My Emacs Configuration

Straight

Bootstrap straight.el for installing packages

```
(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))

(straight-use-package 'use-package)
;; (setq use-package-verbose t)
```

Custom Settings

Set seperate custom file so that custom settings don't get overwritten by org-babel-tangle.

```
(setq custom-file "~/.emacs.d/custom.el")
(load custom-file)
```

Emacs

This section is for miscellaneous settings which do not belong elsewhere, everything should be commented explaining why it was added.

```
;; clean scratch buffer
(setq initial-scratch-message nil)
```

```
;; fix sizing errors with tiling window managers
(setq frame-resize-pixelwise t)
```

Backup

```
(setq backup-directory-alist `(("." . "~/.saves")))
(setq backup-by-copying t)
(setq delete-old-versions t
      kept-new-versions 6
      kept-old-versions 2
      version-control t)
```

General

This section sets up general.el. This section should not be used to set keybindings (except for the most general ones). Keybindings should go with their respective packages.

```
(use-package general
  :straight t
  :config
  (general-auto-unbind-keys)
  (general-evil-setup)

  (general-create-definer leader-key-def
    :keymaps '(normal insert visual emacs)
    :prefix "SPC"
    :global-prefix "C-SPC")
  )
```

Dired

```
(use-package dired
  :custom
  (dired-listing-switches "-ABDgho --group-directories-first")
  :general
  (general-def
    :states 'normal
    :keymaps 'dired-mode-map
    "h" 'dired-up-directory
    "l" 'dired-find-file)
  )
```

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


(use-package dired-x
  :after general
  :init
  (leader-key-def
    "<SPC>" 'dired-jump
    "m" '(lambda () (interactive) (find-file "~/.local/share/chezmoi/dot_emacs.d/emacs.org")
    ))

(global-auto-revert-mode)
(setq global-auto-revert-non-file-buffers t)
```

Minibuffer

Configuration for the minibuffer. Currently using vertico.

```
;; Enable vertico
(use-package vertico
  :straight t
  :commands execute-extended-command
  :init
  (vertico-mode)

  ;; Grow and shrink the Vertico minibuffer
  ;; (setq vertico-resize t)

  ;; Optionally enable cycling for `vertico-next' and `vertico-previous'.
  ;; (setq vertico-cycle t)
  )

;; Use the `orderless' completion style.
;; Enable `partial-completion' for file path expansion.
;; You may prefer to use `initials' instead of `partial-completion'.
(use-package orderless
  :straight t
  :after vertico
  :custom
  (completion-styles '(orderless))
  (completion-category-defaults nil)
  (completion-category-overrides '((file (styles partial-completion)))))
```

```
;; Persist history over Emacs restarts. Vertico sorts by history position.
(use-package savehist
  :after vertico
  :init
  (savehist-mode))

;; A few more useful configurations...
(use-package emacs
  :after vertico
  :custom
  (minibuffer-prompt-properties
   '(read-only t cursor-intangible t face minibuffer-prompt))
  :init
  ;; Add prompt indicator to `completing-read-multiple'.
  ;; Alternatively try `consult-completing-read-multiple'.
  (defun crm-indicator (args)
    (cons (concat "[CRM] " (car args)) (cdr args)))
  (advice-add #'completing-read-multiple :filter-args #'crm-indicator)

  ;; Do not allow the cursor in the minibuffer prompt
  (add-hook 'minibuffer-setup-hook #'cursor-intangible-mode)

  ;; Emacs 28: Hide commands in M-x which do not work in the current mode.
  ;; Vertico commands are hidden in normal buffers.
  ;; (setq read-extended-command-predicate
  ;;       #'command-completion-default-include-p)

  ;; Enable recursive minibuffers
  (setq enable-recursive-minibuffers t))

(use-package consult
  :straight t
  :after vertico
  :custom
  (consult-project-root-function #'projectile-project-root)
  :config
  (autoload 'projectile-project-root "projectile")

  :general
  (leader-key-def
    "/" 'consult-line
    "b" 'consult-buffer
    "r" 'consult-recent-file
    "i" 'consult-imenu
    "s" 'consult-ripgrep
    ))
```

```
(use-package marginalia
  :straight t
  :after vertico
  :custom
  (marginalia-annotators '(marginalia-annotators-heavy))
  :init
  (marginalia-mode)
  )
```

Eshell

Configuration and setup for eshell

```
(use-package eshell
  :init
  (defun eshell-other-window ()
    "Open a `eshell' in a new window."
    (interactive)
    (let ((buf (eshell)))
      (switch-to-buffer (other-buffer buf))
      (switch-to-buffer-other-window buf)))
  :general
  (leader-key-def
    "t" 'eshell-other-window)
  )
```

Org Mode

```
(use-package org
  :straight
  :defer t
  :custom
  (org-confirm-babel-evaluate nil)
  (org-src-window-setup 'current-window)
  (org-M-RET-may-split-line nil)

  (org-directory "~/Dropbox/org")
  (org-default-notes-file (concat org-directory "/notes.org"))

  (org-capture-bookmark nil)
  (org-capture-templates
   '(("t" "add a [t]ask" entry (file (concat org-directory "/tasks.org"))
      "* TODO %?" :kill-buffer t)))
```

```
  (require 'org-tempo)

  :general
  ("C-c c" 'org-capture)
  ("C-c a" 'org-agenda)

  :config
  (org-babel-do-load-languages
   'org-babel-load-languages
   '((emacs-lisp . t)
     ))
  (add-to-list 'org-structure-template-alist '("el" . "src emacs-lisp"))
  (add-to-list 'org-agenda-files (concat org-directory "/tasks.org"))

  (use-package evil-org
    :straight t
    :after org
    :hook (org-mode . (lambda () evil-org-mode))
    :config
    (require 'evil-org-agenda)
    (evil-org-agenda-set-keys))


  :hook
  (org-mode . org-indent-mode)
  (org-capture-mode . evil-insert-state)
  )

(use-package org-superstar ;; has to be outside org use-package, messes up org export for s
  :straight t
  :after org
  :hook (org-mode . (lambda () (org-superstar-mode 1))))

(use-package ox-pandoc
  :straight t
  :after org
  )
```

Yequake

Create temporary emacs frames for simple tasks, i.e. org capture. Currently will not work unless an emacs frame is already open as per this issue, which sort of makes this useless for my purposes (which are quickly adding tasks from outside of emacs). Despite that issue, I am leaving this here in case the issue is fixed or I can find a non-hacky workaround, because the concept is really cool. Also,

doesn't seem to load properly.

```
;; (use-package yequake
;;   :custom
;;   (yequake-frames
;;    '(("org-capture"
;;       (buffer-fns . (yequake-org-capture))
;;       (width . 0.75)
;;       (height . 0.5)
;;       (alpha . 0.95)
;;       (frame-parameters . ((undecorated . t)
;;                            (skip-taskbar . t)
;;                            (sticky . t))))))
;;   )
```

EVIL

setup evil and related packages

```
(use-package evil
  :straight t
  :init
  (global-visual-line-mode 1)
  :custom
  (evil-undo-system 'undo-fu)
  (evil-want-keybinding nil) ;; needed for evil-collection
  (evil-want-C-u-scroll t)
  (evil-respect-visual-line-mode t)

  :config
  (evil-mode 1)
  (general-def
    "C-M-u" 'universal-argument ;; doesn't work with :general for some reason
    )

  (use-package undo-fu :straight t)
  (use-package evil-collection
    :straight t
    :after evil
    :config
    (evil-collection-init))
  (use-package evil-goggles
    :straight t
    :config
    (evil-goggles-mode 1))
```

```
  (use-package evil-commentary
    :straight t
    :config
    (evil-commentary-mode 1))
  (use-package evil-snipe
    :straight t
    :general
    (:states '(normal motion)
       "s" 'evil-snipe-s
       "S" 'evil-snipe-S)
    )

  (use-package evil-multiedit
    :straight t
    :config
    (evil-multiedit-default-keybinds)
    )
  (use-package evil-surround
    :straight t
    :config
    (global-evil-surround-mode 1))
  )
```

Help

```
(use-package which-key
  :defer t
  :straight t
  :init (which-key-mode)
  :custom
  (which-key-idle-delay 0.3))

(use-package helpful
  :straight t
  :general
  ("C-h f" 'helpful-callable
   "C-h v" 'helpful-variable
   "C-h k" 'helpful-key
   "C-c C-h" 'helpful-at-point
   )
  )

(use-package define-word
  :straight t
  :general
```

```
  (
   "C-h C-w" 'define-word-at-point
   ))
```

Elfeed

Setup for Elfeed, an RSS reader

```
(use-package elfeed
  :straight t
  :general
  ("C-x w" 'elfeed)
  :custom
  (elfeed-feeds
   '(
     ;; Blogs
     ("http://daviddfriedman.blogspot.com/atom.xml" blog)
     ("http://www.econlib.org/feed/indexCaplan_xml" blog)
     ("https://nullprogram.com/feed/" blog)
     ("https://feeds.feedburner.com/mrmoneymustache" blog)
     ("https://astralcodexten.substack.com/feed" blog)
     ("https://www.singlelunch.com/feed" blog)
     ("https://www.overcomingbias.com/feed" blog)
     ("https://planet.emacslife.com/atom.xml" blog)

     ;; Fora
     ("https://www.lesswrong.com/feed.xml?view=curated-rss" forum)

     ;; Comics
     ("https://xkcd.com/rss.xml" comic)
     ("https://www.monkeyuser.com/feed.xml" comic)
     ))
  :config
  (use-package elfeed-goodies
    :straight t
    :after elfeed
    :custom
    (byte-compile-warnings '(cl-functions)) ;; supress warning about cl.el deprecation
    :config
    (elfeed-goodies/setup)
    )
  )
```

Editing

This section contains packages and settings for non-evil specific editing

```
;; Delimiters
(use-package rainbow-delimiters
  :straight t
  :hook (prog-mode . rainbow-delimiters-mode))

(show-paren-mode 1)
(electric-pair-mode 1)
(setq electric-pair-inhibit-predicate 'electric-pair-conservative-inhibit)
```

Navigation

This section contains packages/configuration for non-evil-specific navigation

```
(use-package avy
  :straight t
  :general
  ("C-s" 'avy-goto-char-timer)
  )

(use-package winum
  :straight t
  :general
  (
   "M-1" 'winum-select-window-1
   "M-2" 'winum-select-window-2
   "M-3" 'winum-select-window-3
   "M-4" 'winum-select-window-4
   "M-5" 'winum-select-window-5
   "M-6" 'winum-select-window-6
   "M-7" 'winum-select-window-7
   "M-8" 'winum-select-window-8
   )
  :config
  (winum-mode t))

(use-package smartscan
  :straight t
  :hook (prog-mode . smartscan-mode))

(use-package rg
  :defer t
  :straight t
```

```
  :config
  (rg-enable-default-bindings))
```

Formatting

Automatic formatting

```
(use-package aggressive-indent
  :straight t
  :hook (prog-mode . aggressive-indent-mode)
  )

(use-package format-all
  :straight t
  :hook
  (prog-mode . format-all-mode)
  )
```

Git

Setup git integration

```
(use-package magit
  :commands magit-status
  :straight t
  )
```

Project Management

```
(use-package projectile
  :straight t
  :custom
  (projectile-switch-project-action #'projectile-dired)
  :config (projectile-mode)
  :general
  (leader-key-def
    "p" 'projectile-command-map
    ))
```

Major Modes

Set up major modes for languages, etc

```
(use-package toml-mode :straight t
  :mode "\\.toml\\'")
```

```
(use-package yaml-mode
  :straight t
  :mode "\\.yml\\'"
  )
(use-package rustic :straight t)
(use-package nix-mode :straight t
  :mode "\\.nix\\'")
(use-package json-mode :straight t
  :mode "\\.json\\'")

(use-package kbd-mode
  :straight (kbd-mode :type git :host github :repo "kmonad/kbd-mode")
  :mode "\\.kbd\\'")
```

Checkers

Set up checkers, i.e. syntax checking, spell checkers, etc

```
(use-package flycheck
  :straight t
  :defer t
  :init (global-flycheck-mode)
  )
```

Completion

```
(use-package company
  :defer t
  :straight t
  :custom
  (company-minimum-prefix-length 1)
  (company-idle-delay 0.0) ;; default is 0.2
  :config
  (global-company-mode)
  )
```

LSP

```
(use-package lsp-mode
  :straight t
  :custom
  (gc-cons-threshold 100000000) ;; set per the lsp-doctor recommendation
  (read-process-output-max (* 1024 1024)) ;; same reason ^
  (lsp-keymap-prefix "C-c l")
```

```
  :hook (
   (rustic-mode . lsp)
   (lsp-mode . lsp-enable-which-key-integration))
  :commands lsp)

(use-package lsp-ui
  :straight t
  :hook (lsp-mode . lsp-ui-mode)
  :commands lsp-ui-mode)
(use-package lsp-treemacs
  :straight t
  :after lsp-mode
  :commands lsp-treemacs-errors-list)
```

GUI

Set gui settings, theme, fonts, etc

```
;; disabling useless ui elements
(scroll-bar-mode -1)
(menu-bar-mode -1)
(tool-bar-mode -1)
(setq inhibit-startup-screen t)

(global-hl-line-mode)

;; theme
(use-package doom-themes
  :straight t
  :config (load-theme 'doom-one t))

;; dashboard
(use-package dashboard
  :straight t
  :custom
  (initial-buffer-choice (lambda () (get-buffer "*dashboard*")))
  (dashboard-startup-banner 'official)
  (dashboard-set-heading-icons t)
  (dashboard-set-file-icons t)
  (dashboard-set-init-info t)
  (dashboard-center-content t)
  (dashboard-items '((agenda . 5)
           (bookmarks . 5)
           (projects . 5)))
```

```
  :config
  (dashboard-setup-startup-hook))


;; font
(set-face-attribute 'default nil :font "Source Code Pro" :height 120)
(set-face-attribute 'fixed-pitch nil :font "Source Code Pro" :height 120)

;; line numbers
(dolist (mode '(text-mode-hook
      prog-mode-hook
      conf-mode-hook
      rust-mode-hook))
  (add-hook mode (lambda () (display-line-numbers-mode 1))))

;; modeline
(use-package doom-modeline
  :straight t
  :custom
  (doom-modeline-icon t) ;; fix icons in server
  :init
  (doom-modeline-mode 1))
```

local variables

;; Local Variables: ;; eval: (add-hook 'after-save-hook (lambda ()(if (y-or-n-p "Tangle?")(org-babel-tangle))) nil t) ;; End: