
      (define-key ruby-mode-map (kbd "C-c C-b") 'ruby-send-block)
))
(add-to-list 'auto-mode-alist
  '("\\.\\(?:erb\\)\\'" . web-mode))

(add-to-list 'auto-mode-alist
  '("\\.\\(?:cap\\|gemspec\\|irbrc\\|gemrc\\|rake\\|rb\\|ru\\|thor\\)\\'" . ruby-mode))
(add-to-list 'auto-mode-alist
  '("\\(?:Brewfile\\|Capfile\\|Gemfile\\(?:\\.[a-zA-Z0-9._-]+\\)?\\|[rR]akefile\\)\\'" . rub
```

10.6 web-mode

`Web-mode` is an Emacs major mode that gives syntax highlighting for web source files with multiple languages like html with php or .erb files.

```
(use-package web-mode
  :ensure t
  :defer t
  :config
  (add-to-list 'auto-mode-alist '("\\.html$" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.erb$" . web-mode))
  (add-to-list 'auto-mode-alist '("\\.twig$" . web-mode))
  (rainbow-delimiters-mode)
  (setq web-mode-attr-indent-offset 2)
  (setq web-mode-code-indent-offset 2)
  (setq web-mode-css-indent-offset 2)
  (setq web-mode-indent-style 2)
  (setq web-mode-markup-indent-offset 2)
  (setq web-mode-sql-indent-offset 2))
```

11 Powerline

Powerline is a mode line replacement for Emacs, based on Vim powerline. I've spent far too much time tweaking my Powerline and it shows no sign of stopping.

You customize the look of your powerline by defining faces for when the powerline is on the active buffer, or it's inactive. I've taken my colors from Dracula Theme, so it matches the rest of my config. Emacs in the terminal is limited to 256 colors, almost all of them bright, so darker colors don't look good when using the terminal. `(display-graphic-p)` lets me check if I'm on a terminal or not and set colors that look better if so.

```
(setq display-time-format "%I:%M")
(setq display-time-mail-directory "~/.Maildir/Personal/INBOX/new")
(setq display-time-default-load-average nil)
(display-time-mode 1)

  (defgroup segments-group nil "My powerline line segments" :group 'segments)

(if (display-graphic-p)
  (defface my-pl-segment1-active
    '((t (:foreground "#f1fa8c" :background "#3a2e58")))
    "Powerline first segment active face.")
  (defface my-pl-segment1-active
    '((t (:foreground "#f1fa8c" :background "#5f00af")))
    "Powerline first segment active face."))
  (defface my-pl-segment1-inactive
   '((t (:foreground "#f8f8f2" :background "#545565")))
    "Powerline first segment inactive face.")
  (defface my-pl-segment2-active
    '((t (:foreground "#f8f8f2" :background "#bd93f9")))
    "Powerline second segment active face.")
  (defface my-pl-segment2-inactive
    '((t (:foreground "#f8f8f2" :background "#545565")))
    "Powerline second segment inactive face.")

(if (display-graphic-p)
  (defface my-pl-segment3-active
    '((t (:foreground "#bd93f9" :background "#3a2e58")))
    "Powerline third segment active face.")
  (defface my-pl-segment3-active
    '((t (:foreground "#bd93f9" :background "#5f00af")))
    "Powerline third segment active face."))

  (defface my-pl-segment3-inactive
    '((t (:foreground "#f8f8f2" :background "#545565")))
    "Powerline third segment inactive face.")
  (defface my-pl-segment4-active
    '((t (:foreground "#ffffff" :background "#ff79c6")))
    "Powerline hud segment active face.")
  (defface my-pl-segment4-inactive
    '((t (:foreground "#ffffff" :background "#f8f8f2")))
    "Powerline hud segment inactive face.")


(if (display-graphic-p)
  (defface my-pl-segment5-active
    '((t (:foreground "#ff79c6" :background "#3a2e58")))
    "Powerline buffersize segment active face.")
  (defface my-pl-segment5-active
    '((t (:foreground "#ff79c6" :background "#5f00af")))
    "Powerline buffersize segment active face."))

  (defface my-pl-segment5-inactive
    '((t (:foreground "#f8f8f2" :background "#545565")))
    "Powerline buffersize segment inactive face.")

(if (display-graphic-p)
  (defface my-pl-segment6-active
   '((t (:foreground "#f1fa8c" :background "#3a2e58" :weight bold)))
    "Powerline buffer-id  segment active face.")
  (defface my-pl-segment6-active
   '((t (:foreground "#f1fa8c" :background "#5f00af" :weight bold)))
    "Powerline buffer-id  segment active face."))

  (defface my-pl-segment6-inactive
   '((t (:foreground "#f8f8f2" :background "#545565" :weight bold)))
    "Powerline buffer-id  segment inactive face.")
```

Then I use them to define a theme in a function. It looks a little confusing at first but it becomes easy with a little experimentation. The powerline is broken into two halves, the left (lhs) and right (rhs) with a section in the middle that fills any empty space.

```
(defun andy--powerline-default-theme ()
  "Set up my custom Powerline with Evil indicators."
  (interactive)
  (setq-default mode-line-format
    '("%e"
      (:eval
       (let* ((active (powerline-selected-window-active))
         (seg1 (if active 'my-pl-segment1-active 'my-pl-segment1-inactive))
         (seg2 (if active 'my-pl-segment2-active 'my-pl-segment2-inactive))
         (seg3 (if active 'my-pl-segment3-active 'my-pl-segment3-inactive))
         (seg4 (if active 'my-pl-segment4-active 'my-pl-segment4-inactive))
         (seg5 (if active 'my-pl-segment5-active 'my-pl-segment5-inactive))
         (seg6 (if active 'my-pl-segment6-active 'my-pl-segment6-inactive))
         (separator-left (intern (format "powerline-%s-%s"
                               (powerline-current-separator)
                               (car powerline-default-separator-dir))))
         (separator-right (intern (format "powerline-%s-%s"
                                (powerline-current-separator)
                                (cdr powerline-default-separator-dir))))
              (lhs (list (let ((evil-face (powerline-evil-face)))
                           (if evil-mode
                               (powerline-raw (powerline-evil-tag) evil-face)
                             ))
                         (if evil-mode
                             (funcall separator-left (powerline-evil-face) seg1))
                         (powerline-raw "[%*]" seg1 'l)

                         ;; (when powerline-display-buffer-size
                           ;; (powerline-buffer-size seg5 'l))
                         (powerline-vc seg5 'l)
                         (powerline-buffer-id seg6 'l)
                         (when (and (boundp 'which-func-mode) which-func-mode)
                           (powerline-raw which-func-format seg1 'l))
                         (powerline-raw " " seg1)
```

```
                                  (funcall separator-left seg1 seg2)
                                  (when (boundp 'erc-modified-channels-object)
                                    (powerline-raw erc-modified-channels-object seg2 'l))
                                  (powerline-major-mode seg2 'l)
                                  (powerline-process seg2)
                                  (powerline-narrow seg2 'l)
                                  (powerline-raw " " seg2)
                                  (funcall separator-left seg2 seg3)
                                  (powerline-minor-modes seg3 'l)
                                  ))
                                  (rhs (list
                                  (funcall separator-right seg3 seg2)
                                  (powerline-raw (char-to-string #xe0a1) seg2 'l)
                                  (powerline-raw "%l" seg2 'l)
                                  (powerline-raw ":" seg2 'r)
                                  (powerline-raw "%c" seg2 'r)
                                  (funcall separator-right seg2 seg1)
                                  (powerline-raw " " seg1)
                                  (powerline-raw "%6p" seg3 'r)
                                  (when powerline-display-hud
                                    (powerline-hud seg4 seg1))
                                  (powerline-raw " " seg1 'r)
                                  (funcall separator-right seg1 seg2)
                                  (powerline-raw global-mode-string seg2 'r)
)))
             (concat (powerline-render lhs)
                     (powerline-fill seg3 (powerline-width rhs))
                     (powerline-render rhs)))))))

    (use-package powerline
      :load-path "~/Documents/Programming_Projects/powerline"
      :ensure t
      :config
      (setq powerline-height 26)
      (setq powerline-default-separator (if (display-graphic-p) 'arrow-fade
                                          nil))
      (andy--powerline-default-theme))
```

I use `powerline-evil` to put a color changing evil state face on my powerline. If you are using Dracula theme, they will be Dracula colors. That change was my first accepted pull request to an open source project!

```
(use-package powerline-evil
  :ensure t)
```

12 Projectile

`projectile` is a helpful way to search through files in a project.

```
(use-package projectile)
(use-package helm-projectile)
```

I use a few basic settings and have `projectile` auto load whenever I'm in `ruby-mode`.

```
(add-hook 'ruby-mode-hook 'projectile-mode)
(add-hook 'web-mode-hook 'projectile-mode)
(setq projectile-indexing-method 'alien)
(setq projectile-switch-project-action 'projectile-find-file)
(setq projectile-completion-system 'default)
(setq projectile-enable-caching nil)

(helm-projectile-on)

(set-face-attribute 'helm-source-header nil :foreground "#ffb86c" :height 1.66)
```

13 mu4e

`mu4e` is an email client that works within Emacs. I use `mu4e-multi` to manage my work and personal accounts and `evil-mu4e` for some keybinding changes.

```
(use-package mu4e)
(require 'mu4e-multi)
(use-package evil-mu4e)
```

General Configuragtion

```
(setq mu4e-mu-binary "/usr/local/Cellar/mu/HEAD/bin/mu")
(setq mu4e-maildir "/Users/Andy/.Maildir")

(setq mu4e-multi-account-alist
  '(("personal"
     (user-mail-address .  "andy@andypierz.com")
     (user-full-name  .   "Andy Pierz")
     (mu4e-sent-folder .  "/personal/Sent\ Items")
     (mu4e-drafts-folder . "/personal/Drafts")
     (mu4e-trash-folder .  "/personal/Trash")
     (mu4e-refile-folder . "/personal/Archive"))
    ("work"
     (user-mail-address .  "andy@mutdut.com")
     (user-fullname . "Andy Pierz")
     (mu4e-sent-folder .  "/work/Sent\ Items")
     (mu4e-drafts-folder . "/work/Drafts")
     (mu4e-trash-folder .  "/work/Trash")
     (mu4e-refile-folder . "/work/Archive"))))

(mu4e-multi-enable)

(setq mu4e-drafts-folder "/drafts")


;;set attachment downloads directory
(setq mu4e-attachment-dir  "~/Downloads")

;; 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
  ’( ("/personal/INBOX"              . ?i)
     ("/personal/Sent\ Items"   . ?s)
     ("/personal/Trash"       . ?t)
     ("/personal/Archive"    . ?a)
     ("/personal/Starred"    . ?p)
     ("/personal/Drafts"    . ?d)

     ("/work/INBOX"      . ?w)
     ("/work/Drafts"      . ?z)
     ("/work/Sent\ Items"       . ?f)
     ("/work/Archive"    . ?o)))


;; allow for updating mail using ’U’ in the main view:
(setq mu4e-get-mail-command "offlineimap")
(setq mu4e-update-interval 300)

;; something about ourselves
(setq
  user-mail-address "andy@andypierz.com"
  user-full-name  "Andy Pierz"
  mu4e-compose-signature
  (concat
    ""
    ""))


(require ’smtpmail)
```

```
(setq message-send-mail-function 'smtpmail-send-it
  smtpmail-stream-type 'ssl
  smtpmail-default-smtp-server "mail.hover.com"
  smtpmail-smtp-server "mail.hover.com"
  smtpmail-smtp-service 465)

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

(defvar my-mu4e-account-alist
  '(("personal"
  ;; about me
  (user-mail-address      "andy@andypierz.com")
  (user-full-name         "Andy Pierz")
  ;; smtp
  (smtpmail-stream-type ssl)
  (smtpmail-starttls-credentials '(("mail.hover.com" 587 nil nil)))
  (smtpmail-default-smtp-server "mail.hover.com")
  (smtpmail-smtp-server "mail.hover.com")
  (smtpmail-smtp-service 465))
  ("work"
  ;; about me
  (user-mail-address      "andy@mutdut.com")
  (user-full-name         "Andy Pierz")
  ;;(mu4e-compose-signature "0xAX")

  ;; smtp
  (smtpmail-stream-type ssl)
  (smtpmail-auth-credentials '(("mail.hover.com" 25 "andy@mutdut.com" nil)))
  (smtpmail-default-smtp-server "mail.hover.com")
  (smtpmail-smtp-service 465))))

(defun my-mu4e-set-account ()
  "Set the account for composing a message."
  (let* ((account
    (if mu4e-compose-parent-message
      (let ((maildir (mu4e-message-field mu4e-compose-parent-message :maildir)))
      (string-match "/\\(.*?\\)/" maildir)
      (match-string 1 maildir))
      (completing-read (format "Compose with account: (%s) "
        (mapconcat #'(lambda (var) (car var)) my-mu4e-account-alist "/"))
        (mapcar #'(lambda (var) (car var)) my-mu4e-account-alist)
          nil t nil nil (car my-mu4e-account-alist))))
        (account-vars (cdr (assoc account my-mu4e-account-alist))))
    (if account-vars
      (mapc #'(lambda (var)
       (set (car var) (cadr var)))
          account-vars)
    (error "No email account found"))))

(add-hook 'mu4e-compose-pre-hook 'my-mu4e-set-account)


(use-package evil-mu4e)

(define-key mu4e-headers-mode-map "p" 'mu4e-headers-mark-for-flag)

(add-hook 'mu4e-main-mode-hook 'evil-motion-state)
(add-hook 'mu4e-headers-mode-hook 'evil-motion-state)
```

Use Dired to add attachments to emails.

```
(require 'gnus-dired)
;; make the `gnus-dired-mail-buffers' function also work on
;; message-mode derived modes, such as mu4e-compose-mode
(defun gnus-dired-mail-buffers ()
  "Return a list of active message buffers."
  (let (buffers)
    (save-current-buffer
      (dolist (buffer (buffer-list t))
  (set-buffer buffer)
  (when (and (derived-mode-p 'message-mode)
    (null message-sent-message-via))
    (push (buffer-name buffer) buffers))))
    (nreverse buffers)))

(setq gnus-dired-mail-mode 'mu4e-user-agent)
(add-hook 'dired-mode-hook 'turn-on-gnus-dired-mode)
```

Show some images in email messages.

```
(setq mu4e-view-show-images t)
(setq mu4e-view-show-image-max-width 800)
(when (fboundp 'imagemagick-register-types)
  (imagemagick-register-types))
(setq mu4e-view-prefer-html nil)
```

Convert html emails to text.

```
(setq mu4e-html2text-command 'mu4e-shr2text)
```

14 Magit

Magit is Emacs' Git interface.

```
(use-package magit)
(use-package evil-magit)
```

This is to encrypt my password so I can use it when sending email. If anyone knows how to set this up so it doesn't trigger an error everytime I re eval my buffer please let me know:

```
(use-package epa-file
  :config
  (unless (memq epa-file-handler file-name-handler-alist)
  (epa-file-enable)))
```