Emacs Configuration File

To speed things up

```
(setq gc-cons-threshold 16777216)

(setq gc-cons-percentage 0.6)

(setq  message-log-max 16384)
```

Package Manager

To start we need the package “package”

```
(require 'package)

(setq gnutls-algorithm-priority "NORMAL:-VERS-TLS1.3")

(setq package-archives '(("org" . "https://orgmode.org/elpa/")
                         ("gnu" . "https://elpa.gnu.org/packages/")
                         ("melpa"     . "https://melpa.org/packages/")))
(package-initialize)
```

Next let’s make sure we have use-package

```
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))
```

```
(setq use-package-always-ensure t)
(setq use-package-verbose t)
```

General Settings

A. Init File Support

Load up a collection of enhancements to Emacs Lisp, including dash, s for string manipulation, and f for file manipulation.

```
(use-package cl-lib
:ensure t
)
     (use-package dash

       :config (eval-after-load "dash" '(dash-enable-font-lock))
```

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

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

B. Trace for obsolete packages

```
(defun debug-on-load-obsolete (filename)
  (when (equal (car (last (split-string filename "[/\\]") 2))
               "obsolete")
    (debug)))
(add-to-list 'after-load-functions #'debug-on-load-obsolete)
```

C. Whoami

```
(setq user-full-name "Timo Lassmann"
      user-mail-address "timo.lassmann@telethonkids.org.au")
```

D. Emacs directory

```
(defconst tl/emacs-directory (concat (getenv "HOME") "/.emacs.d/"))
(defun tl/emacs-subdirectory (d) (expand-file-name d tl/emacs-directory))
```

```
(message "%s" (tl/emacs-subdirectory "elisp"))
(add-to-list 'load-path (tl/emacs-subdirectory "elisp"))
```

E. Basic looks

remove startup screen

```
(setq inhibit-startup-message t)
```

remove bars ets

```
(tool-bar-mode -1)
(menu-bar-mode -1)
(scroll-bar-mode -1)
(mouse-wheel-mode -1)
```

Disable bell

```
(setq ring-bell-function 'ignore)
```

UTF-8

```
(setq locale-coding-system 'utf-8)
(set-terminal-coding-system 'utf-8)
(set-keyboard-coding-system 'utf-8)
(set-selection-coding-system 'utf-8)
(prefer-coding-system 'utf-8)
```

F. Turn off arrow keys

```
(require 'no-easy-keys)
(no-easy-keys 1)
```

G. Turn off mouse

H. Diminish

```
(use-package diminish

  :demand t)
```

I. Turn off auto-fill mode

```
(setq auto-fill-mode -1)
(setq-default fill-column 99999)
(setq fill-column 99999)
```

J. Undo / redo

According to this article, I get better functionality than the redo+ plugin (which I can't seem to get working well).

```
(use-package undo-tree

  :diminish
  :init
  (global-undo-tree-mode 1)
  :config
  (defalias 'redo 'undo-tree-redo)
  :bind (("C-z" . undo)      ; Zap to character isn't helpful
         ("C-S-z" . redo)))
```

K. Kill this buffer

Assume that I always want to kill the current buffer when hitting C-x k.

```
(defun tl/kill-current-buffer ()
  "Kill the current buffer without prompting."
  (interactive)
  (kill-buffer (current-buffer)))
(global-set-key (kbd "C-x k") 'tl/kill-current-buffer)
```

L. Tabs

Never use tabs. Tabs are the devil's whitespace.

```
(setq-default indent-tabs-mode nil)
(setq tab-width 4)
;; (setq-default tab-always-indent 'complete)
```

M. Location of mactex (if we are using mac - unlikely...

Tell emacs about the mactex installation...

```
(setenv "PATH" (concat "/Library/TeX/texbin" ":" (getenv "PATH")))
```

This should do it...

N. Dired settings

Taken from: `https://github.com/munen/emacs.d`
Ability to use `a` to visit a new directory or file in dired instead of using `RET`. `RET` works just fine, but it will create a new buffer for every interaction whereas a reuses the current buffer.

```
(put 'dired-find-alternate-file 'disabled nil)
(setq-default dired-listing-switches "-alh")
```

O. Path

```
;;(let ((path-from-shell (shell-command-to-string "/bin/bash -l -c 'echo
 ↪  $PATH'")))
;;   (setenv "PATH" path-from-shell)
;;   (setq exec-path (split-string path-from-shell path-separator)))
```

```
(setq temporary-file-directory "/tmp")
```

P. Modernizing Emacs

Found here how to remove the warnings from the GnuTLS library when using HTTPS... increase the minimum prime bits size:

```
(setq gnutls-min-prime-bits 4096)
```

Q. Turn off some default key-bindings

I keep hitting this by accidental

```
(global-unset-key (kbd "C-z"))
(global-unset-key (kbd "C-x C-z"))
(global-unset-key (kbd "C-h h"))
(global-unset-key (kbd "C-x C-c"))


(defun tl/quit-emacs ()
  "Kill the current buffer without prompting."
  (interactive)
  (save-buffers-kill-terminal))
```

#+END$_{\text{SRC}}$

R. Recentf

```
(use-package recentf
  :custom
  (recentf-save-file "~/.emacs.d/recentf")
  (recentf-max-menu-items 10)
  (recentf-max-saved-items 200)
  (recentf-show-file-shortcuts-flag nil)
  :config
  (recentf-mode 1)
  (add-to-list 'recentf-exclude
               (expand-file-name "~/.emacs.d/company-statistics-cache.el"))
  ;; rename entries in recentf when moving files in dired
  (defun rjs/recentf-rename-directory (oldname newname)
    ;; oldname, newname and all entries of recentf-list should already
    ;; be absolute and normalised so I think this can just test whether
    ;; oldname is a prefix of the element.
    (setq recentf-list
          (mapcar (lambda (name)
                    (if (string-prefix-p oldname name)
                        (concat newname (substring name (length oldname)))
                      name))
                  recentf-list))
    (recentf-cleanup))

  (defun rjs/recentf-rename-file (oldname newname)
    (setq recentf-list
          (mapcar (lambda (name)
                    (if (string-equal name oldname)
                        newname
                      oldname))
                  recentf-list))
    (recentf-cleanup))

  (defun rjs/recentf-rename-notify (oldname newname &rest args)
    (if (file-directory-p newname)
        (rjs/recentf-rename-directory oldname newname)
      (rjs/recentf-rename-file oldname newname)))

  (advice-add 'dired-rename-file :after #'rjs/recentf-rename-notify)

  (defun contrib/recentf-add-dired-directory ()
    "Include Dired buffers in the list.  Particularly useful when
     combined with a completion framework's ability to display virtual
     buffers."
    (when (and (stringp dired-directory)
               (equal "" (file-name-nondirectory dired-directory)))
      (recentf-add-file dired-directory))))
```

S. Saveplace (remember point position)

```
(use-package saveplace
  :custom
  (save-place-file "~/.emacs.d/saveplace")
  :config
  (save-place-mode 1))
```

Basic Functionality

A. Highlight current line

hl-line is awesome! It's not very awesome in the terminal version of emacs though, so we don't use that. Besides, it's only used for programming.

```
(when window-system (add-hook 'prog-mode-hook 'hl-line-mode))
```

B. yes-no to y-n

```
(defalias 'yes-or-no-p 'y-or-n-p)
```

C. Async

Lets us use asynchronous processes wherever possible, pretty useful.

```
(use-package async

  :init (dired-async-mode 1))
```

D. Projectile

Projectile is an awesome project manager, mostly because it recognizes directories with a .git directory as projects and helps you manage them accordingly. Enable projectile globally This makes sure that everything can be a project.

```
(use-package projectile
  :ensure t
  ;; :delight '(:eval (concat " " (projectile-project-name)))
  :delight
  :custom
  (projectile-project-search-path '("~/"))
  (projectile-indexing-method 'alien)
  (projectile-enable-caching t)
  (projectile-completion-system 'ivy))

(use-package counsel-projectile
  :ensure t
  :config
  (add-to-list 'ivy-initial-inputs-alist '(counsel-projectile-switch-project .
  ↪  ""))
  (counsel-projectile-mode 1)
  ;; :bind-keymap ("M-s p" . projectile-command-map)
  :bind (("M-s b" . counsel-projectile-switch-to-buffer)
         ("M-s d" . counsel-projectile-find-dir)
         ("M-s p" . (lambda ()
                      (interactive)
                      (counsel-projectile-switch-project 4)))))
```

Let projectile call make

```
(global-set-key (kbd "<f5>") 'projectile-compile-project)
```

E. Insert date

This is a piece of code from JorgenSchaefersEmacsConfig.

```
(defun insert-date (prefix)
  "Insert the current date. With prefix-argument, use ISO format. With
   two prefix arguments, write out the day and month name."
  (interactive "P")
  (let ((format (cond
                 ((not prefix) "%d.%m.%Y")
                 ((equal prefix '(4)) "%Y-%m-%d")
                 ((equal prefix '(16)) "%A, %d. %B %Y")))
        (system-time-locale "en_US.UTF-8"))
    (insert (format-time-string format))))

(global-set-key (kbd "C-c d") 'insert-date)
```

F. Aggressive Auto Indention

Improvements

A. Better beginning of line

```
(defun smarter-move-beginning-of-line (arg)
  "Move point back to indentation of beginning of line.

Move point to the first non-whitespace character on this line.
If point is already there, move to the beginning of the line.
Effectively toggle between the first non-whitespace character and
the beginning of the line.

If ARG is not nil or 1, move forward ARG - 1 lines first.  If
point reaches the beginning or end of the buffer, stop there."
  (interactive "^p")
  (setq arg (or arg 1))

  ;; Move lines first
  (when (/= arg 1)
    (let ((line-move-visual nil))
      (forward-line (1- arg))))

  (let ((orig-point (point)))
    (back-to-indentation)
    (when (= orig-point (point))
      (move-beginning-of-line 1))))

;; remap C-a to `smarter-move-beginning-of-line'
(global-set-key [remap move-beginning-of-line]
↪  'smarter-move-beginning-of-line)
(global-set-key [remap org-beginning-of-line]
↪  'smarter-move-beginning-of-line)
```

Terminal

I have used urxvt for years, and I miss it sometimes, but ansi-term is enough for most of my tasks.

A. Default shell should be zsh

I don’t know why this is a thing, but asking me what shell to launch every single time I open a terminal makes me want to slap babies, this gets rid of it. This goes without saying but you can replace bash with your shell of choice.

```
(defvar my-term-shell "/usr/bin/zsh")
(defadvice ansi-term (before force-bash)
  (interactive (list my-term-shell)))
(ad-activate 'ansi-term)
```

B. Easy to remember keybinding

In loving memory of bspwm, Super + Enter opens a new terminal, old habits die hard.

```
(global-set-key (kbd "<s-return>") 'eshell)
```

Moving around

One of the most important things about a text editor is how efficient you manage to be when using it, how much time do basic tasks take you and so on and so forth. One of those tasks is moving around files and buffers, whatever you may use emacs for you will be jumping around buffers like it's serious businexss, the following set of enhancements aims to make it easier.
As a great emacs user once said:
Do me the favor, do me the biggest favor, matter of fact do yourself the biggest favor and integrate those into your workflow.

A. scrolling and why does the screen move

I don't know to be honest, but this little bit of code makes scrolling with emacs a lot nicer.

```
(setq scroll-conservatively 100)
```

B. which-key and why I love emacs

In order to use emacs, you don't need to know how to use emacs. It's self documenting, and coupled with this insanely useful package, it's even easier. In short, after you start the input of a command and stop, pondering what key must follow, it will automatically open a non-intrusive buffer at the bottom of the screen offering you suggestions for completing the command, that's it, nothing else.
It's beautiful

```
(use-package which-key

  :diminish which-key-mode
  :config
  (which-key-mode))
```

C. windows,panes and why I hate other-window

Some of us have large displays, others have tiny netbook screens, but regardless of your hardware you probably use more than 2 panes/windows at times, cycling through all of them with C-c o is annoying to say the least, it's a lot of keystrokes and takes time, time you could spend doing something more productive. switch-window
This magnificent package takes care of this issue. It's unnoticeable if you have <3 panes open, but with 3 or more, upon pressing C-x o you will notice how your buffers turn a solid color and each buffer is asigned a letter (the list below shows the letters, you can modify them to suit your liking), upon pressing a letter asigned to a window, your will be taken to said window, easy to remember, quick to use and most importantly, it annihilates a big issue I had with emacs. An alternative is ace-window, however by default it also changes the behaviour of C-x o even if only 2 windows are open, this is bad, it also works less well with exwm for some reason.

```
(use-package switch-window

  :config
  (setq switch-window-input-style 'minibuffer)
  (setq switch-window-increase 4)
  (setq switch-window-threshold 2)
  (setq switch-window-shortcut-style 'qwerty)
  (setq switch-window-qwerty-shortcuts
        '("a" "s" "d" "f" "j" "k" "l" "i" "o"))
  :bind
  ([remap other-window] . switch-window))
```

D. Following window splits

After you split a window, your focus remains in the previous one. This annoyed me so much I wrote these two, they take care of it.

```
(defun split-and-follow-horizontally ()
  (interactive)
  (split-window-below)
  (balance-windows)
  (other-window 1))
(global-set-key (kbd "C-x 2") 'split-and-follow-horizontally)

(defun split-and-follow-vertically ()
  (interactive)
  (split-window-right)
  (balance-windows)
  (other-window 1))
(global-set-key (kbd "C-x 3") 'split-and-follow-vertically)
```

E. Ivy

This Ivy configuration is copied from Protesilaos Stavrou's old setup (https://gitlab.com/protesilaos/dotfiles)
And here are some 'hidden' key bindings for making the most out of Ivy (find more in the official manual).

Key	Function	Description
M-o	ivy-dispatching-done	Show actions for current match.
C-c C-o	ivy-occur	Place the list in a standalone buffer.
C-M-m	ivy-call	Run command, keep minibuffer open.
M-i	ivy-insert-current	Insert match in the prompt.
M-j	ivy-yank-word	Put word at point in the minibuffer prompt.
S-SPC	ivy-restrict-to-matches	Restrict list to prompt (and search anew).
C-SPC	ivy-restrict-to-matches	My alias for the above.

```
(use-package ivy
  :ensure t
  :delight
  :custom
  (ivy-count-format "(%d/%d) ")
  (ivy-height-alist '((t lambda (_caller) (/ (window-height) 4))))
  (ivy-use-virtual-buffers t)
  (ivy-wrap nil)
  (ivy-re-builders-alist
   '((counsel-M-x . ivy--regex-fuzzy)
     (ivy-switch-buffer . ivy--regex-fuzzy)
     (ivy-switch-buffer-other-window . ivy--regex-fuzzy)
     (counsel-rg . ivy--regex-or-literal)
     (t . ivy--regex-plus)))
  (ivy-display-style 'fancy)
  (ivy-use-selectable-prompt t)
  (ivy-fixed-height-minibuffer nil)
  (ivy-initial-inputs-alist
   '((counsel-M-x . "^")
     (ivy-switch-buffer . "^")
     (ivy-switch-buffer-other-window . "^")
     (counsel-describe-function . "^")
     (counsel-describe-variable . "^")
     (t . "")))
  :config
  (ivy-set-occur 'counsel-fzf 'counsel-fzf-occur)
  (ivy-set-occur 'counsel-rg 'counsel-ag-occur)
  (ivy-set-occur 'ivy-switch-buffer 'ivy-switch-buffer-occur)
  (ivy-set-occur 'swiper 'swiper-occur)
  (ivy-set-occur 'swiper-isearch 'swiper-occur)
  (ivy-set-occur 'swiper-multi 'counsel-ag-occur)
  (ivy-mode 1)
  :hook
  (ivy-occur-mode . hl-line-mode)
  :bind (("<s-up>" . ivy-push-view)
         ("<s-down>" . ivy-switch-view)
         ("C-S-r" . ivy-resume)
         :map ivy-occur-mode-map
         ("f" . forward-char)
         ("b" . backward-char)
         ("n" . ivy-occur-next-line)
         ("p" . ivy-occur-previous-line)
         ("<C-return>" . ivy-occur-press)))
```

Prescient

```
(use-package prescient
  :ensure t
  :custom
  (prescient-history-length 200)
  (prescient-save-file "~/.emacs.d/prescient-items")
  (prescient-filter-method '(literal regexp))
  :config
  (prescient-persist-mode 1))

(use-package ivy-prescient
  :ensure t
  :after (prescient ivy)
  :custom
  (ivy-prescient-sort-commands
   '(:not counsel-grep
          counsel-rg
          counsel-switch-buffer
          ivy-switch-buffer
          swiper
          swiper-multi))
  (ivy-prescient-retain-classic-highlighting t)
  (ivy-prescient-enable-filtering nil)
  (ivy-prescient-enable-sorting t)
  :config
  (ivy-prescient-mode 1))
```

```
(use-package counsel
  :ensure t
  :after ivy
  :custom
  (counsel-yank-pop-preselect-last t)
  (counsel-yank-pop-separator "\n---------\n")
  (counsel-rg-base-command
   "rg -SHn --no-heading --color never --no-follow --hidden %s")
  (counsel-find-file-occur-cmd          ; TODO Simplify this
   "ls -a | grep -i -E '%s' | tr '\\n' '\\0' | xargs -0 ls -d
    ↪  --group-directories-first")
  :config
  (defun prot/counsel-fzf-rg-files (&optional input dir)
    "Run `fzf' in tandem with `ripgrep' to find files in the
present directory.  If invoked from inside a version-controlled
repository, then the corresponding root is used instead."
    (interactive)
    (let* ((process-environment
            (cons (concat "FZF_DEFAULT_COMMAND=rg -Sn --color never --files
             ↪  --no-follow --hidden")
                  process-environment))
           (vc (vc-root-dir)))
      (if dir
          (counsel-fzf input dir)
        (if (eq vc nil)
            (counsel-fzf input default-directory)
          (counsel-fzf input vc)))))

  (defun prot/counsel-fzf-dir (arg)
    "Specify root directory for `counsel-fzf'."
    (prot/counsel-fzf-rg-files ivy-text
                               (read-directory-name
                                (concat (car (split-string counsel-fzf-cmd))
                                        " in directory: "))))

  (defun prot/counsel-rg-dir (arg)
    "Specify root directory for `counsel-rg'."
    (let ((current-prefix-arg '(4)))
      (counsel-rg ivy-text nil "")))

  ;; TODO generalise for all relevant file/buffer counsel-*?
  (defun prot/counsel-fzf-ace-window (arg)
    "Use `ace-window' on `prot/counsel-fzf-rg-files' candidate."
    (ace-window t)
    (let ((default-directory (if (eq (vc-root-dir) nil)
                                 counsel--fzf-dir
                               (vc-root-dir))))
      (if (> (length (aw-window-list)) 1)
          (find-file arg)
        (find-file-other-window arg))
      (balance-windows (current-buffer))))

  ;; Pass functions as appropriate Ivy actions (accessed via M-o)
  (ivy-add-actions
   'counsel-fzf
   '(("r" prot/counsel-fzf-dir "change root directory")
     ("g" prot/counsel-rg-dir "use ripgrep in root directory")
     ("a" prot/counsel-fzf-ace-window "ace-window switch")))

  (ivy-add-actions
   'counsel-rg
   '(("r" prot/counsel-rg-dir "change root directory")
     ("z" prot/counsel-fzf-dir "find file with fzf in root directory")))

  (ivy-add-actions
   'counsel-find-file
   '(("g" prot/counsel-rg-dir "use ripgrep in root directory")
     ("z" prot/counsel-fzf-dir "find file with fzf in root directory")))

  ;; Remove commands that only work with key bindings
  (put 'counsel-find-symbol 'no-counsel-M-x t)
  :bind (("M-x" . counsel-M-x)
         ("C-x C-f" . counsel-find-file)
         ;;("s-f" . counsel-find-file)
         ;;("s-F" . find-file-other-window)
         ("C-x b" . ivy-switch-buffer)
         ;;("s-b" . ivy-switch-buffer)
         ("C-x B" . counsel-switch-buffer-other-window)
         ;;("s-B" . counsel-switch-buffer-other-window)
         ("C-x d" . counsel-dired)
         ;;("s-d" . counsel-dired)
         ;;("s-D" . dired-other-window)
         ("C-x C-r" . counsel-recentf)
         ;;("s-r" . counsel-recentf)
         ;;("s-y" . counsel-yank-pop)
         ("C-h f" . counsel-describe-function)
         ("C-h v" . counsel-describe-variable)
         ("M-s r" . counsel-rg)
         ("M-s g" . counsel-git-grep)
         ("M-s l" . counsel-find-library)
         ("M-s z" . prot/counsel-fzf-rg-files)
         :map ivy-minibuffer-map
         ("C-r" . counsel-minibuffer-history)
         ("s-y" . ivy-next-line)        ; Avoid 2× `counsel-yank-pop'
         ("C-SPC" . ivy-restrict-to-matches)))
```

F. Swiper

```
(use-package swiper
  :ensure t
  :after ivy
  :custom
  (swiper-action-recenter t)
  (swiper-goto-start-of-match t)
  (swiper-include-line-number-in-search t)
  :bind (("C-s" . swiper)
         ("M-s s" . swiper-multi)
         ("M-s w" . swiper-thing-at-point)))
```

G. Avy

```
(use-package avy
  :config
  (global-set-key (kbd "M-SPC") 'avy-goto-char-timer)
  (global-set-key (kbd "C-:") 'avy-goto-char)
  (global-set-key (kbd "C-'") 'avy-goto-char-2)
  (global-set-key (kbd "M-g f") 'avy-goto-line)
  (global-set-key (kbd "M-g w") 'avy-goto-word-1)
  (global-set-key (kbd "M-g e") 'avy-goto-word-0))
```

H. Ivy extensions

```
(use-package ivy-rich
  :ensure t
  :custom
  (ivy-rich-path-style 'abbreviate)
  :config
  (setcdr (assq t ivy-format-functions-alist)
          #'ivy-format-function-line)
  (ivy-rich-mode 1))
```

```
(use-package ivy-posframe
  :ensure t
  :delight
  :custom
  (ivy-posframe-parameters
   '((left-fringe . 2)
     (right-fringe . 2)
     (internal-border-width . 2)
     ))
  (ivy-posframe-height-alist
   '((swiper . 15)
     (swiper-isearch . 15)
     (t . 10)))
  (ivy-posframe-display-functions-alist
   '((complete-symbol . ivy-posframe-display-at-point)
     (swiper . nil)
     (swiper-isearch . nil)
     (t . ivy-posframe-display-at-frame-center)))
  :config
  (ivy-posframe-mode 1))
```

I. Swoop

```
(use-package helm-swoop
  :bind (("C-c h o" . helm-swoop)
         ("C-c s" . helm-multi-swoop-all))
  :config
  ;; When doing isearch, hand the word over to helm-swoop
  (define-key isearch-mode-map (kbd "M-i") 'helm-swoop-from-isearch)

  ;; From helm-swoop to helm-multi-swoop-all
  (define-key helm-swoop-map (kbd "M-i") 'helm-multi-swoop-all-from-helm-swoop)

  ;; Save buffer when helm-multi-swoop-edit complete
  (setq helm-multi-swoop-edit-save t)

  ;; If this value is t, split window inside the current window
  (setq helm-swoop-split-with-multiple-windows t)

  ;; Split direcion. 'split-window-vertically or 'split-window-horizontally
  (setq helm-swoop-split-direction 'split-window-vertically)
  ;; If nil, you can slightly boost invoke speed in exchange for text color
  (setq helm-swoop-speed-or-color t))
```

J. Winner mode

```
(use-package winner

  :init (winner-mode 1))
```

Completion

A. IDO

```
(use-package ido

  :init  (setq ido-enable-flex-matching t
               ido-ignore-extensions t
               ido-use-virtual-buffers t
               ido-everywhere t)
  :config
  (ido-mode 1)
  (ido-everywhere 1)
  (add-to-list 'completion-ignored-extensions ".pyc"))
```

ido-completing-read+

```
(use-package ido-completing-read+

  :config
  (ido-ubiquitous-mode))
```

FLX package

```
(use-package flx-ido

  :init (setq ido-enable-flex-matching t
              ido-use-faces nil)
  :config (flx-ido-mode 1))
```

Vertical mode

```
(use-package ido-vertical-mode

  :init              ; I like up and down arrow keys:
  (setq ido-vertical-define-keys 'C-n-C-p-up-and-down)
  :config
  (ido-vertical-mode 1))
```

```
(defun ido-sort-mtime ()
  "Reorder the IDO file list to sort from most recently modified."
  (setq ido-temp-list
        (sort ido-temp-list
              (lambda (a b)
                (ignore-errors
                  (time-less-p
                   (sixth (file-attributes (concat ido-current-directory b)))
                   (sixth (file-attributes (concat ido-current-directory
                    ↪  a)))))))))
  (ido-to-end  ;; move . files to end (again)
   (delq nil (mapcar
              (lambda (x) (and (char-equal (string-to-char x) ?.) x))
              ido-temp-list))))

(add-hook 'ido-make-file-list-hook 'ido-sort-mtime)
(add-hook 'ido-make-dir-list-hook 'ido-sort-mtime)
```

B. Counsel

Counsel tramp

```
(use-package counsel-tramp)
```

C. Company

```
(use-package company-c-headers
  :ensure t
  )
(use-package company
  :ensure t
  :delight
  :custom
  (company-auto-complete nil)
  (company-dabbrev-code-everywhere t)
  (company-dabbrev-code-modes t)
  (company-dabbrev-code-other-buffers 'all)
  (company-dabbrev-downcase nil)
  (company-dabbrev-ignore-case t)
  (company-dabbrev-other-buffers 'all)
  (company-idle-delay 0.3)
  (company-minimum-prefix-length 3)
  (company-require-match nil)
  (company-selection-wrap-around t)
  (company-show-numbers t)
  ;;(company-transformers
  ;;'(company-sort-by-backend-importance
  ;;          company-sort-prefer-same-case-prefix
  ;;        company-sort-by-occurrence))
  (company-tooltip-align-annotations t)
  (company-tooltip-limit 10)
  (company-tooltip-margin 1)
  (company-tooltip-offset-display 'scrollbar)
  :config
  (global-company-mode 1)

  (add-to-list 'company-backends '(company-clang
                                   company-capf
                                   company-dabbrev
                                   company-c-headers
                                   company-gtags))
  :bind (:map company-mode-map
              ("M-/" . company-manual-begin)
              :map company-active-map
              (("M-/" . company-other-backend)
               ("<tab>" . company-complete-selection)
               ("<C-tab>" . company-complete-common-or-cycle)
               ("C-n" . company-select-next)
               ("C-p" . company-select-previous))))

(use-package company-statistics
  :init
  (company-statistics-mode))


(setq company-global-modes '(
                             org-mode
                             c-mode
                             c++-mode
                             ))

(add-hook 'c-mode-hook
          (lambda ()
            (set (make-local-variable 'company-backends) '(company-clang
            ↪  company-gtags  company-c-headers company-dabbrev ))))
```

I had to add the hook and local variable to stop company from selecting capf before clang. To make this work properly, I need to manually specify the include paths by putting a `.dir-locals.el` into the source directory of my C code. I.e. most of the time this will be `src` and I need to point to `../tldevel`.
In addition add the include path to flycheck-clang!

```
(
 (c-mode . ((company-clang-arguments . ("-I."  "-I../tldevel-1.2.8/"))))
 (c-mode . ((company-c-headers-path-user . ("." "../tldevel-1.2.8/"))))
 (c-mode . ((flycheck-clang-include-path . ("-I." "-I../tldevel-1.2.8/"))))
 )
```

Company Prescient

```
(use-package company-prescient
  :ensure t
  :after (company prescient)
  :config
  (company-prescient-mode 1))
```

Writing

A. Flyspell config

Installing aspell on linux:

```
apt install aspell aspell-en
```

on mac:

```
brew install aspell
```

Note in the config below I assume aspell is installed in `/usr/bin/` !.

```
(use-package flyspell

  :diminish flyspell-mode
  :init
  (add-hook 'prog-mode-hook 'flyspell-prog-mode)

  (dolist (hook '(text-mode-hook org-mode-hook))
    (add-hook hook (lambda () (flyspell-mode 1))))

  (dolist (hook '(change-log-mode-hook log-edit-mode-hook
  ↪  org-agenda-mode-hook))
    (add-hook hook (lambda () (flyspell-mode -1))))

  :config
  (setq ispell-program-name "aspell"
        ispell-local-dictionary "en_GB"
        ;;ispell-dictionary "american" ; better for aspellr
        ispell-extra-args '("--sug-mode=ultra" "--lang=en_GB")
        ispell-list-command "--list"
        ispell-local-dictionary-alist '(("en_GB" "[[:alpha:]]" "[^[:alpha:]]"
        ↪  "['‘’]"
                                         t ; Many other characters
                                         ("-d" "en_GB") nil utf-8))))
```

There is more stuff in Howard Abram's config but I'll leave this for now..

B. Writegood mode

This does not work - there is a wring gpg signature in melpa...
this mode will improve various aspects of writing.
end.

C. LangTool

I added the Emacs-langtool code from:
https://github.com/mhayashi1120/Emacs-langtool
To my `/elisp/` directory.
To install langtool install `maven` package, java 8 then:

```
cd ~/programs
git clone https://github.com/languagetool-org/languagetool.git
./build.sh languagetool-standalone package
```

This does not work!
I now simply download the pre-compiles zip package...
To load:

```
(require 'langtool)
(setq langtool-language-tool-jar
 ↪  "/home/user/programs/langtool/LanguageTool-4.0/languagetool-commandline.jar")
```

Org-mode

A. General setup

load org mode

```
(use-package org
  :init
  (setq org-use-speed-commands t
        org-return-follows-link t
        org-completion-use-ido t
        org-outline-path-complete-in-steps nil))
```

Directory, inbox ..

```
(setq org-directory "~/")
(defun org-file-path (filename)
  "Return the absolute address of an org file, given its relative name."
  (concat (file-name-as-directory org-directory) filename))
(setq org-index-file (org-file-path "/capture/inbox.org"))
(setq org-archive-location
      (concat (org-file-path "archive.org") "::* From %s"))
```

Multiple files for agenda source:

```
;;   (setq org-agenda-files (list org-index-file))
(setq org-agenda-files '("~/capture"
                          "~/work"
                          "~/work/roam"
                          "~/work/roam/dailies"
                          "~/life"))
```

Refile targets / create new targets if necessary

```
;;(setq org-refile-targets '((org-agenda-files :maxlevel . 3)))
(setq org-refile-targets '(("~/work/work-todo.org" :maxlevel . 2)
                            ("~/work/work-todo-archive.org" :maxlevel . 2)
                            ("~/life/life-todo.org" :maxlevel . 2)
                            ))
(setq org-refile-use-outline-path 'file)
(setq org-refile-allow-creating-parent-nodes 'confirm)
(setq org-refile-allow-creating-parent-nodes 'confirm)
```

Drawers

```
(setq org-log-into-drawer t)

;; Add the REPORT drawer
(setq org-drawers '("PROPERTIES" "CLOCK" "LOGBOOK" "REPORT"))
```

Hitting C-c C-x C-s will mark a todo as done and move it to an appropriate place in the archive.

```
(defun tl/mark-done-and-archive ()
  "Mark the state of an org-mode item as DONE and archive it."
  (interactive)
  (org-todo 'done)
  (org-archive-subtree))

;;    (define-key org-mode-map (kbd "C-c C-x C-s")
 ↪  'tl/mark-done-and-archive)
```

Record the time that a todo was archived.

```
(setq org-log-done 'time)
```

```
(add-hook 'org-mode-hook 'visual-line-mode)
```

B. Capture

Capture templates..

```
(setq org-capture-templates
      (quote (("t" "todo" entry (file+headline org-index-file "Inbox")
               "* TODO %?\nSCHEDULED: %(org-insert-time-stamp (org-read-date nil t \"+0d\"))\n%a\n")
              ("n" "note" entry (file+headline org-index-file "Inbox")
               "* %?\n\n  %i\n\n  See: %a" :empty-lines 1)
              ("r" "respond" entry (file+headline org-index-file "Inbox")
               "* TODO Respond to %:from on %:subject\nSCHEDULED: %(org-insert-time-stamp (org-read-date nil t \"+0d\"))\n%a\n")
              ("m" "Mail" entry (file+headline org-index-file "Inbox")
               "* TODO %?\n%a   %:from %:fromname %:fromaddress" :prepend t :jump-to-captured t)
              ("p" "Daily Plan" plain (file+datetree "~/planning/daily-plan.org")
               "+ [ ] The 3 most important tasks [/]
                - [ ]
                - [ ]
                - [ ]
              + [ ] Other tasks that are in the system [/]
                - [ ]
              + [ ] ToDos which are not tracked by my system [/]
                - [ ] " :immediate-finish t)
              )))
```

C. Taking Meeting Notes

directly from https://github.com/howardabrams/dot-files/blob/master/emacs-org.org) I've notice that while I really like taking notes in a meeting, I don't always like the multiple windows I have opened, so I created this function that I can easily call to eliminate distractions during a meeting.

```
(defun meeting-notes ()
  "Call this after creating an org-mode heading for where the notes for the meeting
should be. After calling this function, call 'meeting-done' to reset the environment."
  (interactive)
  (outline-mark-subtree)                              ;; Select org-mode section
  (narrow-to-region (region-beginning) (region-end))  ;; Only show that region
  (deactivate-mark)
  (delete-other-windows)                              ;; Get rid of other windows
  (text-scale-set 3)                                ;; Text is now readable
  ↪  by others
  (fringe-mode 0)
  (message "When finished taking your notes, run meeting-done."))
```

Of course, I need an 'undo' feature when the meeting is over…

```
(defun meeting-done ()
  "Attempt to 'undo' the effects of taking meeting notes."
  (interactive)
  (widen)                                     ;; Opposite of
  ↪  narrow-to-region
  (text-scale-set 0)                          ;; Reset the font size
  ↪  increase
  (fringe-mode 1)
  (winner-undo))                              ;; Put the windows back in
  ↪  place
```

End.

D. Coding

Allow babel to evaluate C …

```
(org-babel-do-load-languages
 'org-babel-load-languages
 '((C . t)
   (R . t)
   (dot . t)
   (emacs-lisp . t)
   (shell . t)
   (awk . t)
   (makefile . t)
   (latex . t)
   (java . t)
   (clojure . t)
   ))
```

Don't ask before evaluating code blocks.

```
(setq org-confirm-babel-evaluate nil)
```

smart brackets in export

```
(setq org-export-with-smart-quotes t)
```

Done.

E. Export

Export packages...

```
(require 'ox-latex)
(require 'ox-beamer)
```

Htmlize required for reveal...

```
(use-package htmlize
  )
```

Use minted package for code:

```
(setq org-latex-listings 'minted)
(setq org-latex-minted-options
      '(("frame" "lines") ("linenos=true")("breaklines")))
```

F. Flyspell

Enable spell-checking in Org-mode.

```
(add-hook 'org-mode-hook 'flyspell-mode)
```

G. Color and display

Use syntax highlighting in source blocks while editing.

```
(setq org-src-fontify-natively t)
```

Make TAB act as if it were issued in a buffer of the language's major mode.

```
(setq org-src-tab-acts-natively t)
```

When editing a code snippet, use the current window rather than popping open a new one (which shows the same information).

```
(setq org-src-window-setup 'current-window)
```

H. Bullets

```
(use-package org-superstar

  :init
  (add-hook 'org-mode-hook (lambda () (org-superstar-mode 1)))
  )
```

I. Image preview

Inline images support:

```
(setq org-latex-create-formula-image-program 'imagemagick)

(add-to-list 'org-latex-packages-alist
             '("" "tikz" t))

(eval-after-load "preview"
  '(add-to-list 'preview-default-preamble "\\PreviewEnvironment{tikzpicture}"
  ↪  t))
(setq org-latex-create-formula-image-program 'imagemagick)


(setq org-confirm-babel-evaluate nil)
(add-hook 'org-babel-after-execute-hook 'org-display-inline-images)
(add-hook 'org-mode-hook 'org-display-inline-images)
```

J. Keybindings

Standard bindings

```
(define-key global-map "\C-cl" 'org-store-link)
(define-key global-map "\C-ca" 'org-agenda)
(define-key global-map "\C-cc" 'org-capture)
```

Quickly open index file

```
(defun open-index-file ()
  "Open the master org TODO list."
  (interactive)
  (find-file org-index-file)
  (flycheck-mode -1)
  (end-of-buffer))

(global-set-key (kbd "C-c i") 'open-index-file)
```

undef a key

```
(add-hook 'org-mode-hook
          '(lambda ()
             ;; Undefine C-c [ and C-c ] since this breaks my
             ;; org-agenda files when directories are include It
             ;; expands the files in the directories individually
             (org-defkey org-mode-map "\C-c[" 'undefined))
          'append)
```

K. deft

L. Helm-bibtex

Define format for bibtex entries

```elisp
;; variables that control bibtex key format for auto-generation
;; I want firstauthor-year-title-words
;; this usually makes a legitimate filename to store pdfs under.
(setq bibtex-autokey-year-length 4
      bibtex-autokey-name-year-separator "-"
      bibtex-autokey-year-title-separator "-"
      bibtex-autokey-titleword-separator "-"
      bibtex-autokey-titlewords 2
      bibtex-autokey-titlewords-stretch 1
      bibtex-autokey-titleword-length 5)

(setq bibtex-completion-bibliography "~/work/bibliography/references.bib"
      bibtex-completion-library-path "~/work/bibliography/bibtex-pdfs"
      bibtex-completion-notes-path "~/work/bibliography/helm-bibtex-notes"
      bibtex-completion-pdf-field "file")
(use-package ivy-bibtex
  :ensure t
  )
```

M. Org-ref

```elisp
(setq reftex-default-bibliography '("~/work/bibliography/references.bib"))
```

```elisp
(setq  notes-directory (concat (getenv "HOME") "/work/roam/"))

(use-package org-ref
  :ensure t
  :config
  (setq
   org-ref-completion-library 'org-ref-ivy-cite
   org-ref-get-pdf-filename-function 'org-ref-get-pdf-filename-helm-bibtex
   org-ref-default-bibliography '("~/work/bibliography/references.bib")
   org-ref-bibliography-notes "~/work/roam/notes.org"
   org-ref-pdf-directory "~/work/bibliography/bibtex-pdfs/"

   org-ref-notes-directory "~/work/roam/"
   org-ref-notes-function 'orb-edit-notes
   org-ref-default-citation-link "supercite"
   )
  )


```

```
;;Hack ....
(defun org-ref-add-labels (start end)
  "Add labels in the region from START to END.
     This is run by font-lock. START tends to be the beginning of the
     line, and END tends to be where the point is, so this function
     seems to work fine at recognizing labels by the regexps in
     `org-ref-label-regexps'."
  (interactive "r")
  (save-excursion
    (save-match-data
      (cl-loop for rx in org-ref-label-regexps
               do
               (goto-char start)
               (while (re-search-forward rx end t)
                 (let ((label (match-string-no-properties 1)))
                   ;; I don't know why this gets found, but some labels are
                   ;; empty strings. we don't store these.
                   (unless (string= "" label)
                     ;; if the last end is the new end -1 we are adding to a
                     ;; label, and should pop the old one off before adding
                     ↪  the
                     ;; new one.
                     (when (eq  org-ref-last-label-end (- end 1))
                       (pop org-ref-labels))
                     (with-silent-modifications
                       (put-text-property (match-beginning 1)
                                          (match-end 1)
                                          'org-ref-label t)
                       (put-text-property (match-beginning 1)
                                          (match-end 1)
                                          'rear-nonsticky '(org-ref-label)))
                     (when org-ref-label-debug
                       (message "oral: adding %s" label))

                     (cl-pushnew label
                                 org-ref-labels :test 'string=)
                     ;; now store the last end so we can tell for the next
                     ↪  run
                     ;; if we are adding to a label.
                     (setq org-ref-last-label-end end)))))))))
```

Make `supercite` the default citation type:
Where are the refs?
End.

N. Org roam

```
     (use-package org-roam
       :ensure t
       :hook
       (after-init . org-roam-mode)
       :custom
       (org-roam-directory "~/work/roam/")

       :bind (:map org-roam-mode-map
                   (("C-c m l" . org-roam)
                    ("C-c m F" . org-roam-find-file)
                    ("C-c m r" . org-roam-find-ref)
                    ("C-c m ." . org-roam-find-directory)
                    ("C-c m d" . org-roam-dailies-today)
                    ("C-c m j" . org-roam-jump-to-index)
                    ("C-c m b" . org-roam-switch-to-buffer)
                    ("C-c m g" . org-roam-graph))
                   :map org-mode-map
                   (("C-c m i" . org-roam-insert)))
       )
(setq org-roam-index-file "~/work/roam/Index.org")
```

Templates:

```
(setq org-roam-capture-templates
      (quote (("d" "default" plain
               (function org-roam-capture--get-point)
               "\n* %?"
               :file-name "%<%Y%m%d%H%M%S>-${slug}"
               :head "#+title: ${title}\n#+created: %u\n#+last_modified:
               ↪  %U\n\n"
               :unnarrowed t))
             )
      )
(setq org-roam-capture-ref-templates
      (quote (("r" "ref" plain
               (function org-roam-capture--get-point)
               ""
               :file-name "${slug}"
               :head "#+title: ${title}\n#+roam_key: ${ref}\n#+created:
               ↪  %u\n#+last_modified: %U\n\n"
               :unnarrowed t))
             )
      )
(setq org-roam-dailies-capture-templates
      (quote (("d" "daily" plain (function org-roam-capture--get-point) "*
      ↪  %?\n"
```

```
                ;;                      :immediate-finish t
                :add-created t
                :file-name "dailies/%<%Y-%m-%d>"
                :head "#+TITLE: %<%Y-%m-%d>\n\n"))
              )
       )
```

```
(require 'org-roam-protocol)
(use-package org-roam-server
  :ensure t
  :bind (:map org-roam-mode-map
              (("C-c m G" . org-roam-server-mode)))
  :config
  (setq org-roam-server-host "127.0.0.1"
        org-roam-server-port 8080
        org-roam-server-export-inline-images t
        org-roam-server-authenticate nil
        org-roam-server-network-poll t
        org-roam-server-network-arrows nil
        org-roam-server-network-label-truncate t
        org-roam-server-network-label-truncate-length 60
        org-roam-server-network-label-wrap-length 20))
```

O. Org-roam-bibtex

```
(use-package org-roam-bibtex
  :after org-roam
  :hook (org-roam-mode . org-roam-bibtex-mode)
  :bind (:map org-mode-map
              (("C-c n a" . orb-note-actions))))

(setq orb-preformat-keywords
      '(("citekey" . "=key=") "title" "url" "file" "author-or-editor"
      ↪  "keywords"))

(setq orb-templates
      '(("r" "ref" plain (function org-roam-capture--get-point)
         ""
         :file-name "${citekey}"
         :head "#+TITLE: ${citekey}: ${title}\n#+ROAM_KEY: ${ref}

- tags ::
- keywords :: ${keywords}
\n* ${title}
:PROPERTIES:
:Custom_ID: ${citekey}
:URL: ${url}
:AUTHOR: ${author-or-editor}
:NOTER_DOCUMENT: %(orb-process-file-field \"${citekey}\")
:NOTER_PAGE:
:END:\n%?")))
```

P. company-org-roam

```
(use-package company-org-roam
  :ensure t
  ;; You may want to pin in case the version from stable.melpa.org is not
  ↪  working
                                                ; :pin melpa
  :config
  (push 'company-org-roam company-backends))
```

Q. Org- Noter

```
(setq
 org_notes (concat (getenv "HOME") "/work/roam/")
 deft-directory org_notes
 org-roam-directory org_notes
 )
(use-package org-noter
  :ensure t
  :after (:any org pdf-view)
  :config
  (setq org-noter-hide-other t
        org-noter-auto-save-last-location t
        org-noter-doc-split-fraction '(0.67 0.33)
        org-noter-notes-search-path  (list org_notes)))
```

R. Org-download

S. Latex templates

Latex templates

```
;;(setq org-latex-to-pdf-process '("xelatex %f && bibtex %f && xelatex %f &&
↪  xelatex %f"))
(defun sk-latexmk-cmd (backend)
  "When exporting from .org with latex, automatically run latex,
     pdflatex, or xelatex as appropriate, using latexmk."
  (when (org-export-derived-backend-p backend 'latex)
    (let ((texcmd)))
    ;; default command: xelatex
    (setq texcmd "jobname=$(basename %f | sed 's/\.tex//');latexmk -xelatex
    ↪  -shell-escape -quiet %f && mkdir -p latex.d && mv ${jobname}.*
    ↪  latex.d/. && mv latex.d/${jobname}.{org,pdf,fdb_latexmk,aux} .")
    ;; pdflatex -> .pdf
    (if (string-match "LATEX_CMD: pdflatex" (buffer-string))
        (setq texcmd "latexmk -pdflatex='pdflatex -shell-escape -interaction
        ↪  nonstopmode' -pdf -bibtex -f %f"))
    ;; xelatex -> .pdf
    (if (string-match "LATEX_CMD: xelatex" (buffer-string))
        (setq texcmd "latexmk -pdflatex='xelatex -shell-escape -interaction
        ↪  nonstopmode' -pdf -bibtex -f  %f"))
    ;; LaTeX compilation command
    (setq org-latex-pdf-process (list texcmd)))

(org-add-hook 'org-export-before-processing-hook 'sk-latexmk-cmd)

(unless (boundp 'org-latex-classes)
  (setq org-latex-classes nil))
```

T. CV

```
(add-to-list 'org-latex-classes
             '("CV"
               "\\documentclass[11pt]{article}
     \\usepackage{\\string~\"/.emacs.d/latex_templates/cv\"}
     [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}")))
```

U. NHMRC project grant

```
(add-to-list 'org-latex-classes
             '("NHMRC_project_grant"
               "\\documentclass[12pt,table,names]{article}
\\usepackage{\\string~\"/.emacs.d/latex_templates/NHMRC_grant\"}
[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}")))
```

Rebuttal…

```
(add-to-list 'org-latex-classes
             '("NHMRC_project_grant_rebuttal"
               "\\documentclass[12pt,table,names]{article}
```

```
  \\usepackage{\\string~\"/.emacs.d/latex_templates/NHMRC_grant\"}
  [NO-DEFAULT-PACKAGES]
  [NO-PACKAGES]"
                ("\\subsection{%s}" . "\\section*{%s}")
                ("\\subsubsection{%s}" . "\\subsection*{%s}")q
                ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                ("\\paragraph{%s}" . "\\paragraph*{%s}")
                ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
```

V. NHMRC Investigator

```
(add-to-list 'org-latex-classes
             '("NHMRC_investigator_grant"
               "\\documentclass[12pt,table,names]{article}
\\usepackage{\\string~\"/.emacs.d/latex_templates/NHMRC_investigator\"}
[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}")))
```

W. ARC Discovery Grant

Main grant

```
(add-to-list 'org-latex-classes
             '("ARC_discovery_grant"
               "\\documentclass[12pt]{article}
\\usepackage{\\string~\"/.emacs.d/latex_templates/ARC_discovery\"}
[NO-DEFAULT-PACKAGES]
[NO-PACKAGES]"
                ("\\section{%s}" . "\\section*{%s}")
                ("\\subsection{%s}" . "\\subsection*{%s}")
                ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                ("\\paragraph{%s}" . "\\paragraph*{%s}")))
```

Special formatting for the ROPE sections.

```
(add-to-list 'org-latex-classes
             '("ARC_ROPE"
               "\\documentclass[12pt]{article}
\\usepackage{\\string~\"/.emacs.d/latex_templates/ARC_discovery_ROPE\"}
[NO-DEFAULT-PACKAGES]
[NO-PACKAGES]"
                ("\\section{%s}" . "\\section*{%s}")
                ("\\subsection{%s}" . "\\subsection*{%s}")
                ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                ("\\paragraph{%s}" . "\\paragraph*{%s}")))
```

X. Nature style paper

```
(add-to-list 'org-latex-classes '("naturedef"
                                  "\\documentclass[fleqn,10pt]{wlscirep}
 [NO-DEFAULT-PACKAGES]
 [PACKAGES]
 [EXTRA]"
                                  ("\\section{%s}" . "\\section*{%s}")
                                  ("\\subsection{%s}" . "\\subsection*{%s}")
                                  ("\\subsubsection{%s}" .
                                  ↪  "\\subsubsection*{%s}")
                                  ("\\paragraph{%s}" . "\\paragraph*{%s}")
                                  ("\\subparagraph{%s}" .
                                  ↪  "\\subparagraph*{%s}")))
```

```
(add-to-list 'org-latex-classes
             '("nature"
               "\\documentclass[12pt]{article}
     \\usepackage{\\string~\"/.emacs.d/latex_templates/nature\"}
     [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}")))
```

Y. Bioinformatics paper

```
(add-to-list 'org-latex-classes '("bioinfo"
                                  "\\documentclass{bioinfo}
 [NO-DEFAULT-PACKAGES]
 [PACKAGES]
 [EXTRA]"
                                  ("\\section{%s}" . "\\section*{%s}")
                                  ("\\subsection{%s}" . "\\subsection*{%s}")
                                  ("\\subsubsection{%s}" .
                                  ↪  "\\subsubsection*{%s}")
                                  ("\\paragraph{%s}" . "\\paragraph*{%s}")
                                  ("\\subparagraph{%s}" .
                                  ↪  "\\subparagraph*{%s}")))
```

Z. Internal report

```
(add-to-list 'org-latex-classes
             '("report"
               "\\documentclass[12pt]{article}
\\usepackage{\\string~\"/.emacs.d/latex_templates/report\"}
[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}")))
```

. RoamCard

```
(add-to-list 'org-latex-classes
             '("roamcard"
               "\\documentclass[12pt,notitlepage]{article}
\\usepackage{\\string~\"/.emacs.d/latex_templates/roamcard\"}
[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}")))
```

. Simple presentation

```
(add-to-list 'org-latex-classes
             '("simplepresentation"
               "\\documentclass[aspectratio=169,18pt,t]{beamer}
\\usepackage{\\string~\"/.emacs.d/latex_templates/simple\"}
[NO-DEFAULT-PACKAGES]
[NO-PACKAGES]"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\begin{frame}[fragile]\\frametitle{%s}"
                "\\end{frame}"
                "\\begin{frame}[fragile]\\frametitle{%s}"
                "\\end{frame}")))
```

. Fancier presentation

```
(add-to-list 'org-latex-classes
             '("modernpresentation"
               "\\documentclass[14pt]{beamer}
    \\usepackage{\\string~\"/.emacs.d/latex_templates/modern\"}
    [NO-DEFAULT-PACKAGES]
    [NO-PACKAGES]"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\begin{frame}[fragile]\\frametitle{%s}"
                "\\end{frame}")))
```

end.

Programming

General programming settings..

A. General

I like shallow indentation, but tabs are displayed as 8 characters by default. This reduces that.

```
(setq-default tab-width 2)
```

Treating terms in CamelCase symbols as separate words makes editing a little easier for me, so I like to use subword-mode everywhere.

```
(global-subword-mode 1)
```

Compilation output goes to the **compilation** buffer. I rarely have that window selected, so the compilation output disappears past the bottom of the window. This automatically scrolls the compilation window so I can always see the output.

```
;;(setq compilation-scroll-output t)
(setq compile-command "make -j 6")
(setq compilation-scroll-output 'first-error)
(setq compilation-always-kill t)
(setq compilation-disable-input t)
(add-hook 'compilation-mode-hook 'visual-line-mode)
```

Flycheck

```
(use-package flycheck

  :init
  (add-hook 'after-init-hook 'global-flycheck-mode)
  (add-hook 'c-mode-hook (lambda () (setq flycheck-clang-language-standard
  ↪  "c11")))
  :config
  (setq-default flycheck-disabled-checkers '(emacs-lisp-checkdoc)))


(use-package flycheck-clang-analyzer

  :after flycheck
  :config (flycheck-clang-analyzer-setup))

(use-package flycheck-clang-tidy

  :after flycheck
  :hook
  (flycheck-mode . flycheck-clang-tidy-setup)
  :config
  (setq-default flycheck-clang-tidy-extra-options
   ↪  "--checks=-*,bugprone-*,cert-*,clang-analyzer-*,darwin-*,linuxkernel-*,misc-*,perf

  )
```

B. Line Numbering

C. Magit

I played with this before..

```
(use-package magit

  :commands magit-status magit-blame
  :init
  (defadvice magit-status (around magit-fullscreen activate)
    (window-configuration-to-register :magit-fullscreen)
    ad-do-it
    (delete-other-windows))
  :config
  (setq magit-branch-arguments nil
        ;; use ido to look for branches
        magit-completing-read-function 'magit-ido-completing-read
        ;; don't put "origin-" in front of new branch names by default
        magit-default-tracking-name-function
        ↪  'magit-default-tracking-name-branch-only
        magit-push-always-verify nil
        ;; Get rid of the previous advice to go into fullscreen
        magit-restnore-window-configuration t)

  :bind ("C-x g" . magit-status))
```

magit end.

D. Yasnippet

```
(use-package yasnippet

  :diminish
  :init
  (yas-global-mode 1)
  :config
  (use-package yasnippet-snippets
    )
  (yas-reload-all));
```

E. Comments

```
(use-package smart-comment

  :bind ("M-;" . smart-comment))
```

F. C

compile

```
(global-set-key (kbd "<f5>") (lambda ()
                               (interactive)
                               (setq-local compilation-read-command nil)
                               (call-interactively 'compile)))
```

Follow compilation

```
(setq compilation-scroll-output t)
```

Indent style

SET BSD indent style

```
(add-hook 'c-mode-hook
          '(lambda()
             (c-set-style "bsd")
             ))
```

ggtags

```
(use-package ggtags

  :init
  (add-hook 'c-mode-common-hook
            (lambda ()
              (when (derived-mode-p 'c-mode)
                (ggtags-mode 1))))
  :config

                                              ; This must be set to the location of
                                              ↪  gtags (global)
  ;;(setq ggtags-executable-directory "~/global-6.5.6/bin/")
                                                ; Allow very large database files
  (setq ggtags-oversize-limit 104857600)
  (setq ggtags-sort-by-nearness t)
  (setq ggtags-use-idutils t)
  (setq ggtags-use-project-gtagsconf nil)

  :bind (
         ;;("M-," . gtags-pop-stack)
         ;; ("M-/" . ggtags-find-reference)
         ;;("M-]" . ggtags-idutils-query)

         :map ggtags-navigation-map
                                                ;Ergo
         ("M-u" . ggtags-navigation-previous-file)
         ("M-o" . ggtags-navigation-next-file)
         ("M-l" . ggtags-navigation-visible-mode)
         ("M-j" . ggtags-navigation-visible-mode)
         ("M-k" . next-error)
         ("M-i" . previous-error)
         ) ; end :bind
  )
```

Counsel-gtags

```
(use-package counsel-gtags

  ;;:bind (
  ;;   ("M-t" . counsel-gtags-find-definition)
  ;; ("M-r" . counsel-gtags-find-reference)
  ;;("M-s" . counsel-gtags-find-symbol)
  ;; ("M-," . counsel-gtags-go-backward)
  ;; )
  :init
  (add-hook 'c-mode-hook 'counsel-gtags-mode)
  (add-hook 'c++-mode-hook 'counsel-gtags-mode)
  )
```

Smartparens

```
(use-package smartparens

  :config
  (sp-pair "'" nil :actions :rem)
  (sp-pair "`" nil :actions :rem)
  :init (add-hook 'c-mode-hook 'smartparens-mode))
```

smart scan

```
(use-package smartscan

  :bind ("M-n" . smartscan-symbol-go-forward)
  ("M-p" . smartscan-symbol-go-backward))
```

semantic

#+EXAMPLE$_{\mathrm{SRC}}$ emacs-lisp (use-package semantic
) (global-semanticdb-minor-mode 1) (global-semantic-idle-scheduler-mode 1)
(semantic-mode 1)
#+EXAMPLE$_{\mathrm{SRC}}$

Whitespace mode

Whitespace mode

```
(global-set-key (kbd "C-c w") 'whitespace-mode)
(add-hook 'prog-mode-hook (lambda () (interactive) (setq
↪  show-trailing-whitespace 1)))
```

clean aindent mode

```
(use-package clean-aindent-mode

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

dtrt indent mode

```
(use-package dtrt-indent

  :init
  (dtrt-indent-mode 1)
  (setq dtrt-indent-verbosity 0)
  )
```

G. Don’t ask for permission to run make

Don’t ask with make command to run…

```
(setq compilation-read-command nil)
```

H. ESS (emacs speaks statistics…) and R

I prefer to have my own R installation locally (see Rinstallation.org). Let’s let emacs know about this:

```
;; (setq-default inferior-R-program-name "/home/user/bin/R")
```

Default ESS config:

```
(use-package ess

  :init (require 'ess-site))
```

I. Snakemake

```
(use-package snakemake-mode

  )
```

Email

A. Mu4e

On a mac install mu via brew:

```
brew install mu --with-emacs --HEAD
```

and make sure the path below points to the same `HEAD` directory!

```
(cond
 ((string-equal system-type "windows-nt") ; Microsoft Windows
  (progn
    (message "Microsoft Windows")))
 ((string-equal system-type "darwin") ; Mac OS X
  (progn
    (add-to-list 'load-path "/usr/local/Cellar/mu/HEAD-7d6c30f/share/emacs/site-lisp/mu/
    (setq mu4e-mu-binary "/usr/local/bin/mu")
    ))
 ((string-equal system-type "gnu/linux") ; linux
  (progn
    ;;  (add-to-list 'load-path "~/programs/mu/mu4e")
    (add-to-list 'load-path "/usr/share/emacs/site-lisp/mu4e")
    (add-to-list 'load-path "/usr/local/share/emacs/site-lisp/mu4e")
    ;;         (setq mu4e-mu-binary "/usr/local/bin/mu")
    )))

(message "Loading Mu4e...")

;;  (add-to-list 'load-path "~/programs/mu/mu4e")

;;         (add-to-list 'load-path "/usr/local/share/emacs/site-lisp/mu/mu4e")

;; the modules
```

```
;;(if (file-exists-p mu4e-mu-binary)
;;    (message "Loading Mu4e...")


(if (not (require 'mu4e nil t))
    (message "`mu4e' not found")

  ;;(require 'mu4e)
  (require 'org-mu4e)

  (setq mu4e-maildir (expand-file-name "~/Maildir"))

  (setq mu4e-sent-folder "/office365/sent")
  (setq mu4e-drafts-folder "/drafts")
  (setq mu4e-refile-folder "/office365/Archive")   ;; saved messages
  (setq mu4e-trash-folder "/office365/trash")


  (setq message-kill-buffer-on-exit t)
  (setq mu4e-change-filenames-when-moving t)
  (setq mu4e-confirm-quit nil)
  (setq mail-user-agent 'mu4e-user-agent)
  (setq mu4e-completing-read-function 'ivy-completing-read)

  (setq mu4e-sent-messages-behavior 'sent)

  (setq mu4e-view-show-addresses t)

  (setq mu4e-attachment-dir "~/Downloads")


  (define-key mu4e-headers-mode-map (kbd "C-c c") 'org-mu4e-store-and-capture)
  (define-key mu4e-view-mode-map (kbd "C-c c") 'org-mu4e-store-and-capture)

  (setq mu4e-get-mail-command "offlineimap")

  (setq mu4e-compose-signature
        "Associate Professor Timo Lassmann
Feilman Fellow
Academic Head of Computational Biology, Telethon Kids Institute
Adjunct Associate Professor, Center for Child Health Research
University of Western Australia

Telethon Kids Institute
Northern Entrance, Perth Children's Hospital
15 Hospital Avenue, Nedlands, Western Australia, 6009
PO Box 855, West Perth, Western Australia, 6872
https://scholar.google.com.au/citations?user=7fZs_tEAAAAJ&hl=en

Visiting Scientist, RIKEN Yokohama Institute, Japan
Division of Genomic Technology,
RIKEN Center for Life Science Technologies,
Yokohama Institute,1-7-22 Suehiro-cho,
Tsurumi-ku, Yokohama, 230-0045 JAPAN")
  )
```

Spell check

```
(add-hook 'mu4e-compose-mode-hook
          'flyspell-mode)
(add-hook 'message-mode-hook 'turn-on-orgtbl)
(add-hook 'message-mode-hook 'turn-on-orgstruct++)
(add-hook 'mu4e-compose-mode-hook 'turn-off-auto-fill)
```

TRAMP

```
(use-package tramp

  :config
  (with-eval-after-load 'tramp-cache
    (setq tramp-persistency-file-name "~/.emacs.d/tramp"))
  (setq tramp-default-method "ssh")
  (setq tramp-use-ssh-controlmaster-options nil)
  (message "tramp-loaded"))
```

```
(use-package ssh

  )
```

Autoinsert templates

Eshell

Set up environment.

```
(setenv "LD_LIBRARY_PATH" "/usr/local/lib")
;;(setenv "PATH"
;;        (concat
;;         "/usr/local/bin:/usr/local/sbin:"
;;         (getenv "PATH")))
```

```
(use-package eshell
  :init
  (setq ;; eshell-buffer-shorthand t ...  Can't see Bug#19391
   eshell-scroll-to-bottom-on-input 'all
   eshell-error-if-no-glob t
   eshell-hist-ignoredups t
   eshell-save-history-on-exit t
   eshell-prefer-lisp-functions nil
   eshell-destroy-buffer-when-process-dies t))
```

```
(use-package eshell
  :init
  (add-hook 'eshell-mode-hook
            (lambda ()
              (add-to-list 'eshell-visual-commands "ssh")
              (add-to-list 'eshell-visual-commands "tail")
              (add-to-list 'eshell-visual-commands "top"))))
```

Alias

```
(add-hook 'eshell-mode-hook (lambda ()
                              (eshell/alias "e" "find-file $1")
                              (eshell/alias "ff" "find-file $1")
                              (eshell/alias "emacs" "find-file $1")
                              (eshell/alias "ee" "find-file-other-window $1")

                              (eshell/alias "gd" "magit-diff-unstaged")
                              (eshell/alias "gds" "magit-diff-staged")
                              (eshell/alias "d" "dired $1")
                              (eshell/alias "val" "valgrind --leak-check=yes
                              ↪  --show-leak-kinds=all
                              ↪  --exit-on-first-error=yes --error-exitcode=1
                              ↪  $*")

                              ;; The 'ls' executable requires the Gnu version
                              ↪  on the Mac
                              (let ((ls (if (file-exists-p
                              ↪  "/usr/local/bin/gls")
                                            "/usr/local/bin/gls"
                                          "/bin/ls")))
                                (eshell/alias "ll" (concat ls " -AlohG
                                ↪  --color=always")))))
```

PDF tools

```
(use-package pdf-tools
  :config
  ;; initialise
  (pdf-tools-install)
  ;; open pdfs scaled to fit page
  (setq-default pdf-view-display-size 'fit-page)
  ;; automatically annotate highlights
  (setq pdf-annot-activate-created-annotations t)
  ;; use normal isearch
  (define-key pdf-view-mode-map (kbd "C-s") 'isearch-forward))
```

```
;; PDF links for org-mode
(with-eval-after-load 'pdf-tools
  (use-package org-pdftools
    :config
    ;;
    ↪  https://lists.gnu.org/archive/html/emacs-orgmode/2016-11/msg00169.html
    ;; Before adding, remove it (to avoid clogging)
    (delete '("\\.pdf\\'" . default) org-file-apps)
    ;;
    ↪  https://lists.gnu.org/archive/html/emacs-orgmode/2016-11/msg00176.html
    (add-to-list 'org-file-apps
     '("\\.pdf\\'" . (lambda (file link)
            (org-pdftools-open link))))))
```

End

Run client

```
(if (daemonp)
    (add-hook 'after-make-frame-functions
              '(lambda (f)
                 (with-selected-frame f
                   (when (window-system f) (require 'init-client) ))))
  (require 'init-client) )
```

```
(require 'init-local nil t)
```

A. Fill Mode

Automatically wrapping when you get to the end of a line (or the fill-region):

```
(use-package fill
  :bind (("C-c T f" . auto-fill-mode)
         ("C-c T t" . toggle-truncate-lines))
  :init (add-hook 'org-mode-hook 'turn-on-auto-fill)
  :diminish auto-fill-mode)
```

End.