1 | Preparation

1.1 | Packages

First, let's set the package archives and do some basic tweaks.

```
(require 'package)
(setq package-enable-at-startup nil) ;; Speed tip taken from Doom Emacs
(setq package-archives '(("ELPA" . "https://tromey.com/elpa/")
 ("gnu" . "https://elpa.gnu.org/packages/")
 ("melpa" . "https://melpa.org/packages/")))
;; General convieniences, somewhat questionable
(setq url-http-attempt-keepalives nil)
(setq package-check-signature nil)
```

We're going to use `straight.el` for package management.

```
(defvar bootstrap-version)
(let ((bootstrap-file
       (expand-file-name "straight/repos/straight.el/bootstrap.el" user-emacs-directory))
      (bootstrap-version 5))
  (unless (file-exists-p bootstrap-file)
    (with-current-buffer
(url-retrieve-synchronously
 "https://raw.githubusercontent.com/raxod502/straight.el/develop/install.el"
 'silent 'inhibit-cookies)
      (goto-char (point-max))
      (eval-print-last-sexp)))
  (load bootstrap-file nil 'nomessage))
```

This configuration also uses `use-package` for package management: it's a macro that'll make things much easier. All that's generally important to understand right now is that code after the `:init` keyword argument will be evaluated as the package is initialized, code after the `:config` keyword arg is lazy-loaded, the `:bind` and `:hook` keyword arguments are for specifying package-related keybinds and hooks respectively.

```
(straight-use-package 'use-package)
(eval-when-compile
  (require 'use-package))
;; Automatically install all packages with straight.el if not present.
(setq straight-use-package-by-default t)
;; Always lazy-load if doable. TODO Properly look into good defer setup
(setq use-package-always-defer t)
(use-package general)
(use-package projectile)
(use-package s)
```

Are we using native-comp? If so, let's actually native compile things. Also, the popup compilation warnings are really bothersome, so turn those off, too.

```
(setq package-native-compile t)
(setq comp-deferred-compilation t)
(setq native-comp-deferred-compilation-deny-list nil)
(setq warning-suppress-log-types '((comp)))
```

1.2 | Speed

Let's then use a variety of the tips for speeding up initialization time given by the creator of Doom Emacs.

```
;; Go back to normal GC behavior after init
(add-hook 'emacs-startup-hook
  (lambda ()
    (setq gc-cons-threshold 16777216 ; 16mb
  gc-cons-percentage 0.1)))

;; Don't do GC when the minibuffer is being used (lag during minibuffer usage is frustrating)
(defun doom-defer-garbage-collection-h ()
  "Disable garbage collection."
  (setq gc-cons-threshold most-positive-fixnum))

(defun doom-restore-garbage-collection-h ()
  "Restore garbage collection."
  (run-at-time
   1 nil (lambda () (setq gc-cons-threshold 16777216))))

(add-hook 'minibuffer-setup-hook #'doom-defer-garbage-collection-h)
(add-hook 'minibuffer-exit-hook #'doom-restore-garbage-collection-h)
;; GCMH (literally Garbage Collector Magic Hack) optimizes GC calls?
(use-package gcmh
  :init
  (setq gcmh-idle-delay 5)
  (setq gcmh-high-cons-threshold (* 16 1024 1024))
  (gcmh-mode))
```

1.3 | Undoing Defaults

Emacs has some default behaviors that are generally annoying. Let's disable them!

```
;; Turn off all unnecessary GUI elements.
(tool-bar-mode -1)
(menu-bar-mode -1)
(scroll-bar-mode -1)

;; Unless something is actively exploding, I do not care.
(setq warning-minimum-level :emergency)

;; customize is the worst.
(setq custom-file "/dev/null")
(setq package-selected-packages "/dev/null/")

;; These keybinds suspend Emacs (in order to mimic terminal behavior).
;; This has *only* caused me trouble in GUI Emacs.
(when (display-graphic-p)
  (global-unset-key (kbd "C-z"))
  (global-unset-key (kbd "C-x C-z")))

;; Stop making backup files everywhere, put them all in one place!
(setq backup-directory-alist `(("." . "~/.saves")))
```

```
(setq backup-by-copying t)

;; Stop Emacs from bothering you about disabled commands.
(setq disabled-command-function nil)

;; Prevent any attempts to resize the frame.
(setq frame-inhibit-implied-resize t)

;; Stop Emacs from trying to use dialog boxes.
(setq use-dialog-box nil)

;; Prefer y/n over yes/no.
(fset 'yes-or-no-p 'y-or-n-p)

;; Mouse behavior tweaks? TODO look into me
(setq mouse-wheel-scroll-amount '(1 ((shift) . 1) ((control) . nil)))
(setq mouse-wheel-progressive-speed nil)

;; Visual line mode is just better.
(global-visual-line-mode)
```

2 | System

2.1 | System Packages

Let's now move on to system-level configuration. First, some utility functions for running commands and deducing distro/OS.

```
(defun process-exit-code-and-output (program &rest args)
  "Run PROGRAM with ARGS and return the exit code and output in a list."
  (with-temp-buffer
    (list (apply 'call-process program nil (current-buffer) nil args)
  (buffer-string))))

(defun get-distro-or-os ()
  "Return the Linux distribution or OS Emacs is running on."
  (if (eq system-type 'darwin)
      "Darwin"
    (when (eq system-type 'gnu/linux)
      (if (file-exists-p "/etc/os-release")
  (substring (shell-command-to-string "source /etc/os-release && echo $NAME") 0 -1)
(substring (car (cdr (process-exit-code-and-output "uname" "-o"))) 0 -1)))))
```

Then, let's set up system-packages, an awesome package that lets you programmatically install packages from Emacs across operating systems.

```
(use-package system-packages
  :init
  (let (os-name (get-distro-or-os))
    ;; system-packages doesn't support yay by default, so add it.
    (when (string= os-name "Arch Linux")
  (add-to-list 'system-packages-supported-package-managers
```

```
       '(yay .
     ((default-sudo . nil)
      (install . "yay -S")
      (uninstall . "yay -Rs")
      (update . "yay -Syu")
      (log . "cat /var/log/pacman.log")
      (change-log . "yay -Qc")
      (clean-cache . "yay -Sc")
      (get-info . "yay -Qi")
      (get-info-remote . "yay -Si")
      (list-files-provided-by . "yay -Ql")
      (owning-file . "yay -Qo")
      (verify-all-dependencies . "yay -Dk")
      (remove-orphaned . "yay -Rsn $(pacman -Qtdq)")
      (list-installed-packages . "yay -Qe")
      (list-installed-packages-all . "yay -Q")
      (noconfirm . "--noconfirm"))))
  (setq system-packages-package-manager 'yay))
    (when (string= os-name "Debian GNU/Linux")
  (setq system-packages-use-sudo t)
  (setq system-packages-package-manager 'apt))
    (if (string= os-name "Darwin")
(setq system-packages-package-manager 'brew)))
    (setq system-packages-noconfirm t))
```

This package also has some nice extensions like `use-package-ensure-system-package` which lets you express system-level dependencies for Emacs packages, and `helm-system-packages` which is the ultimate package manager interface (although it unfortunately means we'll need to install all of Helm for just this).

```
(use-package use-package-ensure-system-package)
(use-package helm-system-packages
  :commands (helm-system-packages))
```

2.2 | External Programs

`pywal` will be our savior for theming by allowing for thematic consistency.

```
;; (use-package exwm
;;   :ensure-system-package python-pywal)
```

`kitty` is a terminal emulator that's featureful and usable.

```
include ~/.cache/wal/colors-kitty.conf
font_family IBM Plex Mono
window_padding_width 10 15
map page_up scroll_page_up
map page_down scroll_page_down
map ctrl+shift+equal change_font_size all +2.0
map ctrl+shift+plus change_font_size all +2.0
map ctrl+shift+kp_add change_font_size all +2.0
initial_window_width 1000
initial_window_height 400
```

`zsh` is good.

```
# p10k instant prompt
if [[ -r "${XDG_CACHE_HOME:-$HOME/.cache}/p10k-instant-prompt-${(%):-%n}.zsh" ]]; then
  source "${XDG_CACHE_HOME:-$HOME/.cache}/p10k-instant-prompt-${(%):-%n}.zsh"
fi

export PATH=$PATH:$HOME/.local/bin/:$HOME/.cargo/bin/

export ZSH="$HOME/.oh-my-zsh"

ZSH_THEME="powerlevel10k/powerlevel10k"

plugins=(git)

source $ZSH/oh-my-zsh.sh

export EDITOR='emacs'

# Aliases
alias ydl="youtube-dl --extract-audio --audio-format mp3 -o '%(title)s.%(ext)s'"
alias neofetch="neofetch --ascii ~/.config/neofetch/arch.ascii"
alias gs="git status"
alias nano=mg
alias ls="exa --icons"
alias hexdump=hexyl
alias cat=bat
alias rm=rip
alias gcc="gcc -Wall -Werror -pedantic-errors"
alias g++="g++ -Wall -Weffc++ -Werror -pedantic-errors"

function recompile() {
    cd ~/.config/$1
    sudo make clean install &> /dev/null
    cd -
}

function fix_titles() {
    for a in *
    id3v2 -t ${a%.mp3} $a
}

function themeage() {
    wal -i $1 &> /dev/null
    xdotool key alt+r &> /dev/null
    emacsclient --eval "(load-theme 'ewal-doom-one)" &> /dev/null
    /home/quantumish/.local/bin/pywalfox update
    python ~/test.py colors-wal-dwm2.h
    python ~/test.py colors-wal-dmenu2.h
    python ~/test.py zathurarc
    python ~/test.py colors-vis
    recompile dmenu
}
```

```
# To customize prompt, run `p10k configure` or edit ~/.p10k.zsh.
[[ ! -f ~/.p10k.zsh ]] || source ~/.p10k.zsh
source  /usr/share/zsh/plugins/zsh-syntax-highlighting/zsh-syntax-highlighting.zsh
source /usr/share/zsh/plugins/zsh-autosuggestions/zsh-autosuggestions.zsh

export PATH=$PATH:$HOME/.local/bin/:$HOME/.cargo/bin/

export ZSH="$HOME/.oh-my-zsh"

ZSH_THEME="lambdamod"

plugins=(git zsh-autosuggestions zsh-syntax-highlighting)

source $ZSH/oh-my-zsh.sh

export EDITOR='emacs'

alias gs="git status"
alias nano=mg
alias gcc="gcc -Wall -Werror -pedantic-errors"
alias g++="g++ -Wall -Weffc++ -Werror -pedantic-errors"
```

It is clearly of top priority to ensure the Arch logo in `neofetch` looks good.

```
${c1}
```

Firefox could be prettier.

```
;; (use-package exwm
;;   :ensure-system-package (firefox python-pywalfox))

#TabsToolbar {visibility: collapse;}
#statuspanel[type="overLink"] #statuspanel-label {
    display:none!important;
}
```

2.3 | Desktop

It's time to load EXWM, the Emacs X Window Manager.

```
(use-package exwm
  :init
  (setq exwm-workspace-number 3)
  (setq exwm-input-global-keys
`(([?\s-r] . exwm-reset)
  ([?\s-w] . exwm-workspace-switch)
  ([?\s-&] . (lambda (command)
       (interactive (list (read-shell-command "$ ")))
       (start-process-shell-command command nil command)))))
  ;; Set default simulation keys
  (setq exwm-input-simulation-keys
'(([?\C-b] . [left])
  ([?\C-f] . [right])
  ([?\C-p] . [up])
  ([?\C-n] . [down])
  ([?\C-a] . [home])
  ([?\C-e] . [end])
  ([?\M-v] . [prior])
  ([?\C-v] . [next])
  ([?\C-d] . [delete])
  ([?\C-k] . [S-end delete])))
  ;; Allow windows to be moved across screens and interacted with normally.
  (setq exwm-layout-show-all-buffers t)
  (setq exwm-workspace-show-all-buffers t)
  (exwm-enable))
```

Setting up multi-monitor support is a bit of a hack in my configuration since my input devices tend to mysteriously swap around. You'll notice I'm using `use-package` for the same package twice in a row here, but fear not, it merely executes them sequentially and it means I can intersperse long-winded package configuration with text without fear of accidentally breaking something one day.

```
(use-package exwm
  :init
  (defvar left-screen "DP-1")
  (defvar middle-screen "HDMI-1")
  (defvar right-screen "DP-2")
  (require 'exwm-randr)
  (setq exwm-randr-workspace-output-plist `(0 ,middle-screen 1 ,left-screen 2 ,right-screen))
  (add-hook 'exwm-randr-screen-change-hook
    (lambda ()
      (start-process-shell-command
       "xrandr" nil (concat "xrandr --output " left-screen
    " --output " middle-screen
    " --output " right-screen
    " --auto"))))
  (exwm-randr-enable)
  (add-hook 'exwm-init-hook
    (lambda ()
      (start-process-shell-command
       "xrandr" nil (concat "xrandr --output " left-screen " --rotate left")))))
```

Next, if we're on Linux, let's do everything we need to do at startup.

`xmodmap` lets you modify the keys, so let's make things a lot nicer for Emacs.

```
clear      lock
clear   control
clear      mod1
clear      mod2
clear      mod3
clear      mod4
clear      mod5
keycode      37 = Hyper_L
keycode      66 = Control_L
keycode       9 = Escape
keycode  0xffca = Escape
add     control = Control_L Control_R
add        mod1 = Alt_L Alt_R Meta_L
add        mod2 = Num_Lock
add        mod3 = Hyper_L
add        mod4 = Super_L Super_R
add        mod5 = Mode_switch ISO_Level3_Shift
```

`xbindkeys` allows for customizing system-wide keybinds which can be useful when you're in a pickle. Most of this is legacy config from back before I started using EXWM.

```
# -*- shell-script -*-
# TODO Phase me out!

# Increase volume
"amixer set Master 5%+"
XF86AudioRaiseVolume

# Decrease volume
"amixer set Master 5%-"
XF86AudioLowerVolume

"amixer set Master toggle"
XF86AudioMute

"bash ~/.config/rofi/applets/menu/screenshot.sh"
Print

"bash ~/.config/rofi/applets/menu/powermenu.sh"
Pause

"bash ~/.config/rofi/applets/menu/apps.sh"
Scroll_Lock

"bash ~/.config/rofi/launchers/text/launcher.sh"
alt + p

"bash ~/.config/rofi/launchers/ribbon/launcher.sh"
alt + shift + p

"sh ~/.config/focus.sh"
```

```
alt + shift + f

"python ~/.config/modeset.py 'normal'"
m:0x20 + c:37 + F1

"rofi -show calc -modi calc -no-show-match -no-sort"
XF86Calculator
```

`xcape` allows for ”dual-function” keys that can act as one key when held down, and another when tapped. It’s niche but useful. We’ll remap tapping left-shift and right-shift to left and right parentheses respectively, as well as remap tapping caps-lock to escape.

```
xcape -e "Control_L=Escape"
xcape -e "Shift_R=parenright"
xcape -e "Shift_L=parenleft"
```

`dunst` is a great notification server.

```
[global]
monitor = 0
follow = keyboard
geometry = "320x20-36+36"
indicate_hidden = yes
shrink = yes
transparency = 0
notification_height = 0
separator_height = 0
padding = 8
horizontal_padding = 8
frame_width = 2
frame_color = "#000000"
separator_color = frame
sort = yes
idle_threshold = 120
font = IBM Plex Mono 10
line_height = 0
markup = full
format = "<b>%s</b>\n<i>%b</i>"
alignment = left
show_age_threshold = 60
word_wrap = yes
ellipsize = middle
ignore_newline = no
stack_duplicates = true
hide_duplicate_count = false
show_indicators = false
icon_position = left
max_icon_size = 32
icon_path = /usr/share/icons/candy-icons/apps/scalable:/usr/share/icons/candy-icons/devices/scalable/
sticky_history = yes
history_length = 20
dmenu = /usr/bin/dmenu -p dunst:
browser = /usr/bin/firefox -new-tab
always_run_script = true
```

```
title = Dunst
class = Dunst
startup_notification = false
verbosity = mesg
corner_radius = 0
force_xinerama = false
mouse_left_click = close_current
mouse_middle_click = do_action
mouse_right_click = close_all

[experimental]
per_monitor_dpi = false

[shortcuts]
close = ctrl+space
close_all = ctrl+shift+space
history = ctrl+grave
context = ctrl+shift+grave

[urgency_low]
foreground = "#ffd5cd"
background = "#121212"
frame_color = "#a2c5de"
timeout = 10
icon = ~/.config/dunst/images/notification.png

[urgency_normal]
background = "#121212"
foreground = "#ffd5cd"
frame_color = "#a2c5de"
timeout = 10
icon = ~/.config/dunst/images/notification.png

[urgency_critical]
background = "#121212"
foreground = "#ffd5cd"
frame_color = "#a2c5de"
timeout = 0
icon = ~/.config/dunst/images/alert.png
```

Let's define a quick script to reload it based on pywal, too.

```
. "${HOME}/.cache/wal/colors.sh"

pkill dunst
dunst \
      -frame_width 2 \
  -lb "${color0}" \
  -nb "${color0}" \
  -cb "${color0}" \
  -lf "${color7}" \
  -bf "${color7}" \
  -cf "${color7}" \
  -nf "${color7}" \
```

```
      -frame_color "${color2}" &
```

`picom` is a nice compositor, and will allow us to have effects like rounded corners and transparency if we want them. Dual kawase blur looks very nice, so let's use it.

```
backend = "glx";
blur: {
      method = "dual_kawase";
      strength = 10;
      background = false;
      background-frame = false;
      background-fixed = false;
}
```

Finally, we actually run the startup.

```
(use-package exwm
  ; :ensure-system-package (xbindkeys xcape dunst flameshot unclutter polybar feh picom)
  :init
  ;; Rebind keys
  (call-process-shell-command "xmodmap ~/.config/X/Xmodmap" nil 0)
  (call-process-shell-command "xbindkeys" nil 0)
  (call-process-shell-command "sh ~/.config/X/xcape.sh" nil 0)
  ;; Notifications w/ dunst
  (call-process-shell-command "dunst &" nil 0)
  (call-process-shell-command "sh ~/.config/dunst/reload_dunst.sh" nil 0)
  ;; Make mouse vanish when not used
  (call-process-shell-command "unclutter &" nil 0)
  ;; The best screenshot utility!
  (call-process-shell-command "flameshot &" nil 0)
  ;; Compositor
  (call-process-shell-command "picom &" nil 0)
  (call-process-shell-command "feh --bg-fill ~/.config/wallpapers/firewatch-galaxy.jpg" nil 0))
```

Let's make moving across monitors and workspaces a little easier.

```
(defun exwm-workspace-next ()
  (interactive)
  (if (< exwm-workspace-current-index (- exwm-workspace-number 1))
      (exwm-workspace-switch (+ exwm-workspace-current-index 1))))

(defun exwm-workspace-prev ()
  (interactive)
  (if (> exwm-workspace-current-index 0)
      (exwm-workspace-switch (- exwm-workspace-current-index 1))))

(general-define-key
 "M-h" 'exwm-workspace-next
 "M-l" 'exwm-workspace-prev)

;; Make mouse follow focus
(use-package exwm-mff
  :init (exwm-mff-mode))
```

```
(use-package exwmsw
  :straight (exwmsw :type git :host github :repo "Lemonbreezes/exwmsw"
    :fork (:host github :repo "richardfeynmanrocks/exwmsw"))
  :init
  (setq exwmsw-active-workspace-plist `(,middle-screen 0 ,right-screen 0 ,left-screen 0))
  (setq exwmsw-the-right-screen right-screen)
  (setq exwmsw-the-center-screen middle-screen)
  (setq exwmsw-the-left-screen left-screen)
  :general
  (override-global-map
    "C-M-j" #'exwmsw-cycle-screens
    "C-M-k" #'exwmsw-cycle-screens-backward)
  (exwm-mode-map ;; HACK
    "C-M-j" #'exwmsw-cycle-screens
    "C-M-k" #'exwmsw-cycle-screens-backward))
```

Then, make it so EXWM buffer names contain part of the the window title based off this great tip from r/emacs.

```
(use-package exwm
  :init

  (defun b3n-exwm-set-buffer-name ()
    (if (and exwm-title (string-match "\\`http[^ ]+" exwm-title))
(let ((url (match-string 0 exwm-title)))
  (setq-local buffer-file-name url)
  (setq-local exwm-title (replace-regexp-in-string
  (concat (regexp-quote url) " - ")
  ""
  exwm-title))))
    (setq-local exwm-title
(concat
 exwm-class-name
 "<"
 (if (<= (length exwm-title) 50)
     exwm-title
   (concat (substring exwm-title 0 50) "…"))
 ">"))

    (exwm-workspace-rename-buffer exwm-title))

  (add-hook 'exwm-update-class-hook 'b3n-exwm-set-buffer-name)
  (add-hook 'exwm-update-title-hook 'b3n-exwm-set-buffer-name))
```

Finally, update polybar config file to match monitor and make it so we have decorative gaps around all of EXWM (not individual buffers/windows unfortunately).

```
;; TODO Use Org Babel and tangle polybar config?
(start-process-shell-command "polybar-update" nil
    (concat "sed s/<MONITOR>/"
    middle-screen
    "/g -i ~/.config/polybar/config.ini.bak > ~/.config/polybar/config.ini"))

(use-package exwm-outer-gaps
```

```
  :straight (exmw-outer-gaps :type git :host github :repo "lucasgruss/exwm-outer-gaps")
  :hook (exwm-init . (lambda () (exwm-outer-gaps-mode))))

(use-package exwm
  :hook (exwm-init .
     (lambda () (call-process-shell-command "bash ~/.config/polybar/launch.sh --docky" nil 0))))
```

3 | Interface

3.1 | Theming

```
;; TODO: Set up treemacs.

(use-package hide-mode-line)

(use-package doom-themes
  :init
  ;; Global settings (defaults)
  (setq doom-themes-enable-bold t    ; if nil, bold is universally disabled
doom-themes-enable-italic t) ; if nil, italics is universally disabled

  (doom-themes-visual-bell-config)

  ;(setq doom-themes-treemacs-theme "doom-colors") ; use the colorful treemacs theme
  ;(doom-themes-treemacs-config)
  (doom-themes-org-config))

(use-package ewal)
(use-package ewal-doom-themes
  :init
  (load-theme 'ewal-doom-one t))

(use-package doom-modeline
  :init
  (setq doom-modeline-height 40)
  (setq doom-modeline-buffer-encoding nil)
  (doom-modeline-mode))

;; TODO: Contextual solaire
(use-package solaire-mode
  :init
  (solaire-global-mode))

(use-package centaur-tabs
  :init
  (setq centaur-tabs-height 16)
  (setq centaur-tabs-style "bar")
  (setq centaur-tabs-set-icons t)
  (setq centaur-tabs-icon-scale-factor 0.7)
  (setq centaur-tabs-set-bar 'left)
  (setq x-underline-at-descent-line t)
  (defun contextual-tabs ()
```

```
(interactive)
(if (and (centaur-tabs-mode-on-p) (eq (derived-mode-p 'prog-mode) nil))
(centaur-tabs-local-mode)))
  (defun centaur-tabs-hide-tab (x)
(let ((name (format "%s" x)))
  (or
   (window-dedicated-p (selected-window))
   (string-match-p (regexp-quote "<") name)
   (string-prefix-p "*lsp" name)
   (string-prefix-p "*Compile-Log*" name)
   (string-prefix-p "*company" name)
   (string-prefix-p "*compilation" name)
   (string-prefix-p "*Help" name)
   (string-prefix-p "*straight" name)
   (string-prefix-p "*Flycheck" name)
   (string-prefix-p "*tramp" name)
   (string-prefix-p "*help" name)
   (and (string-prefix-p "magit" name)
(not (file-name-extension name)))
   )))
  (defun centaur-tabs-hide-tab-cached (x) (centaur-tabs-hide-tab x))
  (centaur-tabs-mode)
  :hook
  (after-change-major-mode . contextual-tabs)
  :bind
  ("H-l" . 'centaur-tabs-forward-tab)
  ("H-h" . 'centaur-tabs-backward-tab))

(use-package treemacs
  :after doom-themes
  :init
  (doom-themes-treemacs-config)
  (setq doom-themes-treemacs-theme "doom-colors")
  (setq treemacs-width 30)
  :bind
  ("C-c t" . treemacs))

(use-package treemacs-all-the-icons
  :after treemacs
  :init
  (treemacs-load-theme "all-the-icons"))

(use-package olivetti
  :hook (prog-mode . (lambda () (olivetti-mode))))
```

3.1.1 | Translucent

Transparency can look nice - sometimes. Polybar clashes with transparency, so disable it while we're using it.

```
;; FIXME hacky and broken
(define-minor-mode translucent-mode
  "Make the current frame slightly transparent and don't use polybar."
```

```
  nil
  :global t
  (if translucent-mode
      (set-frame-parameter (selected-frame) 'alpha '(100))
    (set-frame-parameter (selected-frame) 'alpha '(90))))
```

3.2 | Dashboard

```
(use-package dashboard
  :straight (emacs-dashboard :type git :host github :repo "emacs-dashboard/emacs-dashboard"
    :fork (:host github :repo "richardfeynmanrocks/emacs-dashboard"))
  :init
  (setq dashboard-center-content t)
  (setq dashboard-set-heading-icons t)
  (setq dashboard-projects-backend 'projectile)
  (setq dashboard-footer-messages '("The One True Editor!"
    "Protocol 3: Protect the Pilot"
    "All systems nominal."
    "Democracy... is non negotiable."
    "It's my way or... hell, it's my way!"
    "Make life rue the day it though it could give Richard Stallman lemons!"
    "Vi-Vi-Vi, the editor of the beast."
    "Happy hacking!"
    "While any text editor can save your files, only Emacs can save your soul."
    "There's an Emacs package for that."
    "Rip and tear, until it is done!"
    "It's time to kick ass and chew bubblegum... and I'm all outta gum."
    "M-x butterfly"
    ""))
  (setq dashboard-items '((recents  . 3)
  (projects . 3)
  (agenda . 5)))
  (setq dashboard-startup-banner "~/Downloads/firewatch-logo.png")
  (setq dashboard-image-banner-max-height 250)
  (setq dashboard-image-banner-max-width 250)

  (setq dashboard-set-init-info nil)
  ;; (setq dashboard-set-navigator nil)
  ;; ;; Format: "(icon title help action face prefix suffix)"
  ;; (setq dashboard-navigator-buttons
  ;;  `(;; line1
  ;;         ((,(all-the-icons-octicon "mark-github" :height 1.1 :v-adjust 0.0)
  ;;           "Homepage"
  ;;           "Browse homepage"
  ;;           (lambda (&rest _) (browse-url "homepage")))
  ;;          (" " "Star" "Show stars" (lambda (&rest _) (show-stars)) warning)
  ;;          ("?" "" "?/h" #'show-help nil "<" ">"))
  ;;         ;; line 2
  ;;         ((,(all-the-icons-faicon "linkedin" :height 1.1 :v-adjust 0.0)
  ;;           "Linkedin"
  ;;           ""
  ;;           (lambda (&rest _) (browse-url "homepage")))
```

```
  ;;          (" " nil "Show flags" (lambda (&rest _) (message "flag")) error))))
  (setq dashboard-page-separator "\n\n")
  (dashboard-setup-startup-hook)
  :hook
  (dashboard-mode . hide-mode-line-mode)
  (dashboard-mode . turn-off-solaire-mode))
```

3.3 | Minibuffer Completion

Next, let's improve interactions with Emacs: things like finding files, running commands, switching buffers, etc… by using `ivy`, a light(ish) minibuffer completion system. Ivy is one of the more popular packages for this, meaning that there's quite a bit of integration with other packages. Notably, `counsel` extends its functionality and `swiper` provides a nicer interface to interactive search.

On top of this, `prescient` allows for completions to be even more useful by basing them off of history and sorting them better. Finally, we can add some icons and extra text to make it all prettier.

```
(use-package prescient
  :init (setq prescient-persist-mode t))

(use-package ivy
  :init
  (use-package counsel :config (counsel-mode 1))
  (use-package swiper :defer t)
  (ivy-mode 1)
  :bind
  (("C-s"     . swiper-isearch)
   ("M-x"     . counsel-M-x)
   ("C-x C-f" . counsel-find-file)))

(use-package ivy-rich
  :after ivy
  :init (ivy-rich-mode))

(use-package all-the-icons-ivy-rich
  :after ivy-rich counsel
  :init (all-the-icons-ivy-rich-mode))

(use-package ivy-prescient
  :after ivy prescient
  :init (ivy-prescient-mode))

(use-package marginalia
  :config (marginalia-mode))
```

3.4 | Help

In order to make some parts of exploring Emacs slightly nicer, let's install `helpful` which overhauls the Help interface, and `which-key` which helps you discover keybinds.

```
(use-package helpful
  :init
```

```
  ;; Advise describe-style functions so that Helpful appears no matter what
  (advice-add 'describe-function :override #'helpful-function)
  (advice-add 'describe-variable :override #'helpful-variable)
  (advice-add 'describe-command :override #'helpful-callable)
  (advice-add 'describe-key :override #'helpful-key)
  (advice-add 'describe-symbol :override #'helpful-symbol)
  :config
  ;; Baseline keybindings, not very opinionated
  (global-set-key (kbd "C-h f") #'helpful-callable)
  (global-set-key (kbd "C-h v") #'helpful-variable)
  (global-set-key (kbd "C-h k") #'helpful-key)
  (global-set-key (kbd "C-c C-d") #'helpful-at-point)
  (global-set-key (kbd "C-h F") #'helpful-function)
  (global-set-key (kbd "C-h C") #'helpful-command)
  ;; Counsel integration
  (setq counsel-describe-function-function #'helpful-callable)
  (setq counsel-describe-variable-function #'helpful-variable))

(use-package which-key
  :init (which-key-mode))
```

4 | Movement

```
(use-package zygospore
  :bind ("M-m" . 'zygospore-toggle-delete-other-windows))

(defun opposite-other-window ()
  "Cycle buffers in the opposite direction."
  (interactive)
  (other-window -1))

(defun opposite-other-frame ()
  "Cycle frames in the opposite direction."
  (interactive)
  (other-frame -1))

(general-def 'override-global-map
  "C-M-j" 'opposite-other-frame
  "C-M-k" 'other-frame)

(general-def 'override-global-map
  "M-k" 'other-window
  "M-j" 'opposite-other-window)

(general-def 'exwm-mode-map
  "M-k" 'other-window
  "M-j" 'opposite-other-window)
```

4.1 | Hydra

```
(use-package hydra
```

```
  :init
  (global-unset-key (kbd "C-x h"))
  (general-def
    "C-x h l" 'hydra-launcher/body
    "C-x h a" 'hydra-org-agenda/body
    "C-x h f" 'hydra-go-to-file/body))

(use-package pretty-hydra)
(use-package s)
(use-package major-mode-hydra
  :after hydra
  :preface
  (defun with-alltheicon (icon str &optional height v-adjust face)
    "Display an icon from all-the-icon."
    (s-concat (all-the-icons-alltheicon icon :v-adjust (or v-adjust 0) :height (or height 1) :face face)

  (defun with-faicon (icon str &optional height v-adjust face)
    "Display an icon from Font Awesome icon."
    (s-concat (all-the-icons-faicon icon ':v-adjust (or v-adjust 0) :height (or height 1) :face face) "

  (defun with-fileicon (icon str &optional height v-adjust face)
    "Display an icon from the Atom File Icons package."
    (s-concat (all-the-icons-fileicon icon :v-adjust (or v-adjust 0) :height (or height 1) :face face)

  (defun with-octicon (icon str &optional height v-adjust face)
    "Display an icon from the GitHub Octicons."
    (s-concat (all-the-icons-octicon icon :v-adjust (or v-adjust 0) :height (or height 1) :face face) "

(pretty-hydra-define hydra-flycheck
  (:hint nil :color teal :quit-key "q" :title (with-faicon "plane" "Flycheck" 1 -0.05))
  ("Checker"
   (("?" flycheck-describe-checker "describe")
    ("d" flycheck-disable-checker "disable")
    ("m" flycheck-mode "mode")
    ("s" flycheck-select-checker "select"))
   "Errors"
   (("<" flycheck-previous-error "previous" :color pink)
    (">" flycheck-next-error "next" :color pink)
    ("f" flycheck-buffer "check")
    ("l" flycheck-list-errors "list"))
   "Other"
   (("M" flycheck-manual "manual")
    ("v" flycheck-verify-setup "verify setup"))))

(pretty-hydra-define hydra-go-to-file
  (:hint nil :color teal :quit-key "q" :title (with-octicon "file-symlink-file" "Go To" 1 -0.05))
  ("Org"
   (("oi" (find-file "~/sync/org/inbox.org") "inbox")
    ("oc" (find-file "~/sync/org/completed.org") "home")
    ("op" (find-file "~/sync/org/projects.org") "projects"))
   "Config"
   (("cc" (find-file "~/.emacs.d/config.org") "config.org")
    ("ci" (find-file "~/.emacs.d/init.el") "init.el" ))
   "Notes"
```

```
   (("ni" (find-file "~/sync/notes/index.org") "Main Index"))
   ))


(pretty-hydra-define hydra-org-agenda
  (:hint nil :color teal :quit-key "q" :title (with-faicon "list-ol" "Agenda" 1 -0.05))
  ("Standard"
   (("w" (org-agenda)))))


(pretty-hydra-define hydra-launcher
  (:hint nil :color teal :quit-key "q" :title (with-faicon "arrow-right" "Launch" 1 -0.05))
  ("Shell-likes"
   (("v" vterm "Vterm")
    ("e" eshell "Eshell")
    ("l" ielm "IELM")
    ("k" (call-process-shell-command "open -a Kitty" nil 0) "Kitty"))
   "Messaging"
   (("i" erc "ERC")
    ("d" (call-process-shell-command "open -a Discord" nil 0) "Discord")
    ("t" (call-process-shell-command "open -a Telegram" nil 0) "Telegram"))
   "Misc"
   (("f" (call-process-shell-command "open -a Firefox" nil 0) "Firefox")
    ("s" (call-process-shell-command "open -a Spotify" nil 0) "Spotify")
    ("m" (call-process-shell-command "open -a Spark" nil 0) "Spark"))
   ))
```

4.2 | Perspectives

5 | TODO Org

First, let's set up the basics.

```
(use-package org
  :init
  (setq org-todo-keywords '((sequence "TODO(t)" "WAIT(w)" "|" "DONE(d)" "NOPE(n)")))
  (setq org-modules (append org-modules '(org-habit org-id)))  )
```

5.1 | Aesthetics

Let's add aesthetics for normal prose-style Org usage.

```
(use-package org
  :config
  (setq org-fontify-quote-and-verse-blocks t)
  (setq org-fontify-emphasized-text t)
  (setq org-hide-emphasis-markers t)
  (setq org-ellipsis " ")
  (setq org-hide-leading-stars t)
  (set-face-attribute 'org-document-title nil
      :height 2.0
```

```
      :weight 'bold)
  :hook (org-mode . org-indent-mode))
```

There are a variety of useful packages that make Org look nicer:

```
(setq org-latex-create-formula-image-program 'dvisvgm)
;; Smart mixing of variable pitch and monospace
;; This is preferred over `mixed-pitch` because of small details
(use-package org-variable-pitch
  :init (org-variable-pitch-setup))

;; Better headline icons
(use-package org-superstar
  :config
  (setq org-superstar-headline-bullets-list '(" " " " " " " "))
  :hook (org-mode . org-superstar-mode))

;; Auto-toggle emphasis
(use-package org-appear
  :straight (:host github :repo "awth13/org-appear")
  :hook (org-mode . org-appear-mode))

;; Auto-toggle LaTeX rendering
(use-package org-fragtog
  :hook (org-mode . org-fragtog-mode))

;; Natural bulleted lists
(use-package org-autolist
  :hook (org-mode . org-autolist-mode))

;; Centering w/ Olivetti
(use-package olivetti
  :hook (org-mode . (lambda () (interactive) (olivetti-mode) (olivetti-set-width 100))))
```

5.1.1 | Icons

```
(use-package org
  :config
  (defun magic-icon-fix ()
    (let ((fontset (face-attribute 'default :fontset)))
      (set-fontset-font fontset '(?\xf000 . ?\xf2ff) "FontAwesome" nil 'append)))
  :hook
  (org-mode . (lambda () (interactive)
(setq prettify-symbols-alist '(("[#A]" . " ")
       ("[#B]" . " ")
       ("[#C]" . " ")
       ("[ ]" . " ")
       ("[X]" . " ")
       ("[-]" . " ")
       ("#+begin_src" . " ")
       ("#+end_src" . " ")
       ("#+begin_collapsible" . " ")
       ("#+end_collapsible" . " ")
```

```
       ("#+begin_aside" . " ")
       ("#+end_aside" . " ")
       ("#+begin_defn" .  " ")
       ("#+end_defn" . " ")
       ("#+begin_questionable" .  " ")
       ("#+end_questionable" . " ")
       ("#+begin_problem" .  " ")
       ("#+end_problem" . " ")
       ("#+STARTUP:" . " ")
       ("#+TITLE: " . "")
       ("#+title: " . "")
       ("#+RESULTS:" . " ")
       ("#+NAME:" . " ")
       ("#+ROAM_TAGS:" . " ")
       ("#+FILETAGS:" . " ")
       ("#+HTML_HEAD:" . " ")
       ("#+SUBTITLE:" . " ")
       ("#+AUTHOR:" . " ")
       (":Effort:" . " ")
       ("SCHEDULED:" . " ")
       ("DEADLINE:" . " ")
       ("#+begin_defn" .  " ")
       ("#+end_defn" . " ")))
(prettify-symbols-mode)
(let ((fontset (face-attribute 'default :fontset)))
  (set-fontset-font fontset '(?\xf000 . ?\xf2ff) "FontAwesome" nil 'append)))))
```

5.1.2 | Property Drawers

```
(defun org-cycle-hide-drawers (state)
  "Re-hide all drawers after a visibility state change."
  (when (and (derived-mode-p 'org-mode)
 (not (memq state '(overview folded contents))))
(save-excursion
  (let* ((globalp (memq state '(contents all)))
 (beg (if globalp
(point-min)
(point)))
 (end (if globalp
(point-max)
(if (eq state 'children)
  (save-excursion
(outline-next-heading)
(point))
  (org-end-of-subtree t)))))
(goto-char beg)
(while (re-search-forward org-drawer-regexp end t)
  (save-excursion
(beginning-of-line 1)
(when (looking-at org-drawer-regexp)
  (let* ((start (1- (match-beginning 0)))
 (limit
   (save-excursion
```

```
 (outline-next-heading)
   (point)))
 (msg (format
(concat
  "org-cycle-hide-drawers:  "
  " line missing at position %s")
(1+ start))))
  (outline-flag-region start (point-at-eol) t)
  (user-error msg))))))))))
 (add-hook 'org-mode-hook (lambda () (org-cycle-hide-drawers 'all)))
```

5.2 | **Export**

```
(use-package org-special-block-extras
  :init
  (org-special-block-extras-mode)
  (org-special-block-extras-defblock collapsible (title "Details") (contents "")
     (format
      (pcase backend
(_ "<details>
       <summary> <i> %s </i> </summary>
       %s
    </details>"))
      title contents)))

(use-package org
  :init
  (setq org-html-text-markup-alist
'((bold . "<b>%s</b>")
  (code . "<code>%s</code>")
  (italic . "<i>%s</i>")
  (strike-through . "<del>%s</del>")
  (underline . "<span class=\"underline\">%s</span>")
  (verbatim . "<kbd>%s</kbd>")))
  (setq org-html-head "<link rel=\"stylesheet\" href=\"https://quantumish.github.io/org.css\">")
  (setq org-html-postamble nil)
  (setq org-export-with-section-numbers nil)
  (setq org-export-with-toc nil)
  (setq org-publish-project-alist
'(("github.io"
   :base-directory "~/Dropbox/publicnotes/"
   :base-extension "org"
   :publishing-directory "~/richardfeynmanrocks.github.io/notes/"
   :recursive t
   :publishing-function org-html-publish-to-html
   :headline-levels 4
   :html-extension "html"
   :with-toc nil
   :section-numbers nil
   :html-head "<link rel=\"stylesheet\" href=\"https://richardfeynmanrocks.github.io/org.css\">"
   :preserve-breaks t
   ))))
```

6 | Notes

```
(use-package org-roam
  :init
  (setq org-roam-directory "~/sync/notes")
  (setq org-roam-v2-ack t)
  :bind
  ("C-c n i" . org-roam-node-insert)
  ("C-c n f" . org-roam-node-find)
  ("C-c n s" . org-roam-db-sync))

(use-package org-roam-ui
  :straight
  (:host github :repo "org-roam/org-roam-ui" :branch "main" :files ("*.el" "out"))
  :after org-roam
  ;; :hook (after-init . org-roam-ui-mode)
  :config
  (setq org-roam-ui-sync-theme t
org-roam-ui-follow t
org-roam-ui-update-on-save t
org-roam-ui-open-on-start t))

(use-package deft
  :init
  (setq deft-directory org-roam-directory)
  (defun my/deft-parse-title (file contents)
    "Parse the given FILE and CONTENTS and determine the title.
If `deft-use-filename-as-title' is nil, the title is taken to
be the first non-empty line of the FILE.  Else the base name of the FILE is
used as title."
    (let ((begin (string-match "^#\\+[tT][iI][tT][lL][eE]: .*$" contents)))
      (if begin
  (string-trim (substring contents begin (match-end 0)) "#\\+[tT][iI][tT][lL][eE]: *" "[\n\t ]+")
(deft-base-filename file))))

  (advice-add 'deft-parse-title :override #'my/deft-parse-title)

  (setq deft-strip-summary-regexp
(concat "\\("
"[\n\t]" ;; blank
"\\|^#\\+[[:alpha:]_]+:.*$" ;; org-mode metadata
"\\)")))
```

6.1 | Taproot-specific

```
(defvar taproot-dir (concat (getenv "HOME") "/taproot3"))
```

Let's define a function to export to Taproot:

```
(defun org-roam-export-to-taproot ()
  (interactive)
  (call-process-shell-command (concat "cp -R " org-roam-directory "*.org " taproot-dir)))
```

Also, a function to convert to file-based links would be nice.

```
(defun org-roam-emergency-exit (in-path out-path)
  "Emergency exit from Org Roam v2.
   Returns list of commands to convert notes in IN-PATH to traditional format in OUT-PATH."
  (let ((sed (if (eq system-type 'darwin) "gsed" "sed")))
    (progn
      (call-process-shell-command (concat "cp -R " in-path "*.org " out-path))
      (dolist (pair (org-roam-db-query [:select [ID FILE] :from nodes]))
(call-process-shell-command (concat sed " -i \"s/id:" (car pair)
    "/file:" (substring (car (cdr pair)) (length in-path))
    "/g\" " out-path "*.org")))
```

And a function to open in Taproot:

```
(defun org-roam-open-in-taproot ()
  (interactive)
  (if (not (eq (buffer-file-name) nil))
      (if (string= (substring (buffer-file-name) 0 (length taproot-dir)) taproot-dir)
  (call-process-shell-command
   (concat "open https://taproot3.sanity.gq"
   (substring (file-name-sans-extension (buffer-file-name)) (length taproot-dir))))
(message "Not a Taproot buffer!"))
    (message "Not a file buffer!")))
```

7 | Productivity

7.1 | Agenda

```
(use-package org
  :init
  (defvar org-inbox-file (concat (getenv "HOME") "/sync/org/inbox.org"))
  (defvar org-completed-file "~/sync/org/completed.org")
  (setq org-archive-location (concat org-completed-file "::"))
  (setq org-agenda-files `(,org-inbox-file ,org-completed-file))
  :general
  ("C-c o i" #'(lambda () (interactive) (find-file org-inbox-file)))
  ("C-c o a" #'(lambda () (interactive) (org-agenda 'a))))
```

7.2 | Projects

```
(use-package org
  :init
  (setq org-enforce-todo-dependencies t)
  (setq org-enforce-todo-checkbox-dependencies t)
  (setq org-agenda-dim-blocked-tasks t))
```

8 | Development

8.1 | **Terminal**

```
(use-package vterm)


     (defun dw/get-prompt-path ()
      (let* ((current-path (eshell/pwd))
     (git-output (shell-command-to-string "git rev-parse --show-toplevel"))
     (has-path (not (string-match "^fatal" git-output))))
(if (not has-path)
  (abbreviate-file-name current-path)
  (string-remove-prefix (file-name-directory git-output) current-path))))

    ;; This prompt function mostly replicates my custom zsh prompt setup
    ;; that is powered by github.com/denysdovhan/spaceship-prompt.

(defun dw/eshell-prompt ()
      (concat
       "\n"
       (propertize "davfrei" 'face `(:foreground ,(doom-color 'orange)) 'read-only t)
       (propertize " " 'face `(:foreground "white") 'read-only t)
       (propertize (dw/get-prompt-path) 'face `(:foreground ,(doom-color 'orange)) 'read-only t)
       (propertize " · " 'face `(:foreground "white") 'read-only t)
       (propertize (format-time-string "%I:%M:%S %p") 'face `(:foreground ,(doom-color 'cyan)) 'read-onl
       (if (= (user-uid) 0)
   (propertize "\n#" 'face `(:foreground "red2") 'read-only t)
 (propertize "\n " 'face `(:foreground ,(doom-color 'blue)) 'read-only t))
       (propertize " " 'face `(:foreground ,(doom-color 'fg)))
       ))

    (defun dw/eshell-configure ()
      (use-package xterm-color)

      (push 'eshell-tramp eshell-modules-list)
      (push 'xterm-color-filter eshell-preoutput-filter-functions)
      (delq 'eshell-handle-ansi-color eshell-output-filter-functions)

      ;; Save command history when commands are entered
      (add-hook 'eshell-pre-command-hook 'eshell-save-some-history)

      (add-hook 'eshell-before-prompt-hook
(lambda ()
  (setq xterm-color-preserve-properties t)))

      ;; Truncate buffer for performance
      (add-to-list 'eshell-output-filter-functions 'eshell-truncate-buffer)

      ;; We want to use xterm-256color when running interactive commands
      ;; in eshell but not during other times when we might be launching
      ;; a shell command to gather its output.
      (add-hook 'eshell-pre-command-hook
(lambda () (setenv "TERM" "xterm-256color")))
      (add-hook 'eshell-post-command-hook
(lambda () (setenv "TERM" "dumb")))
```

```
      ;; Use completion-at-point to provide completions in eshell
      (define-key eshell-mode-map (kbd "<tab>") 'completion-at-point)

      ;; Initialize the shell history
      (eshell-hist-initialize)

      (setenv "PAGER" "cat")

      (setq eshell-prompt-function      'dw/eshell-prompt
    eshell-prompt-regexp        "^  "
    eshell-history-size         10000
    eshell-buffer-maximum-lines 10000
    eshell-hist-ignoredups t
    eshell-highlight-prompt t
    eshell-scroll-to-bottom-on-input t
    eshell-prefer-lisp-functions nil))

    (use-package eshell
      :hook (eshell-first-time-mode . dw/eshell-configure)
      :init
      (setq eshell-directory-name "~/.dotfiles/.emacs.d/eshell/"
    eshell-aliases-file (expand-file-name "~/.dotfiles/.emacs.d/eshell/alias")))

    (use-package eshell-z
      :hook ((eshell-mode . (lambda () (require 'eshell-z)))
     (eshell-z-change-dir .  (lambda () (eshell/pushd (eshell/pwd))))))

    (use-package exec-path-from-shell
      :init
      (setq exec-path-from-shell-check-startup-files nil)
      :config
      (when (memq window-system '(mac ns x))
(exec-path-from-shell-initialize)))

    (setq eshell-prompt-function 'dw/eshell-prompt)

    (use-package esh-autosuggest
      :hook (eshell-mode . esh-autosuggest-mode))

    (use-package eshell-toggle
      :straight (eshell-toggle :type git :host github :repo "4DA/eshell-toggle")
      :init
      (setq eshell-toggle-size-fraction 4)
      (setq eshell-toggle-use-projectile-root t)
      (setq eshell-toggle-run-command nil))

    (use-package eshell-up) ;; TODO eshell-up

    ;; (use-package eshell-info-banner
    ;;   :straight (eshell-info-banner :type git :host github
    ;;  :repo "phundrak/eshell-info-banner.el")
    ;;   :hook (eshell-banner-load . eshell-info-banner-update-banner))

    (use-package eshell-manual
```

```
      :straight (eshell-manual :type git :host github
       :repo "nicferrier/eshell-manual"))

    ;; (use-package eshell-fringe-status
    ;;   :init
    ;;   (setq eshell-fringe-status-success-bitmap 'my-flycheck-fringe-indicator)
    ;;   (setq eshell-fringe-status-failure-bitmap 'my-flycheck-fringe-indicator)
    ;;   :hook (eshell-mode . eshell-fringe-status-mode))

    ;; (use-package esh-help
    ;;   :init (setup-esh-help-eldoc))
```

8.2 | LSP

`lsp-mode` enables us to get Intellisense-esque features in Emacs: setting it up requires both config on Emacs' side and installing actual language servers on your side. We'll auto-install them with the magic of `use-package-ensure-system-package`, although brace yourself for the potential for lots of debugging if the server doesn't work as expected on your system.

`lsp-mode` can do more than just provide good completions: you can jump to definitions and references with `lsp-find-definition` and `lsp-find-references` respectively, as well as most other things you'd expect from an IDE.

```
(use-package lsp-mode
  ; :ensure-system-package ccls
  ; :ensure-system-package (pyls . "python -m pip install pyls")
  ; :ensure-system-package rust-analyzer
  :init
  ;; Disable annoying headerline
  (setq lsp-headerline-breadcrumb-enable nil)
  ;; Don't show unneeded function info in completions
  (setq lsp-completion-show-detail nil)
  ;; Disable annoying autoformatting!
  (setq-default lsp-enable-indentation nil)
  (setq-default lsp-enable-on-type-formatting nil)
  :commands lsp
  ;; Add languages of your choice!
  :hook ((c-mode . lsp)
 (c++-mode . lsp)
 (python-mode . lsp)
 (typescript-mode . lsp)
 (rust-mode . lsp)))

(use-package lsp-ui
  :after lsp
  :init
  (setq lsp-ui-doc-delay 5)
  (add-hook 'flycheck-mode-hook 'lsp-ui-mode) ;; HACK
  :config
  (eval `(set-face-attribute 'lsp-ui-doc-background nil :background ,(doom-darken 'bg .2))))
```

8.3 | Company

`company-mode` provides code completions in Emacs, and will work together with `lsp-mode` to provide a nice experience. On top of that, let's use add-ons that allow documentation for completions to pop up and also let `prescient` make things better like it did with Ivy.

```
(use-package company
  :init
  (setq company-idle-delay 0)
  (setq company-tooltip-maximum-width 40)
  :hook
  (prog-mode . company-mode))

(use-package company-quickhelp
  :after company
  :init (company-quickhelp-mode))

(use-package company-quickhelp-terminal
  :after company-quickhelp)

(use-package company-prescient
  :after company prescient
  :init
  (setq-default history-length 1000)
  (setq-default prescient-history-length 1000)
  :init (company-prescient-mode))
```

8.4 | Compilation

```
(use-package kv)

(require 'kv)
(defvar custom-compile-cmds
  '((rustic-mode . ((debug . "cargo build")
    (release . "cargo build --release")
    (test . "cargo test")))
    (c++-mode . ((cmake . "cmake .")
 (test . "ctest")
 (make . "make")
 (this . "g++ $this.cpp -std=c++17 -o $this")
 (this-speedy . "g++ $this.cpp -O3 -std=c++17 -o $this")
 (this-python . "g++ -shared -std=c++17 -undefined_dynamic_lookup `python3 -m pybind11 --includes` $this
    (c-mode . ((make . "make")
       (this . "gcc $this.c -o $this")
       (this-speedy . "gcc $this.c -O3 -o $this")
       (this-archive . "gcc $this.c -O -c -g && ar rcs $this.a $this.o")
       (this-mpi . "mpicc $this.c -o $this")))
    (cuda-mode . ((this . "nvcc $this.cu -o $this")))
    (python-mode . ((this-types . "mypy $this.py --ignore-missing-imports --strict")
    (this-cython . "cython --embed -o $this.c $this.py -3 && sudo gcc $this.c -o $this -I/usr/include/py
    ))

(defun compile-dwim ()
```

```
  (interactive)
  (let ((list (cdr (assoc major-mode custom-compile-cmds)))) ;; Debugging is for suckers
    (ivy-read "Compilation preset: " (kvalist->keys list)
      :preselect (car (kvalist->keys list))
      :action (lambda (name)
(compile
 (replace-regexp-in-string
  (regexp-quote "$this")
  (file-name-sans-extension (buffer-file-name))
  (cdr (assoc (intern-soft name) list))))))))

(use-package compile
  :config
  (setq compilation-scroll-output t)
  (setq compilation-ask-about-save nil)
  (defun compile-project ()
    (interactive)
    (let ((default-directory (projectile-project-root)))
      (call-interactively 'compile-dwim)))
  (require 'ansi-color)
  (defun colorize-compilation-buffer ()
    (toggle-read-only)
    (ansi-color-apply-on-region compilation-filter-start (point))
    (toggle-read-only))
  (add-hook 'compilation-filter-hook 'colorize-compilation-buffer)
  :bind (:map prog-mode-map
      ("C-;" . compile-project))
  :hook
  (compilation-mode . hide-mode-line-mode)
  ; (compilation-mode . (lambda () (set-header-line 200)))
  (compilation-start . olivetti-mode)
  (compilation-start . determine-olivetti))
```

8.5 | Documentation

```
(defun minimal-browse-url (url)
  "Browse an arbitrary url (as URL) in a new frameless Firefox window."
  (split-window-right)
  (other-window 1)
  (call-process-shell-command (concat "firefox -P default-release --new-window " url) nil 0))

(use-package dash-docs)
(use-package counsel-dash
  :config
  (setq dash-docs-browser-func 'minimal-browse-url)
  (setq dash-docs-enable-debugging nil)
  (defun emacs-lisp-doc ()
    "Restrict dash docsets to Emacs Lisp."
    (interactive)
    (setq-local dash-docs-docsets '("Emacs Lisp")))
  (defun c-doc ()
    "Restrict dash docsets to C."
    (interactive)
```

```
    (setq-local dash-docs-docsets '("C")))
  (defun c++-doc ()
    "Restrict dash docsets to C/C++."
    (interactive)
    (setq-local dash-docs-docsets '("C" "C++")))
  (defun rust-doc ()
    "Restrict dash docsets to Rust."
    (interactive)
    (setq-local dash-docs-docsets '("Rust")))
  (defun python-doc ()
    "Restrict dash docsets to Python."
    (interactive)
    (setq-local dash-docs-docsets '("Python 3")))
  :bind (:map prog-mode-map
      ("C-c d" . counsel-dash)
      ("C-c C-d" . counsel-dash-at-point))
  :hook
  (emacs-lisp-mode . emacs-lisp-doc)
  (c-mode . c-doc)
  (c++-mode . c++-doc)
  (python-mode . python-doc)
  (rustic-mode . rust-doc)
  (rust-mode . rust-doc))
```

8.6 | TODO Projectile?

8.7 | Linting

Next, we can add linting to the editor with flycheck!

```
(use-package flycheck
  :hook
  (prog-mode . flycheck-mode)
  (flycheck-mode . (lambda () (set-window-fringes nil 15 0))))
```

With a tweak courtesy of @jemoka, we can smooth over bits of the interface. Goodbye squiggly lines and strange fringe indicators. Goodbye linter errors while typing.

```
(use-package flycheck
  :config
  (setq flycheck-check-syntax-automatically '(mode-enabled save))
  (set-face-attribute 'flycheck-error nil :underline `(:color ,(doom-color 'orange)))
  (set-face-attribute 'flycheck-warning nil :underline `(:color ,(doom-color 'blue)))
  (set-face-attribute 'flycheck-info nil :underline t)
  (define-fringe-bitmap 'my-flycheck-fringe-indicator
    (vector #b00000000
    #b00000000
    #b00000000
    #b00000000
    #b00000000
    #b00000000
    #b00000000
    #b00011100
```

```
    #b00111110
    #b00111110
    #b00111110
    #b00011100
    #b00000000
    #b00000000
    #b00000000
    #b00000000
    #b00000000))
  (let ((bitmap 'my-flycheck-fringe-indicator))
    (flycheck-define-error-level 'error
      :severity 2
      :overlay-category 'flycheck-error-overlay
      :fringe-bitmap bitmap
      :error-list-face 'flycheck-error-list-error
      :fringe-face 'flycheck-fringe-error)
    (flycheck-define-error-level 'warning
      :severity 1
      :overlay-category 'flycheck-warning-overlay
      :fringe-bitmap bitmap
      :error-list-face 'flycheck-error-list-warning
      :fringe-face 'flycheck-fringe-warning)
    (flycheck-define-error-level 'info
      :severity 0
      :overlay-category 'flycheck-info-overlay
      :fringe-bitmap bitmap
      :error-list-face 'flycheck-error-list-info
      :fringe-face 'flycheck-fringe-info)))
```

#+end$_{\text{collapsible}}$

8.8 | **Snippets**

YASnippet is the premiere package for snippets, so let's install it.

```
(use-package yasnippet
  :init (yas-global-mode))
```

`auto-activating-snippets` provides the very useful ability to automatically expand snippets while typing.

```
  (use-package aas
    :hook (LaTeX-mode . ass-activate-for-major-mode)
    :hook (org-mode . ass-activate-for-major-mode)
    :hook (c-mode . ass-activate-for-major-mode)
    :hook (c++-mode . ass-activate-for-major-mode)
    :config
    (aas-set-snippets 'c-mode
      "u64" "uint64_t"~
      "u32" "uint32_t"
      "u16" "uint16_t"
      "u8" "uint8_t"
      "i64" "int64_t"
      "i32" "int32_t"
```

```
      "i16" "int16_t"
      "i8" "int8_t"
      "sz" "size_t")
    (aas-set-snippets 'c++-mode
      "mxf" "Eigen::MatrixXf"
      "mxd" "Eigen::MatrixXd"
      "v2f" "Eigen::Vector2f"
      "v2d" "Eigen::Vector2d"
      "v2i" "Eigen::Vector2i"
      "v3f" "Eigen::Vector3f"
      "v3d" "Eigen::Vector3d"
      "v3i" "Eigen::Vector3i"))
(use-package laas
  :config ; do whatever here
  (aas-set-snippets 'laas-mode
    ;; set condition!
    :cond #'texmathp ; expand only while in math
    "bff" (lambda () (interactive)
    (yas-expand-snippet "\\mathbf{$1}$0"))
    ;; add accent snippets
    :cond #'laas-object-on-left-condition
    "qq" (lambda () (interactive) (laas-wrap-previous-object "sqrt"))
    ))
```

8.9 | Git

Let's install the wonderful git porcelain Magit and some extra usefulness.

```
;; The ultimate Git porcelain.
(use-package magit)
;; Show all TODOs in a git repo
(use-package magit-todos)
;; Edit gitignores w/ highlighting
(use-package gitignore-mode)
```

8.10 | Language-Specific

```
(use-package rustic)
(use-package cuda-mode)
(use-package clojure-mode)
(use-package cmake-mode)
(use-package json-mode)
(use-package rust-mode) ;; for when rustic breaks
(use-package nim-mode)
(use-package zig-mode)
(use-package typescript-mode)
(use-package css-mode)
```

8.10.1 | TODO C++

```
(setq c-default-style "k&r")
(setq-default c-basic-offset 4)

(use-package modern-cpp-font-lock
  :init (modern-c++-font-lock-global-mode t))

(use-package cmake-mode)

(use-package cuda-mode)

(use-package ccls
  ; :ensure-system-package ccls
  :hook ((c-mode c++-mode cuda-mode) .
 (lambda () (require 'ccls) (lsp)))
  :custom
  (ccls-executable (executable-find "ccls")) ; Add ccls to path if you haven't done so
  (ccls-sem-highlight-method 'font-lock)
  (ccls-enable-skipped-ranges nil)
  :config
  (lsp-register-client
   (make-lsp-client
:new-connection (lsp-tramp-connection (cons ccls-executable ccls-args))
:major-modes '(c-mode c++-mode cuda-mode)
:server-id 'ccls-remote
:multi-root nil
:remote? t
:notification-handlers
(lsp-ht ("$ccls/publishSkippedRanges" #'ccls--publish-skipped-ranges)
("$ccls/publishSemanticHighlight" #'ccls--publish-semantic-highlight))
:initialization-options (lambda () ccls-initialization-options)
:library-folders-fn nil)))

;; TODO bind/investigate ccls functions

(use-package cpp-auto-include)
```

8.10.2 | TODO Python

```
(use-package ein)

(use-package lsp-mode
  :config
  (lsp-register-custom-settings
   '(("pyls.plugins.pyls_mypy.enabled" t t)
     ("pyls.plugins.pyls_mypy.live_mode" nil t)
     ("pyls.plugins.pyls_black.enabled" t t)
     ("pyls.plugins.pyls_isort.enabled" t t)
 ("pyls.plugins.flake8.enabled" t t)))
```

```
  (setq lsp-eldoc-enable-hover nil)

  :hook
  ((python-mode . lsp)))


(use-package buftra
  :straight (:host github :repo "humitos/buftra.el"))

(use-package py-pyment
    :straight (:host github :repo "humitos/py-cmd-buffer.el")
    :config
    (setq py-pyment-options '("--output=google")))

(use-package py-isort
    :straight (:host github :repo "humitos/py-cmd-buffer.el")
    :hook (python-mode . py-isort-enable-on-save)
    :config
    (setq py-isort-options '("-m=3" "-tc" "-fgw=0" "-ca")))

(use-package py-autoflake
    :straight (:host github :repo "humitos/py-cmd-buffer.el")
    :hook (python-mode . py-autoflake-enable-on-save)
    :config
    (setq py-autoflake-options '("--expand-star-imports")))

(use-package py-docformatter
    :straight (:host github :repo "humitos/py-cmd-buffer.el")
    :hook (python-mode . py-docformatter-enable-on-save)
    :config
    (setq py-docformatter-options '("--wrap-summaries=88" "--pre-summary-newline")))

(use-package blacken
    :straight t
    :hook (python-mode . blacken-mode)
    :config
    (setq blacken-line-length '100))

(use-package python-docstring
    :straight t
    :hook (python-mode . python-docstring-mode))
```

8.11 | TODO Code Aesthetics

```
(use-package hl-todo
  :init
  (global-hl-todo-mode)
  (doom-color 'red)
  (setq hl-todo-keyword-faces
`(("TODO"   . ,(doom-color 'green))
  ("FIXME"  . ,(doom-color 'red))
  ("DEBUG"  . ,(doom-color 'magenta))
  ("HACK"   . ,(doom-color 'violet))
```

```
  ("NOTE"   . ,(doom-color 'cyan))))
  ;; We already have todos in Org Mode!
  (add-hook 'org-mode-hook (lambda () (hl-todo-mode -1)))
  (set-face-attribute 'hl-todo nil :italic t)
  :bind (:map hl-todo-mode-map
  ("C-c t p" . hl-todo-previous)
  ("C-c t n" . hl-todo-next)
  ("C-c t i" . hl-todo-insert)))

(use-package rainbow-mode)
```

9 | TODO Writing

```
(use-package flyspell)
(use-package lexic
  :bind
  ("C-c w l" . lexic-search)
  ("C-c w w" . lexic-search-word-at-point))
(use-package gdoc
  :straight (gdoc :type git :host github :repo "jemoka/gdoc.el"))
```

Also, Google-Docs esque comments:

```
#+begin_src emacs-lisp
  ;; Google Docs style comments
  (use-package org-marginalia
    :straight (:host github :repo "nobiot/org-marginalia")
    :init (add-hook 'org-mode-hook 'org-marginalia-mode)
    (defun org-marginalia-save-and-open (point)
      (interactive "d")
      (org-marginalia-save)
      (org-marginalia-open point))
    :bind (:map org-marginalia-mode-map
("C-c n o" . org-marginalia-save-and-open)
("C-c m" . org-marginalia-mark)
("C-c n ]" . org-marginalia-next)
("C-c n [" . org-marginalia-prev)))
```

#+end$_{src}$

10 | TODO Vanilla++

```
(use-package crux
  :bind
  (("C-a" . crux-move-beginning-of-line) ;; Move to beginning of text, not line.
   ("C-x 4 t" . crux-transpose-windows)
   ("C-x K" . crux-kill-other-buffers)
   ("C-k" . crux-smart-kill-line))
  :config
  (crux-with-region-or-buffer indent-region)
```

```
  (crux-with-region-or-buffer untabify)
  (crux-with-region-or-point-to-eol kill-ring-save)
  (defalias 'rename-file-and-buffer #'crux-rename-file-and-buffer))

(use-package goto-line-preview
  :init (general-define-key "M-g M-g" 'goto-line-preview
    "C-x n g" 'goto-line-relative-preview))

(use-package all-the-icons-dired
  :hook (dired-mode . all-the-icons-dired-mode))

(use-package diredfl
  :init (diredfl-global-mode))
```

11 | Fun

FIXME

```
(use-package pdf-tools)
```

11.1 | Exit Message

```
(setq exit-messages '(
"Please don't leave, there's more demons to toast!"
"Let's beat it -- This is turning into a bloodbath!"
"I wouldn't leave if I were you. Vim is much worse."
"Don't leave yet -- There's a demon around that corner!"
"Ya know, next time you come in here I'm gonna toast ya."
"Go ahead and leave. See if I care."
"Are you sure you want to quit this great editor?"
"Emacs will remember that."
"Emacs, Emacs never changes."
"Okay, look. We've both said a lot of things you're going to regret..."
"Look, bud. You leave now and you forfeit your body count!"
"Get outta here and go back to your boring editors."
"You're lucky I don't smack you for thinking about leaving."
"Don't go now, there's a dimensional shambler waiting at the prompt!"
"Just leave. When you come back I'll be waiting with a bat."
"Are you a bad enough dude to stay?"
"It was worth the risk... I assure you."
"I'm willing to take full responsibility for the horrible events of the last 24 hours."
))

(defun random-choice (items)
  (let* ((size (length items))
 (index (random size)))
(nth index items)))

(defun save-buffers-kill-emacs-with-confirm ()
  (interactive)
  (if (null current-prefix-arg)
```

```
  (if (y-or-n-p (format "%s Quit? " (random-choice exit-messages)))
(save-buffers-kill-emacs))
(save-buffers-kill-emacs)))

(global-set-key "\C-x\C-c" 'save-buffers-kill-emacs-with-confirm)
```

11.2 | Spotify

Smudge is nice.

```
(use-package smudge
  :straight (smudge :type git :host github :repo "danielfm/smudge"
    :fork (:host github :repo "richardfeynmanrocks/smudge"))
  :init
  (setq smudge-status-location nil)
  ;; FIXME actively destructive to potential mode-line config!
  (setq global-mode-string '(("   ")))
  :bind
  ("C-S-s-l" . smudge-controller-next-track)
  ("C-S-s-h" . smudge-controller-previous-track)
  ("C-S-s-j" . smudge-controller-volume-down)
  ("C-S-s-k" . smudge-controller-volume-up)
  ("C-S-s-p" . smudge-controller-toggle-play)
  ("C-S-s-s" . smudge-controller-toggle-shuffle)
  ("C-S-s-r" . smudge-controller-toggle-repeat))
```

12 | Scratch

13 | The End.

Well, that's it. We're done. Time to get going!

```
(require 'notifications)
(notifications-notify :title "Up and at 'em!"
      :body (format "Loaded %d packages in %s with %d GCs."
 (length package-activated-list)
 (format "%.2f seconds"
 (float-time
  (time-subtract after-init-time before-init-time)))
 gcs-done))
```