Basecamp

Jeffrey Szymanski

CONTENTS

compile using _run-latex-at-point_

Last compiled December 1, 2024.

1 Setup And Configuration

1.1 General Information And Good Practice For Base Computing Setup

1. Bash Configuration .bashrc is used for interactive shell customizations and .profile is used for setting up environment variables and system-wide settings needed during the login process.

   (a) Bashrc
   - The default bashrc lives at *etc/skel*.bashrc.
   - Other programs such as singularity or conda may write directly to bashrc.
   - Otherwise, bashrc is not altered automatically, as through Org-mode tangling. Instead, additional code is sourced though a simple if else statement. I source my basecamp shell libraries as follows:
   - Use functions instead of aliases. Functions are more flexible, and can be debugged.

1.2 An Opinionated And Incomplete Base Computing Configuration

1. Base operating system

   My base operating system is linux Ubuntu using a long-term support (LTS) version.

2. Bash

3. Git repositories Git repos live in ${HOME}/repos.

   Git repos are modular using the git submodule structure.

   Git repos are pushed to GitHub.

   Git version control follows a simple trunk-based development strategy, with a single active branch: master. Major changes can be spun out into separate branches, but should be quickly merged back to master after changes are validated. Stable, validated versions are occasionally saved as git tags.

   (a) Repositories where I am the main or sole author
       Repository code is tangled from a single Org-mode file. Updates from others are through pull requests as tangling would overwrite any direct commits.

4. Core directory structure – /

5. Data governance
   (a) Security
   (b) Architecture
   (c) Lifecycle
   (d) Management
       i. Backup

6. Core applications core components (incomplete)

   - installed via apt

     - syncthing

     - i3

     - emacs

     - texlive

     - texlive

     - rclone

     - okular

     - libreoffice

     - minor

       * tree

   - R

(a) R

- ☐ convert python and R to argparse https://www.google.com/search?q=r+argparse+optparse&oq=r+argparse+optparse&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDg2MDFqMGo0qAIAsAIA&sourceid=chrome-mobile&ie=UTF-8

(b) Python

- ☐ convert python and R to argparse https://www.google.com/search?q=r+argparse+optparse&oq=r+argparse+optparse&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDg2MDFqMGo0qAIAsAIA&sourceid=chrome-mobile&ie=UTF-8

(c) Emacs

i. Initialization
When my Emacs loads, several configuration files and directories are references by symbolic link from git repositories to the default ~/.emacs.d directory:

- init.el
- public_emacs_config.el
- private_emacs_config.el
- public_yasnippets
- private_yasnippets

These are loaded as optional files and if not found, they will throw a warning on initialization. These files are generated by Org-mode tangle.

The init.el file is compiled from the Org-mode header a simple wrapper for other Elisp files.

My public customized Emacs initialization is compiled from the Org-mode header basecamp.org::Public configuration and tangles to ./config/public_emacs_config.el.

A. Style and good practice
Use-package is my preferred package management system. By default, use-package loads are structured as:

(use-package package2
:ensure t
:init
:config
)

Each use-package keyword will expect a list or needs to be removed. Other keywords exist (see documentation).

- :ensure t will install any missing package
- :init is evaluated before package loading
- :config is evaluated after package loading

ii. Org-mode

A. Policy, style, and good practice

B. Block structures, source code, and tangling
Lowercase is preferred for all block notation, *e.g.*

instead of

At the file level, tangled code should reference it's location in orgmode files.
tangle defaults to ./

iii. Templating with YASnippets My public YASnippets are compiled from source code blocks across my Org-mode agenda files. These tangle into appropriate subdirectories of ./emacs/public_yasnippets. Private snippets are tangled into appropriate subdirectories of org/emacs/private_yasnippets.

2 Pan-computing Good Practive And General Style Guide

- Directory names should be in hyphen-case, using only lowercase letters (a-z), digits (0-9), and hyphens (-) as separators. Use as few words as possible
  - Can use more human-friendly title case and spaces with symlinks and shortcuts
- Prefer single-word file and directory names
- For multi-word file and directory names, prefer a dash (-) separator (*e.g.* a-longer-file-name.txt)

3 Conda

For repositories and projects with heavy use of Python and R, software should be managed through the Conda package manager.

3.1 My Basecamp Conda Environment

A basecamp conda environment is stored in this repository at basecamp_env.yaml.

4 Emacs

4.1 Setup, Configuration, And Customization

My setup assumes normal emacs initialization and package structure at ~/.emacs.d. The ~/.emacs.d/init.el ensures package management and then conditionally loads other lisp files, if they exist, in the following order:

1. ~/.emacs.d/load-early.el
2. ~/.emacs.d/config/
3. ~/.emacs.d/load-late.el

The ~/.emacs.d/config directory can contain symlinked lisp files from other locations. There is no assumed order within that directory.

All files mentioned here can be found in the Lisp Files appendix below.

load-path is a list of directories that Emacs searches when you use functions like require, load, or when Emacs needs to find a library or package. When you add a directory to load-path, you're telling Emacs where to look for Emacs Lisp files when they need to be loaded. My load path is simply the default package location ~/.emacs.d/elpa and also all files anywhere within ~/.emacs.d/lisp

- File backup
  - Local
    - Backup on save to ~/.emacs.d/backup-save-list
    - Emacs auto-saves on default settings to ~/.emacs.d/auto-save-list

4.2 Simple Emacs Lisp Tutorial

Alist

alist: association list, stores key-value pairs

progn: special form to evaluate sequence of expressions

4.3 Org-mode

If you place an asterisk at the beginning of your search, Org-mode will search only headlines (and not entry text). E.g., to find all entries with "emacs" in the headline, you could type:

4.4 Literate Programming With Emacs Org-mode

Generally comments should reside within the Org-mode structure and outside of code blocks. Tangling with a header argument :comments org will include both the header text and text between the header and code block. For example:

(property drawers are excluded from tangling with custom-org-remove-properties-drawer)

1. For example:
   (a) This header will be a comment This text in org and below the header will be a comment

   ```
   for file in "${HOME}"/repos/basecamp/lib/*.sh; do
       if [ -f "$file" ]; then
           source "$file"
       fi
   done
   ```

   This comment will not appear in the tangled code

   i. Child headers are not comments unless they contain more code blocks

   (b) Result:
   ls

4.5 YASnippets Style Guide And Good Practice

- - snippets are associated with a specific major mode, use of bash mode is discouraged
  - As snippets are associated with major modes, single word keywords are encouraged (e.g. function instead of bash.function)
  - snippets expand on tab
  - for modes with extensive snippet libraries, prefixes followed by a single period are preferred (e.g. mod.meeting)
  - prefixed snippets keywords are not likely to be confused with non-snippet terms, and they should expand without trigger
  - Snippet creation and storage
    - Snippets are created in org mode bash source code blocks with formatting as in the snippet
      - Each snippet resides under it's own terminal org mode header. The header consists of only the snippet keyword and tags including the :yas: tag (i.e. * <SNIPPET KEYWORD> :yas:)
      - The :yas: tag is reserved exclusively for yas block headers and does not have tag inheritance.
    - Snippets are stored in the main org repository under the directory./snippets which is symlinked to .emacs.d. Snippets are therefore under org version control.

4.6 Notes

See also Emacs and Org-mode use in my LaTeX repository (GitHub, local buffer).

5 Appendices

5.1 Lisp Files

1. Initialization

```
;;-*- mode: elisp -*-

;; Package Management Setup

(require 'package)

(add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t)

;; Ensure 'use-package' is installed
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))

(require 'use-package)
(setq use-package-always-ensure t)
```

```
(defun add-to-load-path-if-exists (dir)
  "Add DIR and its subdirectories to the Emacs load-path if DIR exists."
  (if (file-directory-p dir)
      (let ((default-directory dir))
        (message "Adding %s and its subdirectories to load-path" dir)
        (normal-top-level-add-to-load-path '("."))
        (normal-top-level-add-subdirs-to-load-path))
    (message "Directory %s does not exist, skipping." dir)))

(add-to-load-path-if-exists "~/.emacs.d/lisp/")

;; Function to safely load a file if it exists
(defun safe-load-file-if-exists (filepath)
  "Safely load the Emacs Lisp file at FILEPATH if it exists."
  (when (file-exists-p filepath)
    (condition-case err
        (load (file-name-sans-extension filepath))
      (error (message "Error loading %s: %s" filepath err)))))

;; Load early configuration
(safe-load-file-if-exists "~/.emacs.d/load-first.el")

;; Define the path to your configuration directory
(defvar my-config-dir "~/.emacs.d/config/"
  "Directory containing personal Emacs configuration files.")

;; Load all .el files in the config directory
(when (file-directory-p my-config-dir)
  (dolist (file (directory-files my-config-dir t "\\.el$"))
    (condition-case err
        (load (file-name-sans-extension file))
      (error (message "Error loading %s: %s" file err)))))

;; Load late configuration
(safe-load-file-if-exists "~/.emacs.d/load-last.el")
(custom-set-variables
 ;; custom-set-variables was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(ess-R-font-lock-keywords
   '((ess-R-fl-keyword:modifiers . t)
     (ess-R-fl-keyword:fun-defs . t)
     (ess-R-fl-keyword:keywords . t)
     (ess-R-fl-keyword:assign-ops . t)
     (ess-R-fl-keyword:constants . t)
     (ess-fl-keyword:fun-calls . t)
     (ess-fl-keyword:numbers . t)
     (ess-fl-keyword:operators . t)
     (ess-fl-keyword:delimiters . t)
     (ess-fl-keyword:= . t)
     (ess-R-fl-keyword:F&T . t)
     (ess-R-fl-keyword:%op% . t)))
 '(package-selected-packages
   '(cape corfu casual-agenda casual embark-consult embark yaml-mode yaml ws-butler web-mode vertico use
(custom-set-faces
 ;; custom-set-faces was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(default ((t (:family "Hack" :height 114 :weight light))))
 '(vterm-color-blue ((t (:foreground "#477EFC" :background "#477EFC")))))
```

2. Load First

```
;;-*- mode: elisp -*-

;; Package Management Setup

(require 'package)

(add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t)

;; Ensure 'use-package' is installed
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))

(require 'use-package)
(setq use-package-always-ensure t)

(defun add-to-load-path-if-exists (dir)
  "Add DIR and its subdirectories to the Emacs load-path if DIR exists."
  (if (file-directory-p dir)
      (let ((default-directory dir))
        (message "Adding %s and its subdirectories to load-path" dir)
        (normal-top-level-add-to-load-path '("."))
        (normal-top-level-add-subdirs-to-load-path))
    (message "Directory %s does not exist, skipping." dir)))

(add-to-load-path-if-exists "~/.emacs.d/lisp/")

;; Function to safely load a file if it exists
(defun safe-load-file-if-exists (filepath)
  "Safely load the Emacs Lisp file at FILEPATH if it exists."
  (when (file-exists-p filepath)
    (condition-case err
        (load (file-name-sans-extension filepath))
      (error (message "Error loading %s: %s" filepath err)))))

;; Load early configuration
(safe-load-file-if-exists "~/.emacs.d/load-first.el")

;; Define the path to your configuration directory
(defvar my-config-dir "~/.emacs.d/config/"
  "Directory containing personal Emacs configuration files.")

;; Load all .el files in the config directory
(when (file-directory-p my-config-dir)
  (dolist (file (directory-files my-config-dir t "\\.el$"))
    (condition-case err
        (load (file-name-sans-extension file))
      (error (message "Error loading %s: %s" file err)))))

;; Load late configuration
(safe-load-file-if-exists "~/.emacs.d/load-last.el")
```

```
(custom-set-variables
 ;; custom-set-variables was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(ess-R-font-lock-keywords
   '((ess-R-fl-keyword:modifiers . t)
     (ess-R-fl-keyword:fun-defs . t)
     (ess-R-fl-keyword:keywords . t)
     (ess-R-fl-keyword:assign-ops . t)
     (ess-R-fl-keyword:constants . t)
     (ess-fl-keyword:fun-calls . t)
     (ess-fl-keyword:numbers . t)
     (ess-fl-keyword:operators . t)
     (ess-fl-keyword:delimiters . t)
     (ess-fl-keyword:= . t)
     (ess-R-fl-keyword:F&T . t)
     (ess-R-fl-keyword:%op% . t)))
 '(package-selected-packages
   '(cape corfu casual-agenda casual embark-consult embark yaml-mode yaml ws-butler web-mode vertico use
(custom-set-faces
 ;; custom-set-faces was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(default ((t (:family "Hack" :height 114 :weight light))))
 '(vterm-color-blue ((t (:foreground "#477EFC" :background "#477EFC")))))
```

3. Public configuration

```
(setq large-file-warning-threshold most-positive-fixnum) ; disable large file warning
(setq-default cache-long-scans nil)

; ---   General   --- ;
; ------------------- ;

(setq frame-background-mode 'dark)
(setq inhibit-splash-screen t)

; ---   Windows   --- ;
; ------------------- ;

;; Remove bars:
(menu-bar-mode -1)
(tool-bar-mode -1)
(scroll-bar-mode -1)
;
; Fringe- Set finge color to background
;https://emacs.stackexchange.com/a/31944/11502
(set-face-attribute 'fringe nil :background nil)

; ---   Lines   --- ;
; ----------------- ;

;;
;; Enable visual line mode
(global-visual-line-mode 1)
;;
;; Line highlighting in all buffers
```

```
(global-hl-line-mode t)
;;
;; Line numbers
(global-display-line-numbers-mode 0)
;;;
;;; Disable line numbers by buffer
(dolist (mode '(org-mode-hook
                term-mode-hook
                shell-mode-hook
                eshell-mode-hook))
  (add-hook mode (lambda () (display-line-numbers-mode 0))))
;;
(setq-default indicate-empty-lines t)

; Do not wrap lines, but extend them off screen
(setq default-truncate-lines nil)

;; no line numbers
(setq global-linum-mode nil)

; ---   Syntax Highlighting   --- ;
; ------------------------------- ;

;; When enabled, any matching parenthesis is highlighted
(show-paren-mode)
;;
;; Enables highlighting of the region whenever the mark is active
(transient-mark-mode 1)

; ---   Code   --- ;
; ---------------- ;

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

; ---   Faces   --- ;
; ----------------- ;

;; ?Fix broken face inheritance
(let ((faces (face-list)))
  (dolist (face faces)
    (let ((inh (face-attribute face :inherit)))
      (when (not (memq inh faces))
        (set-face-attribute face nil :inherit nil)))))

; ---   Text   --- ;
; ---------------- ;

;https://emacs.stackexchange.com/questions/72483/how-to-define-consult-faces-generically-for-minibuffer
(global-hl-line-mode 1)
(set-face-attribute 'highlight nil :background "#294F6E")

(setq tramp-default-method "ssh")


(defadvice tramp-completion-handle-file-name-all-completions
  (around dotemacs-completion-docker activate)
  "(tramp-completion-handle-file-name-all-completions \"\" \"/docker:\" returns
    a list of active Docker container names, followed by colons."
  (if (equal (ad-get-arg 1) "/docker:")
      (let* ((dockernames-raw (shell-command-to-string "docker ps | awk '$NF != \"NAMES\" { print $NF \
             (dockernames (cl-remove-if-not
                           #'(lambda (dockerline) (string-match ":$" dockerline))
                           (split-string dockernames-raw "\n"))))
        (setq ad-return-value dockernames))
    ad-do-it))

; https://emacs.stackexchange.com/questions/29286/tramp-unable-to-open-some-files
(setq tramp-copy-size-limit 10000000)

; ASCII Arrows

; ---   ASCII Arrows   --- ;
; ------------------------ ;

(global-set-key (kbd "C-<right>") (lambda () (interactive) (insert "\u2192")))
(global-set-key (kbd "C-<up>") (lambda () (interactive) (insert "\u2191")))

; ---   Disable Keys   --- ;
; ------------------------ ;

;; Minimize
(global-unset-key (kbd "C-z"))
;; Print
(global-unset-key (kbd "s-p"))

(global-set-key (kbd "C-S-n")
                (lambda () (interactive) (next-line 10)))
(global-set-key (kbd "C-S-p")
                (lambda () (interactive) (next-line -10)))

;; Shorthand for save all buffers
;;  https://stackoverflow.com/questions/15254414/how-to-silently-save-all-buffers-in-emacs
(defun save-all ()
  (interactive)
  (save-some-buffers t))


; ---   Saving And Backup   --- ;
; ----------------------------- ;

; Delete trailing whitespace on save
(add-hook 'before-save-hook
          'delete-trailing-whitespace)

;; Backup process upon save

(setq vc-make-backup-files t) ; Allow old versions to be saved
(setq delete-old-versions 20) ; Save 20
(setq backup-directory-alist '(("." . "~/.emacs.d/backup-save-list"))) ; Save them here

(setq auto-save-visited-mode t) ; Visited files will be auto-saved


(setq auto-save-file-name-transforms
      `((".*" ,(concat user-emacs-directory "auto-save-list/") t)))

; ---   Miscellaneous   --- ;
; ------------------------- ;

;https://emacs.stackexchange.com/questions/62419/what-is-causing-emacs-remote-shell-to-be-slow-on-compl
(defun my-shell-mode-setup-function ()
  (when (and (fboundp 'company-mode)
             (file-remote-p default-directory))
    (company-mode -1)))

(add-hook 'shell-mode-hook 'my-shell-mode-setup-function)

;; delete the region when typing, just like as we expect nowadays.
(delete-selection-mode t)

(setq explicit-shell-file-name "/bin/bash")

;; Don't count two spaces after a period as the end of a sentence.
(setq sentence-end-double-space nil)

;; don't check package signatures
;;  https://emacs.stackexchange.com/questions/233/how-to-proceed-on-package-el-signature-check-failure
(setq package-check-signature nil)

;; Avoid nesting exceeds max-lisp-eval-depth error
;;  https://stackoverflow.com/questions/11807128/emacs-nesting-exceeds-max-lisp-eval-depth
(setq max-lisp-eval-depth 1200)

;; allow remembering risky variables
;;  https://emacs.stackexchange.com/questions/10983/remember-permission-to-execute-risky-local-variable
(defun risky-local-variable-p (sym &optional _ignored) nil)

; Disable "buffer is read only" warning
;;https://emacs.stackexchange.com/questions/19742/is-there-a-way-to-disable-the-buffer-is-read-only-war
(defun my-command-error-function (data context caller)
  "Ignore the buffer-read-only signal; pass the rest to the default handler."
  (when (not (eq (car data) 'buffer-read-only))
    (command-error-default-function data context caller)))

(setq command-error-function #'my-command-error-function)

; Follow symlinks in dired
;;https://emacs.stackexchange.com/questions/41286/follow-symlinked-directories-in-dired
(setq find-file-visit-truename t)

(setq browse-url-browser-function 'browse-url-generic
      browse-url-generic-program "/usr/bin/brave-browser")

; y or n instead of yes or no
(setopt use-short-answers t)

;; don't check package signatures
;;  https://emacs.stackexchange.com/questions/233/how-to-proceed-on-package-el-signature-check-failure
(setq package-check-signature nil)

;; Avoid nesting exceeds max-lisp-eval-depth error
;;  https://stackoverflow.com/questions/11807128/emacs-nesting-exceeds-max-lisp-eval-depth
```

```
(setq max-lisp-eval-depth 1200)
;;

;; normal c-c in ansi-term
;; https://emacs.stackexchange.com/questions/32491/normal-c-c-in-ansi-term
(eval-after-load "term"
  '(progn (term-set-escape-char ?\C-c)
          (define-key term-raw-map (kbd "C-c") nil)))

(setq comint-scroll-to-bottom-on-output t)

;; allow kill hidden part of line
;;  https://stackoverflow.com/questions/3281581/how-to-word-wrap-in-emacs
(setq-default word-wrap t)

;; auto-refresh if source changes
;;  https://stackoverflow.com/questions/1480572/how-to-have-emacs-auto-refresh-all-buffers-when-files-h
(global-auto-revert-mode 1)

; ---   Frames And Windows   --- ;
; ------------------------------ ;

(setq truncate-partial-width-windows nil)
(setq split-window-preferred-function (quote split-window-sensibly))

; ---   Other   --- ;
; ----------------- ;

(setq require-final-newline nil)

(defun toggle-theme ()
  "Toggle between dark and light themes."
  (interactive)
  (if (custom-theme-enabled-p 'manoj-dark)
      (progn
        (disable-theme 'manoj-dark)
        (load-theme 'leuven t))
    (progn
      (disable-theme 'leuven)
      (load-theme 'manoj-dark t))))

(setq create-lockfiles nil)

(defun open-texdoc-in-background (docname)
  "Open a TEXDOC for DOCNAME in the background and close the terminal."
  (interactive "sEnter the name of the document: ")
  (let ((term-buffer (ansi-term "/bin/bash")))
    (with-current-buffer term-buffer
      (term-send-raw-string (concat "texdoc " docname "\n"))
      (term-send-raw-string "sleep 2; exit\n")
      (set-process-sentinel
       (get-buffer-process term-buffer)
       (lambda (process signal)
         (when (or (string= signal "finished\n")
                   (string= signal "exited\n"))
           (kill-buffer (process-buffer process)))))
      (bury-buffer))))
```

```
(defun org-mark-readonly ()
  (interactive)
  (let ((buf-mod (buffer-modified-p)))
    (org-map-entries
     (lambda ()
       (org-mark-subtree)
       (add-text-properties (region-beginning) (region-end) '(read-only t)))
     "read_only")
    (unless buf-mod
      (set-buffer-modified-p nil))))


(defun org-remove-readonly ()
  (interactive)
  (let ((buf-mod (buffer-modified-p)))
    (org-map-entries
     (lambda ()
       (let* ((inhibit-read-only t))
     (org-mark-subtree)
     (remove-text-properties (region-beginning) (region-end) '(read-only t))))
     "read_only")
    (unless buf-mod
      (set-buffer-modified-p nil))))

(add-hook 'org-mode-hook 'org-mark-readonly)

(defun make-region-read-only (start end)
  (interactive "*r")
  (let ((inhibit-read-only t))
    (put-text-property start end 'read-only t)
    (put-text-property start end 'font-lock-face '(:background "#8B0000"))))

(defun make-region-read-write (start end)
  (interactive "*r")
  (let ((inhibit-read-only t))
    (put-text-property start end 'read-only nil)
    (remove-text-properties start end '(font-lock-face nil))))

(defun dont-ask-to-kill-shell-buffer ()
  "Don't ask for confirmation when killing *shell* buffer."
  (let ((buffer-name (buffer-name)))
    (when (string-equal buffer-name "*shell*")
      (setq kill-buffer-query-functions
            (delq 'process-kill-buffer-query-function
                  kill-buffer-query-functions)))))

(add-hook 'shell-mode-hook 'dont-ask-to-kill-shell-buffer)

(cua-mode t)

(defun remove-blank-lines ()
  "Remove all blank lines (including lines with only whitespace) in the current buffer."
  (interactive)
  (save-excursion
    (goto-char (point-min))
    (flush-lines "^[[:space:]]*$")))

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

```
(setq org-startup-with-inline-images t)

        (setq
         org-tags-exclude-from-inheritance
         (list
          "alert"
          "biotool"
          "biopipe"
          "bimonthly"
          "block"
          "blk"
          "flat"
          "hierarchy"
          "include"
          "semimonthly"
          "purpose"
          "midGoal"
          "nearGoal"
          "focus"
          "project"
          "daily"
          "dinner"
          "kit"
          "maint"
          "manuscript"
          "mod"
          "monthly"
          "poster"
          "present"
          "prog"
          "report"
          "routine"
          "soln"
          "weekly"
          "write"
          "sci_rep"
          "stretch"
          "study"))

(setq org-todo-keyword-faces
      (quote (("TODO" :background "red")
              ("NEXT" :foreground "black" :background "yellow"))))

;; keep TODO state timestamps in drawer
(setq org-log-into-drawer t)

;; add done timestamp
(setq org-log-done 'time)

;; enforce dependencies
(setq org-enforce-todo-dependencies t)

;; priority levels
(setq org-highest-priority 65)
(setq org-lowest-priority 89)
(setq org-default-priority 89)

(setq org-link-frame-setup
      '((vm . vm-visit-folder)
        (vm-imap . vm-visit-imap-folder)
        (gnus . org-gnus-no-new-news)
        (file . find-file)  ;; Open files in the same frame
        (wl . wl)))

(setq org-cycle-include-plain-lists 'integrate)
(setq org-list-indent-offset 0)

;; https://emacs.stackexchange.com/questions/22210/auto-update-org-tables-before-each-export
(add-hook 'before-save-hook 'org-table-recalculate-buffer-tables)

(setq org-startup-align-all-tables t)
(setq org-startup-shrink-all-tables t)

(require 'org-collector)

(defun endless/follow-tag-link (tag)
  "Display a list of TODO headlines with tag TAG.
With prefix argument, also display headlines without a TODO keyword."
  (org-tags-view current-prefix-arg tag))

(org-add-link-type
 "tag" 'endless/follow-tag-link)

;(setq process-connection-type nil) this breaks *shell*
(setq org-file-apps
      '((directory . "/usr/bin/gnome-terminal --working-directory=\"%s\"")
        ("\\.pdf\\'" . "setsid -w xdg-open \"%s\"")
        ("\\.svg\\'" . "setsid -w xdg-open \"%s\"")
        ("\\.yaml\\'" . "emacsclient -c \"%s\"")
        ("\\.list\\'" . emacsclient)
        (auto-mode . emacsclient)
        (t . "setsid -w xdg-open \"%s\"")
        ))

(setq org-image-actual-width '(300))

(defun shk-fix-inline-images ()
  (when org-inline-image-overlays
    (org-redisplay-inline-images)))

(with-eval-after-load 'org
  (add-hook 'org-babel-after-execute-hook 'shk-fix-inline-images))

(org-babel-do-load-languages
 'org-babel-load-languages
 '(
   (ditaa . t)
   (dot .t)
   (emacs-lisp . t)
   (latex . t)
   (mermaid .t)
   (org . t)
   (python . t)
   (R . t)
   (shell . t)
   (sql .t)
   (sqlite . t)
   ))

(require 'ob-shell)
(require 'yaml-mode)

(defun org-babel-execute:yaml (body params)
  "Execute a block of YAML code with org-babel."
  (let ((temp-file (org-babel-temp-file "yaml-")))
    (with-temp-file temp-file
      (insert body))
    (org-babel-eval (format "cat %s" temp-file) "")))

(add-to-list 'org-src-lang-modes '("yaml" . yaml))

(defun org-remove-properties-drawer ()
  "Remove PROPERTIES drawer from tangled files."
  (save-excursion
    (goto-char (point-min))
    (while (re-search-forward "^# :PROPERTIES:\n\\(?:# .*\n\\)*?# :END:\n" nil t)
      (replace-match "")))
  (save-buffer)
  )

(add-hook 'org-babel-post-tangle-hook 'org-remove-properties-drawer)

(setq org-babel-default-header-args '((:results . "silent")
                                      (:eval . "no-export")
                                      (:exports . "none")
                                      (:tangle . "yes")
                                      (:cache . "yes")
                                      (:noweb . "yes")
                                      (:post-tangle . org-remove-properties-drawer)))

(setq
 ;; Blocks inserted directly without additional formatting
 org-babel-inline-result-wrap "%s"
 ;;
 ;; Preserve language-specific indentation, aligns left
 org-src-preserve-indentation t
 ;;
 ;; Tab works like in major mode of lanuauge
 org-src-tab-acts-natively t
 ;;
 org-babel-python-command "python3"
 ;;
 org-confirm-babel-evaluate nil
 ;;
 org-src-fontify-natively t
 ;;
 ;; Open src windows in current frames
 org-src-window-setup 'current-window)

;; disable confrmation for elisp execution of org src blocks
(setq safe-local-variable-values '((org-confirm-elisp-link-function . nil)))

(setq org-hide-block-startup t)
```

```
(defvar org-blocks-hidden nil)

(defun org-toggle-blocks ()
  (interactive)
  (if org-blocks-hidden
      (org-show-block-all)
    (org-hide-block-all))
  (setq-local org-blocks-hidden (not org-blocks-hidden)))

(setq org-babel-min-lines-for-block-output 1000)

(setq org-babel-noweb-wrap-start "<#"
      org-babel-noweb-wrap-end "#>")

(defun my-org-remove-properties-drawer ()
  "Remove PROPERTIES drawer from tangled files without triggering buffer modification warning."
  (let ((buffer-modified-p (not (buffer-modified-p))))
    (save-excursion
      (goto-char (point-min))
      (while (re-search-forward "^# :PROPERTIES:\n\\(?:# .*\n\\)*?# :END:\n" nil t)
        (replace-match "")))
    (unless buffer-modified-p
      (set-buffer-modified-p nil))))

(advice-add 'org-remove-properties-drawer :override #'my-org-remove-properties-drawer)

(setq org-show-hierarchy-above t)

(setq org-fold-show-context-detail
      '((default . tree)))

(setq
 org-show-context-detail
 '((agenda . ancestors)
   (bookmark-jump . ancestors)
   (isearch . ancestors)
   (default . ancestors))
)

(defun org-hide-all-src-blocks ()
  "Hide all source blocks in the current Org buffer."
  (interactive)
  (org-babel-map-src-blocks nil
    (save-excursion
      (goto-char (org-babel-where-is-src-block-head))
      (org-hide-block-toggle t))))



(defun my-collapse-all-drawers (&optional arg)
  (interactive "P")  ;; "P" means that the function accepts a prefix argument and passes it as ARG
  (org-hide-drawer-all)
  (when arg  ;; When ARG is non-nil (when called with C-u), execute `org-cycle-global`.
    (org-cycle-global)
    (org-hide-all-src-blocks)
    (beginning-of-buffer))
  )
```

```
(global-set-key (kbd "C-c d") 'my-collapse-all-drawers)
;; You might want to remove the hook if you don't want this function to run every time you open an org
(add-hook 'org-mode-hook 'my-collapse-all-drawers)

(setq org-cycle-separator-lines 0)
(setq yas-indent-line 'fixed)

(defun my-remove-trailing-newlines-in-tangled-blocks ()
  "Remove trailing newlines from tangled block bodies."
  (save-excursion
    (goto-char (point-max))
    (when (looking-back "\n" nil)
      (delete-char -1))))

(add-hook 'org-babel-post-tangle-hook #'my-remove-trailing-newlines-in-tangled-blocks)

(setq org-confirm-shell-link-function nil)

(with-eval-after-load 'org
        (add-to-list 'org-modules 'org-habit))

;; Clock times in hours and minutes
;;  (see https://stackoverflow.com/questions/22720526/set-clock-table-duration-format-for-emacs-org-mod
(setq org-time-clocksum-format
      '(:hours "%d" :require-hours t :minutes ":%02d" :require-minutes t))
(setq org-duration-format (quote h:mm))

(setq org-catch-invisible-edits 'error)
(global-set-key (kbd "C-c l") 'org-store-link)
(setq org-refile-targets '((org-agenda-files :maxlevel . 14)))
(setq org-indirect-buffer-display 'current-window)

(setq org-id-link-to-org-use-id 'use-existing)
;;https://stackoverflow.com/questions/28351465/emacs-orgmode-do-not-insert-line-between-headers


(setq org-enforce-todo-checkbox-dependencies t)
;; don't adapt indentation to header level
(setq org-adapt-indentation nil)

(setq org-support-shift-select t)
(setq org-src-window-setup 'current-window)
(setq org-export-async-debug nil)

;; ensures that any file with the .org extension will automatically open in org-mode
(add-to-list 'auto-mode-alist '("\\.org\\'" . org-mode))

;; Make heading regex include tags
(setq org-heading-regexp "^[[:space:]]*\\(\\*+\\)\\(?: +\\(.*?\\)\\)?[ \t]*\\(:[[:alnum:]_@#%:]+:\\)?[ 

(setq org-blank-before-new-entry '((heading . nil) (plain-list-item . nil)))

(defun my-org-tree-to-indirect-buffer (&optional arg)
  "Open current org tree in indirect buffer, using one prefix argument.
When called with two prefix arguments, ARG, run the original function without prefix argument."
  (interactive "P")
  (if (equal arg '(16)) ; 'C-u C-u' produces (16)
      (org-tree-to-indirect-buffer nil) ; original behavior
    (org-tree-to-indirect-buffer t)) ; one prefix argument
  (my-collapse-all-drawers))
(define-key org-mode-map (kbd "C-c C-x b") 'my-org-tree-to-indirect-buffer)

;; the below as nil fucks of export of inline code
(setq org-export-babel-evaluate t)
;; https://emacs.stackexchange.com/questions/23982/cleanup-org-mode-export-intermediary-file/24000#2400


(setq-default cache-long-scans nil)
(setq org-export-with-broken-links t)
(setq org-export-allow-bind-keywords t)

(setq org-export-with-sub-superscripts nil
      org-export-headline-levels 2
      org-export-with-toc nil
      org-export-with-section-numbers nil
      org-export-with-tags nil
      org-export-with-todo-keywords nil)

(setq org-odt-preferred-output-format "docx")

(require 'ox-latex)

(customize-set-value 'org-latex-with-hyperref nil)

(setq org-latex-logfiles-extensions (quote ("auto" "lof" "lot" "tex~" "aux" "idx" "log" "out" "toc" "na

(add-to-list 'org-latex-packages-alist '("" "listings"))
(add-to-list 'org-latex-packages-alist '("" "color"))
(setq org-latex-caption-above nil)

(setq org-latex-remove-logfiles t)

(add-to-list 'org-latex-packages-alist '("" "listingsutf8"))
(setq org-latex-src-block-backend 'minted)

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

(with-eval-after-load 'ox-latex
  (add-to-list 'org-latex-classes '("empty"
                                    "\\documentclass{article}
\\newcommand\\foo{bar}
[NO-DEFAULT-PACKAGES]
[NO-PACKAGES]"
                                    ("\\section{%s}" . "\\section*{%s}")
                                    ("\\subsection{%s}" . "\\subsection*{%s}")
                                    ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                                    ("\\paragraph{%s}" . "\\paragraph*{%s}")
                                    ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))

(defun my-org-open-in-brave-new-window ()
  "Open the link at point in Brave browser in a new window."
  (interactive)
  (let* ((context (org-element-context))
         (link (and (eq (org-element-type context) 'link)
                    (org-element-property :raw-link context))))
    (if link
        (start-process "brave-new-window" nil "brave-browser" "--new-window" link)
      (message "No link at point"))))

(defun my-org-open-at-point (&optional arg)
  "Open the link at point, using a new window in Brave if ARG is non-nil."
  (interactive "P")
  (if arg
      (my-org-open-in-brave-new-window)
    (org-open-at-point)))

;; Rebind C-c C-o in org mode to our custom function
(define-key org-mode-map (kbd "C-c C-o") 'my-org-open-at-point)

;; (defun org-toggle-checkbox-and-children ()
;;   "Toggle checkbox and all children checkboxes."
;;   (interactive)
;;   (save-excursion
;;     (let* ((parent-indent (current-indentation))
;;            (end (save-excursion
;;                   (org-end-of-subtree)
;;                   (point)))
;;            (current-checkbox (save-excursion
;;                                (beginning-of-line)
;;                                (when (re-search-forward "\\[[ X-]\\]" (line-end-position) t)
;;                                  (match-string 0))))
;;            (new-state (if (equal current-checkbox "[ ]") "[X]" "[ ]")))
;;       ;; Toggle the parent checkbox
;;       (beginning-of-line)
;;       (when (re-search-forward "\\[[ X-]\\]" (line-end-position) t)
;;         (replace-match new-state))
;;       ;; Toggle all children checkboxes
;;       (forward-line)
;;       (while (< (point) end)
;;         (let ((line-start (point)))
;;           (when (and (> (current-indentation) parent-indent)
;;                      (re-search-forward "\\[[ X-]\\]" (line-end-position) t))
;;             (replace-match new-state)
;;             (goto-char line-start)))
;;         (forward-line 1)))))

;; ;; Bind it to a convenient key
;; (define-key org-mode-map (kbd "C-c x") 'org-toggle-checkbox-and-children)

(with-eval-after-load 'ox-latex
  (add-to-list 'org-latex-classes
               '("documentation"
                 "\\documentclass{article}
                 \\usepackage{/home/jeszyman/repos/latex/sty/documentation}
                 [NO-DEFAULT-PACKAGES]
                 [NO-PACKAGES]
                 [EXTRA]
                 \\tableofcontents"
                 ("\\section{%s}" . "\\section{%s}")
                 ("\\subsection{%s}" . "\\subsection{%s}")
```

```
                ("\\subsubsection{%s}" . "\\subsubsection{%s}")
                ("\\paragraph{%s}" . "\\paragraph{%s}")
                ("\\subparagraph{%s}" . "\\subparagraph{%s}"))))

(with-eval-after-load 'ox-latex
  (add-to-list 'org-latex-classes
               '("documentation"
                 "\\documentclass{article}
                 \\usepackage{/home/jeszyman/repos/latex/sty/documentation}
                 [NO-DEFAULT-PACKAGES]
                 [NO-PACKAGES]
                 [EXTRA]
                 \\begin{document}
                 \\tableofcontents
                 \\vspace{1cm}"
                 ("\\section{%s}" . "\\section{%s}")
                 ("\\subsection{%s}" . "\\subsection{%s}")
                 ("\\subsubsection{%s}" . "\\subsubsection{%s}")
                 ("\\paragraph{%s}" . "\\paragraph{%s}")
                 ("\\subparagraph{%s}" . "\\subparagraph{%s}"))))

(setq org-icalendar-with-timestamps 'active)
(setq org-icalendar-use-scheduled t)
(setq org-icalendar-use-deadline nil)
(setq org-icalendar-include-todo t)
(setq org-icalendar-exclude-tags (list "noexport"))
(setq org-icalendar-include-body '1)
(setq org-icalendar-alarm-time '5)
(setq org-icalendar-store-UID t) ;;Required for syncs
(setq org-icalendar-timezone "America/Chicago")
(setq org-agenda-default-appointment-duration 30)
(setq org-icalendar-combined-agenda-file "/tmp/org.ics")

(setq org-use-property-inheritance t)

(defun browse-org-table-urls-by-name (table-name)
  "Browse URLs listed in an Org-mode table identified by TABLE-NAME.
TABLE-NAME is the name of the table identified as #+name."

  (interactive "sEnter table name: ")
  (let* ((element (ignore-errors
                    (org-element-map (org-element-parse-buffer) 'table
                      (lambda (el)
                        (when (string= (org-element-property :name el) table-name)
                          el))
                      nil t))))
    (if (not element)
        (message "Table with name %s not found" table-name)
      (let ((table-begin (org-element-property :contents-begin element))
            (table-end (org-element-property :contents-end element)))
        (if (or (null table-begin) (null table-end))
            (message "No contents found for the table with name %s" table-name)
          (let ((table-content (buffer-substring-no-properties table-begin table-end)))
            (with-temp-buffer
              (insert table-content)
              (goto-char (point-min))
              (let ((urls (org-table-to-lisp)))
                (if (not urls)
                    (message "No URLs found in the table with name %s" table-name)
                  (let ((first-url (car (car urls))))
                    (start-process "brave-browser" nil "brave-browser" "--new-window" first-url)
                    (sit-for 2)  ;; Wait for the new window to open
                    (dolist (url-row (cdr urls))
                      (start-process "brave-browser" nil "brave-browser" (car url-row))
                      (sit-for 0.5)))  ;; Delay between each URL
                  (message "Opened URLs from table with name %s" table))))))))))

(setq org-agenda-repeating-timestamp-show-all nil)
(setq org-sort-agenda-notime-is-late nil)
(setq org-agenda-start-on-weekday nil)
(setq org-agenda-remove-tags t)
(setq org-agenda-skip-scheduled-if-done t)
(setq
 org-agenda-files
 (list "~/repos/org/"))

(define-key global-map "\C-ca" 'org-agenda)
(setq org-agenda-skip-unavailable-files t)

(setq org-agenda-use-tag-inheritance t)
;;  http://stackoverflow.com/questions/36873727/make-org-agenda-full-screen
(setq org-agenda-window-setup (quote only-window))
(setq org-agenda-todo-ignore-time-comparison-use-seconds t)

;;; Based on http://article.gmane.org/gmane.emacs.orgmode/41427
  (defun my-skip-tag(tag)
    "Skip entries that are tagged TAG"
    (let* ((entry-tags (org-get-tags-at (point))))
      (if (member tag entry-tags)
          (progn (outline-next-heading) (point))
        nil)))

;; Needed for no y/n prompt at linked agenda execution
(setq org-confirm-elisp-link-function nil)

;; https://emacs.stackexchange.com/questions/19742/is-there-a-way-to-disable-the-buffer-is-read-only-wa
(defun my-command-error-function (data context caller)
  "Ignore the buffer-read-only signal; pass the rest to the default handler."
  (when (not (eq (car data) 'buffer-read-only))
    (command-error-default-function data context caller)))

(setq command-error-function #'my-command-error-function)

(defun org-plain-follow (id _)
  "Follow a plain link as if it were an ID link."
  (org-id-open id nil))

(org-link-set-parameters "plain"
                         :follow #'org-plain-follow
                         :export #'org-plain-export)

(defun org-plain-export (link description format _)
  "Export a plain link. Always export as plain text."
  (cond
   ((eq format 'html) (or description link))
   ((eq format 'latex) (or description link))
   ((eq format 'ascii) (or description link))
   (t link)))

(provide 'ol-plain)

(with-eval-after-load 'org
  (require 'ol-plain))

(defun shk-fix-inline-images ()
  (when org-inline-image-overlays
    (org-redisplay-inline-images)))

(with-eval-after-load 'org
  (add-hook 'org-babel-after-execute-hook 'shk-fix-inline-images))

(setq org-image-actual-width '(300))

;; https://emacs.stackexchange.com/questions/19742/is-there-a-way-to-disable-the-buffer-is-read-only-wa
(defun custom-command-error-function (data context caller)
  "Ignore the buffer-read-only signal; pass the rest to the default handler."
  (when (not (eq (car data) 'buffer-read-only))
    (command-error-default-function data context caller)))

(setq command-error-function #'my-command-error-function)

;;https://emacs.stackexchange.com/questions/12701/kill-a-line-deletes-the-line-but-leaves-a-blank-newli
(setq kill-whole-line t)

(setq reftex-default-bibliography '("~/repos/org/bib.bib"))

;; see org-ref for use of these variables

(setq bibtex-completion-bibliography "~/repos/org/bib.bib"
      bibtex-completion-library-path "~/library"
      bibtex-completion-notes-path "~/repo/org/notes")

;; Amazing bibtex from doi fetcher
;; https://www.anghyflawn.net/blog/2014/emacs-give-a-doi-get-a-bibtex-entry/
(defun get-bibtex-from-doi (doi)
 "Get a BibTeX entry from the DOI"
 (interactive "MDOI: ")
 (let ((url-mime-accept-string "text/bibliography;style=bibtex"))
   (with-current-buffer
     (url-retrieve-synchronously
       (format "http://dx.doi.org/%s"
               (replace-regexp-in-string "http://dx.doi.org/" "" doi)))
     (switch-to-buffer (current-buffer))
     (goto-char (point-max))
     (setq bibtex-entry
               (buffer-substring
                  (string-match "@" (buffer-string))
              (point)))
     (kill-buffer (current-buffer))))
 (insert (decode-coding-string bibtex-entry 'utf-8))
 (bibtex-fill-entry))

(setq python-shell-completion-native-enable nil)
```

```elisp
(add-hook 'python-mode-hook
  (lambda () (setq indent-tabs-mode nil)))

(setq python-indent-guess-indent-offset-verbose nil)

(set-buffer-file-coding-system 'utf-8)
(prefer-coding-system 'utf-8)
(set-language-environment "UTF-8")

(global-set-key (kbd "C-x a") (lambda () (interactive) (insert " ")))

(defun open-chatgpt-query-in-new-browser-window (query &optional use-gpt-4)
  "Send a QUERY to ChatGPT and open the result in a new browser window.
With a prefix argument USE-GPT-4, use GPT-4 instead of GPT-4-turbo."
  (interactive "sEnter your ChatGPT query: \nP")
  (let* ((model (if use-gpt-4 "gpt-4" "gpt-4-turbo"))
         (url (concat "https://chat.openai.com/?q=" (url-hexify-string query)
                      "&model=" model)))
    (start-process "brave-browser" nil "brave-browser" "--new-window" url)))

(global-set-key (kbd "C-c C-g") 'open-chatgpt-query-in-new-browser-window)

  ;; https://emacs.stackexchange.com/questions/35069/best-way-to-select-a-word
  (defun mark-whole-word (&optional arg allow-extend)
    "Like `mark-word', but selects whole words and skips over whitespace.
  If you use a negative prefix arg then select words backward.
  Otherwise select them forward.

  If cursor starts in the middle of word then select that whole word.

  If there is whitespace between the initial cursor position and the
  first word (in the selection direction), it is skipped (not selected).

  If the command is repeated or the mark is active, select the next NUM
  words, where NUM is the numeric prefix argument.  (Negative NUM
  selects backward.)"
    (interactive "P\np")
    (let ((num  (prefix-numeric-value arg)))
      (unless (eq last-command this-command)
        (if (natnump num)
            (skip-syntax-forward "\\s-")
          (skip-syntax-backward "\\s-")))
      (unless (or (eq last-command this-command)
                  (if (natnump num)
                      (looking-at "\\b")
                    (looking-back "\\b")))
        (if (natnump num)
            (left-word)
          (right-word)))
      (mark-word arg allow-extend)))

  (global-set-key (kbd "C-c C-SPC") 'mark-whole-word)

(use-package tex
  :ensure auctex
  :config
  (setenv "PATH" (concat "/usr/local/texlive/2021/bin/x86_64-linux:"
                         (getenv "PATH")))
  (add-to-list 'exec-path "/usr/local/texlive/2021/bin/x86_64-linux")
  (setq
   TeX-auto-save t
   TeX-parse-self t
   TeX-save-query nil
   TeX-view-program-selection
   '(((output-dvi has-no-display-manager) . "dvi2tty")
     ((output-dvi style-pstricks) . "dvips and gv")
     (output-pdf . "Okular")
     (output-dvi . "xdvi")
     (output-pdf . "Evince")
     (output-html . "xdg-open"))))

(use-package blacken
  :after elpy
  :hook (elpy-mode . blacken-mode))

;; (use-package company
;;   :config
;;   (global-company-mode)
;;   (setq
;;    company-dabbrev-downcase nil)) ; Don't downcase by default

(setenv "WORKON_HOME" "~/miniconda3/envs")

(use-package eglot
  :ensure t
  :init
  (add-hook 'sh-mode-hook 'eglot-ensure)
  (add-hook 'ess-r-mode-hook 'eglot-ensure)
  (add-hook 'python-mode-hook 'eglot-ensure)
  :config
  (add-to-list 'eglot-server-programs '(sh-mode . ("bash-language-server" "start")))
  (add-to-list 'eglot-server-programs '(python-mode . ("pylsp")))
  (add-to-list 'eglot-server-programs '(ess-r-mode . ("R" "--slave" "-e" "languageserver::run()"))))

(with-eval-after-load 'eglot
  (define-key eglot-mode-map (kbd "C-c <tab>") #'company-complete))

(require 'essh)
(defun essh-sh-hook ()
  (define-key sh-mode-map "\C-c\C-r" 'pipe-region-to-shell)
  (define-key sh-mode-map "\C-c\C-b" 'pipe-buffer-to-shell)
  (define-key sh-mode-map "\C-c\C-j" 'pipe-line-to-shell)
  (define-key sh-mode-map "\C-c\C-n" 'pipe-line-to-shell-and-step)
  (define-key sh-mode-map "\C-c\C-f" 'pipe-function-to-shell)
  (define-key sh-mode-map "\C-c\C-d" 'shell-cd-current-directory))
(add-hook 'sh-mode-hook 'essh-sh-hook)

(add-hook 'sh-mode-hook 'flycheck-mode)

(use-package ess
  :init
  (require 'ess-site)
  :config
  (setq ess-ask-for-ess-directory nil
        ess-help-own-frame 'one
        ess-indent-with-fancy-comments nil
        ess-use-auto-complete t
        ess-use-company t
        inferior-ess-own-frame t
        inferior-ess-same-window nil)
  (define-key ess-mode-map (kbd "C-c C-n") 'ess-eval-line-and-step)
  :mode (
         ("/R/.*\\.q\\'"        . R-mode)
         ("\\.[rR]\\'"          . R-mode)
         ("\\.[rR]profile\\'"   . R-mode)
         ("NAMESPACE\\'"        . R-mode)
         ("CITATION\\'"         . R-mode)
         ("\\.[Rr]out"          . R-transcript-mode)
         ("\\.Rd\\'"            . Rd-mode)
         ))

(custom-set-variables
 '(ess-R-font-lock-keywords
   (quote
    ((ess-R-fl-keyword:modifiers . t)
     (ess-R-fl-keyword:fun-defs . t)
     (ess-R-fl-keyword:keywords . t)
     (ess-R-fl-keyword:assign-ops . t)
     (ess-R-fl-keyword:constants . t)
     (ess-fl-keyword:fun-calls . t)
     (ess-fl-keyword:numbers . t)
     (ess-fl-keyword:operators . t)
     (ess-fl-keyword:delimiters . t)
     (ess-fl-keyword:= . t)
     (ess-R-fl-keyword:F&T . t)
     (ess-R-fl-keyword:%op% . t)))))

(use-package elpy
  :init
  (advice-add 'python-mode :before 'elpy-enable)
  :config
  (define-key elpy-mode-map (kbd "C-c C-n") 'elpy-shell-send-statement-and-step)
  (setenv "PATH" (concat "~/miniconda3/bin:" (getenv "PATH")))
  (setenv "WORKON_HOME" "~/miniconda3/envs")
  (setq exec-path (append '("~/miniconda3/bin") exec-path))
  (add-to-list 'process-coding-system-alist '("python" . (utf-8 . utf-8)))
  (setq elpy-rpc-python-command "~/miniconda3/bin/python")
)

(defun my-elpy-shell-display-buffer-in-new-frame (buffer alist)
  "Display the Python shell buffer in a new frame."
  (let ((display-buffer-alist
         '(("*Python*" display-buffer-pop-up-frame))))
    (display-buffer buffer alist)))

(advice-add 'elpy-shell-send-statement-and-step :around
            (lambda (orig-fun &rest args)
              "Send statement and open the Python buffer in a new frame."
              (let ((display-buffer-alist
                     '(("*Python*" . (my-elpy-shell-display-buffer-in-new-frame)))))
                (apply orig-fun args))))

(use-package exec-path-from-shell
  :config
  (when (daemonp)
    (exec-path-from-shell-initialize)))

(use-package expand-region)
(require 'expand-region)
(global-set-key (kbd "C-=") 'er/expand-region)

(use-package flycheck
  :hook
  (org-src-mode . my-org-mode-flycheck-hook)
  :config
  (defun my-org-mode-flycheck-hook ()
    (when (derived-mode-p 'prog-mode) ;; Check if it's a programming mode
      (flycheck-mode 1))))

(use-package flyspell
  :config
  (setq ispell-personal-dictionary "~/.aspell.en.pws"))

(use-package helm
  :config
  (global-set-key (kbd "C-x b") 'helm-mini)
  (global-set-key (kbd "C-s") 'helm-occur)
  (setq
   helm-completion-style 'emacs
   helm-move-to-line-cycle-in-source nil)) ;; allow C-n through different sections

(use-package helm-org
  :config
  (global-set-key (kbd "C-c j") 'helm-org-in-buffer-headings)
  (global-set-key (kbd "C-c w") 'helm-org-refile-locations)
  (setq org-outline-path-complete-in-steps nil
        org-refile-allow-creating-parent-nodes 'confirm
        org-refile-targets '((org-agenda-files :maxlevel . 20))
        org-refile-targets '((org-agenda-files :maxlevel . 3))
        org-refile-use-outline-path 'file))
  (define-key global-map (kbd "C-c C-j") nil)
  (global-set-key (kbd "C-c C-j") 'helm-org-agenda-files-headings)
  (define-key global-map (kbd "C-$") 'org-mark-ring-goto)
  (global-set-key (kbd "C-c C-j") 'helm-org-agenda-files-headings)
  (setq helm-org-ignore-autosaves t)

(global-set-key (kbd "C-c C-j") 'helm-org-agenda-files-headings)

(with-eval-after-load 'org
  (define-key org-mode-map (kbd "C-c C-j") 'helm-org-agenda-files-headings))

(use-package helm-org-rifle
    :config
    (setq helm-org-rifle-show-path nil
          helm-org-rifle-show-full-contents nil)
    (require 'helm)
    (global-set-key (kbd "C-c C-j") 'helm-org-agenda-files-headings))

(use-package htmlize)

(use-package ivy
  :diminish)
```

```
(use-package marginalia
  ;; Either bind `marginalia-cycle' globally or only in the minibuffer
  :bind (("M-A" . marginalia-cycle)
         :map minibuffer-local-map
         ("M-A" . marginalia-cycle))
  ;; The :init configuration is always executed (Not lazy!)
  :init
  (marginalia-mode)
  :ensure t)

(use-package native-complete)

(use-package orderless
  :init
  ;; Configure a custom style dispatcher (see the Consult wiki)
  ;; (setq orderless-style-dispatchers '(+orderless-dispatch)
  ;;       orderless-component-separator #'orderless-escapable-split-on-space)
  (setq completion-styles '(orderless basic)
        completion-category-defaults nil
        completion-category-overrides '((file (styles partial-completion)))))

(use-package org-edna
  :ensure t
  :config
  (org-edna-mode 1))

(require 'org-checklist)
(require 'ox-extra)
(ox-extras-activate '(ignore-headlines))

(use-package org-ql)

(use-package org-ref
  :init
  (require 'bibtex)
  (require 'org-ref-ivy)
  (require 'org-ref-bibtex)
  (require 'org-ref-pubmed)
  (require 'org-ref-scopus)
  (require 'org-ref-wos)
  :config
  (setq
   org-ref-default-bibliography '("~/repos/org/bib.bib")
   org-ref-pdf-directory "~/library/"))

(setq bibtex-completion-bibliography '("~/repos/org/bib.bib")
      bibtex-completion-library-path '("~/data/library/")
      bibtex-completion-additional-search-fields '(keywords)
      bibtex-completion-display-formats
      '((article       . "${=has-pdf=:1}${=has-note=:1} ${year:4} ${author:36} ${title:*} ${journal:40}"
        (inbook        . "${=has-pdf=:1}${=has-note=:1} ${year:4} ${author:36} ${title:*} Chapter ${chap
        (incollection  . "${=has-pdf=:1}${=has-note=:1} ${year:4} ${author:36} ${title:*} ${booktitle:40
        (inproceedings . "${=has-pdf=:1}${=has-note=:1} ${year:4} ${author:36} ${title:*} ${booktitle:40
        (t             . "${=has-pdf=:1}${=has-note=:1} ${year:4} ${author:36} ${title:*}"))
      bibtex-completion-pdf-open-function
      (lambda (fpath)
        (call-process "open" nil 0 nil fpath)))
(define-key org-mode-map (kbd "C-c ]") 'org-ref-insert-link)

(setq org-ref-show-broken-links nil)

(setq org-ref-bibliography-notes "~/repo/org/notes"
      org-ref-default-bibliography '("~/repos/org/bib.bib")
      org-ref-pdf-directory "~/library")

(require 'org-ref-ivy)
(setq org-ref-insert-link-function 'org-ref-insert-link-hydra/body
      org-ref-insert-cite-function 'org-ref-cite-insert-ivy
      org-ref-insert-label-function 'org-ref-insert-label-link
      org-ref-insert-ref-function 'org-ref-insert-ref-link
      org-ref-cite-onclick-function (lambda (_) (org-ref-citation-hydra/body)))

(defun org-ref-link-message (&optional a1 a2 a3)
  (when (and (eq major-mode 'org-mode)
   (eq (get-text-property (point) 'help-echo) 'org-ref-cite-tooltip))
  (display-local-help)))

(advice-add 'right-char :after 'org-ref-link-message)
(advice-add 'left-char :after 'org-ref-link-message)
(advice-add 'evil-forward-char :after 'org-ref-link-message)
(advice-add 'evil-backward-char :after 'org-ref-link-message)

(use-package ox-pandoc
  :after org
  :config
  (setq org-pandoc-options-for-docx '((standalone . nil)))
  )

(use-package savehist)

(use-package snakemake-mode)

(defcustom snakemake-indent-field-offset nil
  "Offset for field indentation."
  :type 'integer)

(defcustom snakemake-indent-value-offset nil
  "Offset for field values that the line below the field key."
  :type 'integer)

;; Install and configure tree-sitter
(use-package tree-sitter
  :ensure t
         )

;; Install and configure tree-sitter-langs
(use-package tree-sitter-langs
  :ensure t
  :after tree-sitter
  :config
  (add-hook 'tree-sitter-after-on-hook #'tree-sitter-hl-mode))


(global-tree-sitter-mode)
```

```
(add-hook 'tree-sitter-after-on-hook #'tree-sitter-hl-mode)

(defun disable-tree-sitter-for-org-mode ()
  (when (eq major-mode 'org-mode)
    (tree-sitter-mode -1)))

(add-hook 'tree-sitter-mode-hook #'disable-tree-sitter-for-org-mode)

(use-package vertico)

;; Otherwise use the default `completion--in-region' function.
(setq completion-in-region-function
      (lambda (&rest args)
        (apply (if vertico-mode
                   #'consult-completion-in-region
                 #'completion--in-region)
               args)))

;; A few more useful configurations...
(use-package emacs
  :init
  ;; Add prompt indicator to `completing-read-multiple'.
  ;; We display [CRM<separator>], e.g., [CRM,] if the separator is a comma.
  (defun crm-indicator (args)
    (cons (format "[CRM%s] %s"
                  (replace-regexp-in-string
                   "\\`\\[.*?]\\*\\|\\[.*?]\\*\\'" ""
                   crm-separator)
                  (car args))
          (cdr args)))
  (advice-add #'completing-read-multiple :filter-args #'crm-indicator)

  ;; Do not allow the cursor in the minibuffer prompt
  (setq minibuffer-prompt-properties
        '(read-only t cursor-intangible t face 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))

(define-key vertico-map (kbd "TAB") #'minibuffer-complete)
(define-key vertico-map (kbd "C-n") #'vertico-next)
(define-key vertico-map (kbd "C-p") #'vertico-previous)

;; Ensure you have these packages installed
(use-package vertico
  :ensure t
  :init
  (vertico-mode))

(use-package marginalia
  :ensure t
  :after vertico
  :init
  (marginalia-mode))

(use-package orderless
  :ensure t
  :init
  ;; Customize completion styles to include orderless
  (setq completion-styles '(orderless basic))
  ;; Optionally configure completion categories
  (setq completion-category-defaults nil)
  (setq completion-category-overrides '((file (styles basic partial-completion)))))

(use-package savehist
  :ensure t
  :init
  (savehist-mode))

(use-package consult
  :ensure t
  :bind (("C-x b" . consult-buffer)
         ("M-y" . consult-yank-pop)
         ("C-s" . consult-line)
         ("M-g g" . consult-goto-line)
         ("M-g M-g" . consult-goto-line)
         ("C-M-l" . consult-imenu)
         :map minibuffer-local-map
         ("M-r" . consult-history))
  :init
  (setq register-preview-delay 0
        register-preview-function #'consult-register-preview)
  ;; Optionally configure preview
  (autoload 'consult-register-window "consult")
  (setq consult-register-window-function #'consult-register-window)
  ;; Optionally configure narrowing key
  (setq consult-narrow-key "<"))

;; Enable vertico-directory for better directory navigation
(use-package vertico-directory
  :ensure nil
  :load-path "path/to/vertico-directory"
  :after vertico
  :bind (:map vertico-map
              ("RET" . vertico-directory-enter)
              ("DEL" . vertico-directory-delete-char)
              ("M-DEL" . vertico-directory-delete-word)))

;; Example configuration for more intuitive completion cycling
(define-key vertico-map (kbd "TAB") #'minibuffer-complete)
(define-key vertico-map (kbd "C-n") #'vertico-next)
(define-key vertico-map (kbd "C-p") #'vertico-previous)

(use-package vterm
  :bind* (:map vterm-mode-map
               ("C-z" . vterm-undo)
               ("C-v" . vterm-yank))
  :init
  (add-hook 'vterm-mode-hook '(lambda () (setq-local cua-mode nil)))
  :config
  (setq vterm-max-scrollback 100000)
  (custom-set-faces
   '(vterm-color-blue ((t (:foreground "#477EFC" :background "#477EFC")))))

(use-package multi-vterm :ensure t)

(use-package web-mode
  :mode ("\\.phtml\\'"
         "\\.tpl\\.php\\'"
         "\\.[agj]sp\\'"
         "\\.as[cp]x\\'"
         "\\.erb\\'"
         "\\.mustache\\'"
         "\\.djhtml\\'"
         "\\.html?\\'"))

(use-package yasnippet
  :init
  ;; Dynamically add subdirectories in ~/.emacs.d/snippets to yas-snippet-dirs
  (setq yas-snippet-dirs
        (directory-files "~/.emacs.d/snippets" t "^[^.]+")) ; Directories only, no append needed
   :config
  (yas-global-mode 1) ; Enable yasnippet globally
  (define-key yas-minor-mode-map (kbd "<C-tab>") 'yas-expand))

(defun my-org-mode-hook ()
  (setq-local yas-buffer-local-condition
              '(not (org-in-src-block-p t))))
(add-hook 'org-mode-hook #'my-org-mode-hook)

(add-hook 'yas-before-expand-snippet-hook (lambda () (setq-local company-backends nil)))
(add-hook 'yas-after-exit-snippet-hook    (lambda () (kill-local-variable 'company-backends)))

(setq require-final-newline nil)
(defun yas-auto-expand ()
  "Function to allow automatic expansion of snippets which contain a condition, auto."

  (when yas-minor-mode
    (let ((yas-buffer-local-condition ''(require-snippet-condition . auto)))
      (yas-expand))))

(defun my-yas-try-expanding-auto-snippets ()
  (when yas-minor-mode
    (let ((yas-buffer-local-condition ''(require-snippet-condition . auto)))
      (yas-expand))))

(add-hook 'post-command-hook #'my-yas-try-expanding-auto-snippets)
```

4. Load Last

```
(run-with-idle-timer
 1 nil
 (lambda ()
   (when (member "Hack" (font-family-list))
     (set-face-attribute 'default nil
                         :family "Hack"
                         :height 114
                         :weight 'light)
     (message "Font set to Hack"))))
```

```
(custom-set-faces
 '(default ((t (:family "Hack" :height 114 :weight light)))))

(add-hook 'emacs-startup-hook
          (lambda ()
            (load-theme 'manoj-dark t)))

(load-theme 'manoj-dark t)
```