Doom Emacs Configuration

Emacs configuration for work and life!

Abdelhak Bougouffa*

February 13, 2022

Contents

*abougouffa@fedoraproject.org

1 Intro

I've been using Linux exclusively since 2010, **GNU Emacs** was always installed on my machine, but I didn't discover the **real** Emacs until 2020, in the beginning, I started my Vanilla Emacs configuration from scratch, but after a while, it becomes a mess. As a new Emacs user, I didn't understand the in the beginning how to optimize my configuration and how to do things correctly. I discovered then Spacemacs, which made things much easier, but it was a little slow, and just after, I found the awesome Doom Emacs, and since, I didn't quit my Emacs screen!

In the beginning, I was basically copying chunks of Emacs Lisp code from the internet, which quickly becomes a mess, specially because I was using a mixture of vanilla Emacs style configurations and Doom style ones.

Now I decided to rewrite a cleaner version of my configuration which will be more Doom friendly, and for that, I found an excellent example in *tecosaur*'s emacs-config, so my current configuration is heavily inspired by *tecosaur*'s.

1.1 This file

This is my literate configuration file, I use it to generate Doom's config files (`$DOOMDIR/init.el`, `$DOOMDIR/packages.el` and `$DOOMDIR/config.el`), as well as some other shell scripts, app installers, app launchers... etc.

Make `config.el` run (slightly) faster with lexical binding (see this blog post for more info).

```
;;; config.el -*- lexical-binding: t; -*-
```

Add the shebang and the description to the `setup.sh` file, which will be used to set system settings and install some missing dependencies.

Add the shebang to the `~/.env_stuff` file used to define some aliases and helpers. This needs to be sourced in the shell session (source it in `~/.zshrc`).

2 General Settings

2.1 User information

```
(setq user-full-name "Abdelhak Bougouffa"
      user-mail-address "abougouffa@fedoraproject.org")
```

2.2 Secrets

Set the path to my GPG encrypted secrets. I like to set the cache expiry to `nil` instead of the default 2h.

```
(setq auth-sources '("~/.authinfo.gpg")
      auth-source-cache-expiry nil ; defaut is 2h (7200)
      password-cache-expiry nil)
```

2.3 Better defaults

2.3.1 File deletion

Delete files by moving them to trash.

```
(setq-default delete-by-moving-to-trash t)
```

2.3.2 Window

Take new window space from all other windows (not just current).

```
(setq-default window-combination-resize t)
```

Split Split horizontally to right, vertically below the current window.

```
(setq evil-vsplit-window-right t
      evil-split-window-below t)
```

Show list of buffers when splitting.

```
(defadvice! prompt-for-buffer (&rest _)
  :after '(evil-window-split evil-window-vsplit)
  (consult-buffer))
```

2.3.3 Undo and auto-save

```
(setq undo-limit 80000000   ; Raise undo-limit to 80Mb
      evil-want-fine-undo t ; By default while in insert all changes are one big blob. Be more granular
      auto-save-default t   ; Nobody likes to lose work, I certainly don't
      scroll-preserve-screen-position 'always ; Don't have `point' jump around
      scroll-margin 2)      ; It's nice to maintain a little margin
```

2.3.4 Editing

```
;; Stretch cursor to the glyph width
(setq-default x-stretch-cursor t)

;; Enable relative line numbers
(setq display-line-numbers-type 'relative)

;; Iterate through CamelCase words
(global-subword-mode 1)
```

2.3.5 Frame

Maximizing

```
;; NOTE: Not tangled, replaced with params passed to emacsclient
;; start the initial frame maximized
(add-to-list 'initial-frame-alist '(fullscreen . maximized))

;; start every frame maximized
(add-to-list 'default-frame-alist '(fullscreen . maximized))
```

To avoid conflict when launching Emacs in `emacs-everywhere` mode. I'm using it in command line when calling `emacsclient`, by adding this:

```
--frame-parameters="'(fullscreen . maximized)"
```

Focus created frame The problem is, every time I launch an Emacs frame (from KDE), The Emacs starts with no focus, I need each time to `Alt-TAB` to get Emacs under focus, and then start typing. I tried changing this behavior from Emacs by hooking `raise-frame` at startup, but it didn't work.

Got from this comment, not working on my Emacs version.

```
;; NOTE: Not tangled, not working
(add-hook 'server-switch-hook #'raise-frame)
```

After some investigations, I found that this issue is probably KDE specific, the issue goes away by setting: **Window Management** > **Window Behavior** > **Focus** > **Focus stealing prevention** to *None* in the KDE Settings.

2.4 Debug

```
;; NOTE: Not tangled, toggle to enable doom debugging, I do enable it to see
;;       which packages are loaded automatically to optimize launch time of my config.
;; The `use-package-verbose' takes the value of `doom-debug-p'.
(setq doom-debug-p t)
```

3 Modules (`init.el`)

Here is the literate configuration which generates the Doom's `init.el` file, this file contains all the enabled Doom modules with the appropriate flags.

This section defines the default source blocks arguments . All source blocks in this section inherits these headers, so they will not be tangled unless overwriting in the block's header.

3.1 File skeleton

This first section defines the template for the subsections, it uses the `no-web` syntax to include subsections specified as .

```
;;; init.el -*- lexical-binding: t; -*-

;; This file controls what Doom modules are enabled and what order they load in.
;; Press 'K' on a module to view its documentation, and 'gd' to browse its directory.

(doom! :completion
       <<doom-completion>>

       :ui
       <<doom-ui>>

       :editor
       <<doom-editor>>

       :emacs
       <<doom-emacs>>

       :term
       <<doom-term>>

       :checkers
       <<doom-checkers>>

       :tools
       <<doom-tools>>

       :os
       <<doom-os>>

       :lang
       <<doom-lang>>

       :email
       <<doom-email>>

       :app
       <<doom-app>>
```

```
       :config
       <<doom-config>>
)
```

3.2 Config (:config)

Enable `literate` configuration (like this file!), and some defaults.

```
literate
(default +bindings
         +smartparens)
```

3.3 Completion (:completion)

I'm lazy, I like Emacs to complete my writings.

```
(company +childframe)          ; the ultimate code completion backend
(vertico +icons)               ; the search engine of the future
;;(ivy +childframe            ; a search engine for love and life
;;     +fuzzy
;;     +icons
;;     +prescient)
;;helm                         ; the *other* search engine for love and life
;;ido                          ; the other *other* search engine...
```

3.4 User interface (:ui)

Enables some user interface features for better user experience, the beautiful `modeline`, the `treemacs` project tree, better version control integration with `vc-gutter`... and other useful stuff.

```
deft                           ; notational velocity for Emacs
doom                           ; what makes DOOM look the way it does
doom-dashboard                 ; a nifty splash screen for Emacs
;;doom-quit                    ; DOOM quit-message prompts when you quit Emacs
(emoji +ascii
       +unicode
       +github)
hl-todo                        ; highlight TODO/FIXME/NOTE/DEPRECATED/HACK/REVIEW
;;fill-column                  ; a `fill-column' indicator
hydra                          ; quick documentation for related commands
;;indent-guides                ; highlighted indent columns, notoriously slow
(ligatures +extra)             ; ligatures and symbols to make your code pretty again
;;minimap                      ; show a map of the code on the side
modeline                       ; snazzy, Atom-inspired modeline, plus API
nav-flash                      ; blink the current line after jumping
;;neotree                      ; a project drawer, like NERDTree for vim
ophints                        ; highlight the region an operation acts on
(popup +all                    ; tame sudden yet inevitable temporary windows
       +defaults)
;;tabs                         ; a tab bar for Emacs
(treemacs +lsp)                ; a project drawer, like neotree but cooler
;;unicode                      ; extended unicode support for various languages
vc-gutter                      ; vcs diff in the fringe
;;vi-tilde-fringe              ; fringe tildes to mark beyond EOB
(window-select +numbers)       ; visually switch windows
workspaces                     ; tab emulation, persistence & separate workspaces
zen                            ; distraction-free coding or writing
```

3.5 Editor (:editor)

Some editing modules, the most important feature is EVIL to enable Vim style editing in Emacs. I like also to edit with multiple cursors, enable `yasnippet` support, wrap long lines, auto format support (however, I don't enable `+onsave` flag even if I like to, I'm experiencing an annoying behavior when I use it with projects that defines `.editorconfig` rules, the formatter do not respect that, nor the `clang-format` rules, I need to fix this).

```
(evil +everywhere)          ; come to the dark side, we have cookies
file-templates              ; auto-snippets for empty files
fold                        ; (nigh) universal code folding
format                      ; automated prettiness
;;god                       ; run Emacs commands without modifier keys
;;lispy                     ; vim for lisp, for people who don't like vim
multiple-cursors            ; editing in many places at once
(objed +manual)             ; text object editing for the innocent
;;parinfer                  ; turn lisp into python, sort of
;;rotate-text               ; cycle region at point between text candidates
snippets                    ; my elves. They type, so I don't have to
word-wrap                   ; soft wrapping with language-aware indent
```

3.6 Emacs' builtin (:emacs)

Beautify Emacs builtin packages.

```
(dired +icons               ; making dired pretty [functional]
       +ranger)
electric                    ; smarter, keyword-based electric-indent
(ibuffer +icons)            ; interactive buffer management
(undo +tree)                ; persistent, smarter undo for your inevitable mistakes
vc                          ; version-control and Emacs, sitting in a tree
```

3.7 Terminals (:term)

Run commands in terminal from Emacs. I use mainly `vterm` on my local machine, however, I like to have `eshell`, `shell` and `term` installed to use them for remote file editing (via Tramp).

```
eshell                      ; the elisp shell that works everywhere
vterm                       ; the best terminal emulation in Emacs
shell                       ; simple shell REPL for Emacs
term                        ; basic terminal emulator for Emacs
```

3.8 Checkers (:checkers)

I like to check my documents for errors while I'm typing, however, sometimes it makes Emacs runs slowly, specially on big files, so I will disable checking by default, and I enable it when I need to.

```
(syntax +childframe)    ; tasing you for every semicolon you forget
(spell +flyspell        ; tasing you for misspelling mispelling
       +hunspell)
grammar                 ; tasing grammar mistake every you make
```

3.9 Tools (:tools)

I enable some useful tools which facilitate my work flow, I like to enable Docker support, EditorConfig is a good feature to have. I like to enable `lsp-mode` and `dap-mode` for coding and debugging by enabling the `lsp` and `debugger` modules with `+lsp` support (further customization for `lsp` and `dap` below). `pdf` adds support through `pdf-tools`, which are great for viewing PDF files inside Emacs, I also enable some extra tools, like `magit`, `lookup`, `tmux`... etc.

```
;;ansible
(debugger +lsp)         ; FIXME stepping through code, to help you add bugs
direnv
(docker +lsp)
editorconfig            ; let someone else argue about tabs vs spaces
ein                     ; tame Jupyter notebooks with emacs
(eval +overlay)         ; run code, run (also, repls)
biblio
gist                    ; interacting with github gists
(lookup +docsets)       ; navigate your code and its documentation
(lsp +peek)             ; LPS
(magit +forge)          ; a git porcelain for Emacs
make                    ; run make tasks from Emacs
;;pass                  ; password manager for nerds
pdf                     ; pdf enhancements
;;prodigy               ; FIXME managing external services & code builders
rgb                     ; creating color strings
;;taskrunner            ; taskrunner for all your projects
;;terraform             ; infrastructure as code
tmux                    ; an API for interacting with tmux
upload                  ; map local to remote projects via ssh/ftp
```

3.10 Operating system (:os)

I enable `tty` for better support of terminal editing.

```
(tty +osc)              ; Configures Emacs for use in the terminal
```

3.11 Language support (:lang)

Most of the projects I'm working on are mainly written in C/C++, Python, Rust and some Lisp stuff, I edit also a lot of configuration and data files in several formats (`csv`, `yaml`, `xml`, `json`, `shell` scripts...). I use Org-mode to manage all my papers and notes, so I need to enable as many features as I need, I do enable `plantuml` also to quickly plot UML models withing Org documents.

```
plantuml                ; diagrams for confusing people more
emacs-lisp              ; drown in parentheses
common-lisp             ; if you've seen one lisp, you've seen them all
markdown                ; writing docs for people to ignore
;;rst                   ; ReST in peace
data                    ; config/data formats
;;qt                    ; the 'cutest' gui framework ever
(cc +lsp)               ; C/C++/Obj-C madness
(json +lsp)             ; At least it ain't XML
(julia +lsp)            ; a better, faster MATLAB
(latex +lsp)            ; writing papers in Emacs has never been so fun
;;(lua +lsp)           ; one-based indices? one-based indices
(rust +lsp)             ; Fe203.unwrap().unwrap().unwrap().unwrap()
(ess +lsp)              ; emacs speaks statistics
(yaml +lsp)             ; JSON, but readable
(sh +lsp)               ; she sells {ba,z,fi}sh shells on the C xor
(python +lsp            ; beautiful is better than ugly
        +pyright
        +pyenv
        +conda)
(org +dragndrop         ; organize your plain life in plain text
     +gnuplot
```

```
     +jupyter
     +pandoc
     +present
     +pomodoro
     +roam2
     +pretty)
(racket +lsp           ; a DSL for DSLs
        +xp)
(scheme +mit           ; a fully conniving family of lisps
        +guile
        +racket
        +chez)
;;agda                ; types of types of types of types...
;;(clojure +lsp)      ; java with a lisp
;;coq                 ; proofs-as-programs
;;crystal             ; ruby at the speed of c
;;csharp              ; unity, .NET, and mono shenanigans
;;(dart +flutter)     ; paint ui and not much else
;;elixir              ; erlang done right
;;elm                 ; care for a cup of TEA?
;;erlang              ; an elegant language for a more civilized age
;;faust               ; dsp, but you get to keep your soul
;;fsharp              ; ML stands for Microsoft's Language
;;fstar               ; (dependent) types and (monadic) effects and Z3
;;gdscript            ; the language you waited for
;;(go +lsp)           ; the hipster dialect
;;(haskell +dante)    ; a language that's lazier than I am
;;hy                  ; readability of scheme w/ speed of python
;;idris               ;
;;(java +meghanada)   ; the poster child for carpal tunnel syndrome
;;javascript          ; all(hope(abandon(ye(who(enter(here))))))
;;kotlin              ; a better, slicker Java(Script)
;;lean
;;factor
;;ledger              ; an accounting system in Emacs
;;nim                 ; python + lisp at the speed of c
;;nix                 ; I hereby declare "nix geht mehr!"
;;ocaml               ; an objective camel
;;php                 ; perl's insecure younger brother
;;purescript          ; javascript, but functional
;;raku                ; the artist formerly known as perl6
;;rest                ; Emacs as a REST client
;;(ruby +rails)       ; 1.step {|i| p "Ruby is #{i.even? ? 'love' : 'life'}"}
;;scala               ; java, but good
;;sml
;;solidity            ; do you need a blockchain? No.
;;swift               ; who asked for emoji variables?
;;terra               ; Earth and Moon in alignment for performance.
;;web                 ; the tubes
```

3.12 Email (:email)

I like to use `mu4e` to manage mail mailboxes. The `+org` flag adds `org-msg` support and `+gmail` adds better management of Gmail accounts.

```
(mu4e +org
      +gmail)
```

```
;; (notmuch +org
;;          +afew)
;; (wanderlust +gmail)
```

3.13 Apps (:app)

Emacs contains a ton of applications, some of them are supported by Doom, I like to use Emacs manage my calendar, chat on IRC, and receive news. I do use EMMS sometimes to play music without leaving Emacs, and I like to enable support for `emacs-everywhere`.

```
calendar
irc                   ; how neckbeards socialize
;;emms
everywhere
(rss +org)            ; emacs as an RSS reader
;;twitter             ; twitter client https://twitter.com/vnought
```

4 Doom Configuration

4.1 User Interface

4.1.1 Font Face

Doom exposes five (optional) variables for controlling fonts in Doom. Here are the three important ones: `doom-font`, `doom-unicode-font` and `doom-variable-pitch-font`. The `doom-big-font` is used for `doom-big-font-mode`; use this for presentations or streaming.

They all accept either a `font-spec`, font string (`"Input Mono-12"`), or xlfd font string. You generally only need these two:

Some good fonts:

- `Iosevka Fixed` (THE FONT)
- `Cascadia Code`
- `JuliaMono` (good Unicode support)
- `mononoki Nerd Font Mono` (good Unicode support)
- `IBM Plex Mono`
- `JetBrains Mono`
- `Roboto Mono`
- `Source Code Pro`
- `Input Mono Narrow`
- `Fira Code`

```
(setq doom-font (font-spec :family "Iosevka Fixed" :size 16)
      doom-variable-pitch-font (font-spec :family "Iosevka Fixed") ; inherits the :size from doom-font
      doom-unicode-font (font-spec :family "Iosevka Fixed")
      doom-serif-font (font-spec :family "Iosevka Fixed" :weight 'light))
```

4.1.2 Theme

Set Doom's theme, some good choices:

- `doom-palenight`
- `doom-one`
- `doom-vibrant`
- `doom-dark+` (VS Code like)
- `doom-tomorrow-night`
- `doom-xcode`
- `doom-material`
- `doom-ayu-mirage`
- `doom-monokai-pro`

```
(setq doom-theme 'doom-one) ; Load theme
```

4.1.3 Mode line

Clock Display time and set the format to 24h.

```
;; NOTE: Not tangled
(setq display-time-string-forms
      '((propertize (concat 24-hours ":" minutes))))

(display-time-mode 1) ; Enable time in the mode-line
```

Battery Show battery level unless battery is not present or battery information is unknown.

```
;; NOTE: Not tangled
;; This code causes 'doom doctor' to fail. TODO: What's wrong with this function?
(defun ab/display-battery ()
  (let ((batt-status (battery)))
    (unless (or (string-match-p "unknown"    batt-status)
                (string-match-p "^Power N/A" batt-status))
      (display-battery-mode 1)))) ; it's nice to know how much power you have

(ab/display-battery)
```

4.1.4 Set transparency

```
;; NOTE: Not tangled
(set-frame-parameter (selected-frame) 'alpha '(98 100))
(add-to-list 'default-frame-alist '(alpha 98 100))
```

4.1.5 Splash Screen

Custom Splash Image Change the logo to a fancy black hole, form this GitHub thread

```
;; (setq fancy-splash-image (expand-file-name "assets/gnu-emacs-logo-flat-light.svg" doom-private-dir))
;; (setq fancy-splash-image (expand-file-name "assets/blackhole-lines-small.svg" doom-private-dir))
;; (setq fancy-splash-image (expand-file-name "assets/gnu-emacs-logo-flat-white.svg" doom-private-dir))
(setq fancy-splash-image (expand-file-name "assets/emacs-e-big.svg" doom-private-dir))
```

Clean Screen Let's disable the dashboard commands, for a particularly *clean* look disable the modeline and `hl-line-mode`, then also hide the cursor.

```
(remove-hook '+doom-dashboard-functions #'doom-dashboard-widget-shortmenu)
(add-hook!   '+doom-dashboard-mode-hook (hide-mode-line-mode 1) (hl-line-mode -1))
(setq-hook!  '+doom-dashboard-mode-hook evil-normal-state-cursor (list nil))
```

The ASCII Banner Add an ASCII banner, used in terminal mode.

```
(defun doom-dashboard-draw-ascii-emacs-banner-fn ()
  (let* ((banner
          '("______  _____  _____ ___  ___"
            "|  _  \|  _  ||  _  ||  \/  |"
            "| | | || | | || | | || .  . |"
            "| | | || | | || | | || |\/| |"
            "| |/ / \ \_/ /\ \_/ /| |  | |"
            "|___/   \___/  \___/ \_|  |_/"))
         (longest-line (apply #'max (mapcar #'length banner))))
    (put-text-property
     (point)
     (dolist (line banner (point))
       (insert (+doom-dashboard--center
                +doom-dashboard--width
                (concat line (make-string (max 0 (- longest-line (length line))) 32))))
       "\n")
     'face 'doom-dashboard-banner)))

(unless (display-graphic-p) ; for some reason this messes up the graphical splash screen atm
  (setq +doom-dashboard-ascii-banner-fn #'doom-dashboard-draw-ascii-emacs-banner-fn))
```

4.1.6 Which key

Make `which-key` popup faster.

```
(setq which-key-idle-delay 0.5 ;; Default is 1.0
      which-key-idle-secondary-delay 0.05) ;; Default is nil
```

4.2 Editor

4.2.1 Scratch buffer

Tell the scratch buffer to start in `emacs-lisp-mode`.

```
(setq doom-scratch-initial-major-mode 'emacs-lisp-mode)
```

4.2.2 Mouse Buttons

Map extra mouse buttons to jump between buffers

```
(map! :n [mouse-8] #'better-jumper-jump-backward
      :n [mouse-9] #'better-jumper-jump-forward)
```

4.2.3 Binary files

Taken from this answer.

```
(defun buffer-binary-p (&optional buffer)
  "Return whether BUFFER or the current buffer is binary.

A binary buffer is defined as containing at least one null byte.

Returns either nil, or the position of the first null byte."
  (with-current-buffer (or buffer (current-buffer))
    (save-excursion
      (goto-char (point-min))
      (search-forward (string ?\x00) nil t 1))))

(defun hexl-if-binary ()
  "If `hexl-mode' is not already active, and the current buffer
is binary, activate `hexl-mode'."
  (interactive)
  (unless (eq major-mode 'hexl-mode)
    (when (buffer-binary-p)
      (hexl-mode))))

(add-to-list 'magic-fallback-mode-alist '(buffer-binary-p . hexl-mode) t)
```

4.3 Allow babel execution in `doom` CLI actions

This file generates all my Doom config files, it works nicely, but for it to work with `doom sync` et al. I need to make sure that Org doesn't try to confirm that I want to allow evaluation (I do!).

Thankfully Doom supports `$DOOMDIR/cli.el` file which is sourced every time a CLI command is run, so we can just enable evaluation by setting `org-confirm-babel-evaluate` to `nil` there.

While we're at it, we should silence `org-babel-execute-src-block` to avoid polluting the output.

```
;;; cli.el -*- lexical-binding: t; -*-
(setq org-confirm-babel-evaluate nil)

(defun doom-shut-up-a (orig-fn &rest args)
  (quiet! (apply orig-fn args)))

(advice-add 'org-babel-execute-src-block :around #'doom-shut-up-a)
```

4.4 Asynchronous config tangling

Doom adds an `org-mode` hook `+literate-enable-recompile-h`. This is a nice idea, but it's too blocking for my taste. Since I trust my tangling to be fairly straightforward, I'll just redefine it to a simpler, async, function.

```
(defadvice! +literate-tangle-async-h ()
  "A very simplified version of `+literate-tangle-h', but async."
  :override #'+literate-tangle-h
  (let ((default-directory doom-private-dir))
    (async-shell-command
     (format "emacs --batch --eval \"(progn \
(require 'org) (setq org-confirm-babel-evaluate nil) \
(org-babel-tangle-file \\\"%s\\\"))\""
             +literate-config-file))))
```

5 Additional Packages (`packages.el`)

This file shouldn't be byte compiled.

```
;; -*- no-byte-compile: t; -*-
```

5.1 General Packages

5.1.1 Weather

```
;; lisp/wttrin/wttrin.el is taken from:
;; https://raw.githubusercontent.com/tecosaur/emacs-config/master/lisp/wttrin/wttrin.el
(package! wttrin
  :recipe (:local-repo "lisp/wttrin"))

(use-package! wttrin
  :commands wttrin)
```

5.1.2 TODO CalDAV

```
(package! caldav
  :recipe (:host github
           :repo "dengste/org-caldav"))
```

5.2 Themes and UI

5.2.1 SVG Tag Mode

```
(package! svg-tag-mode)

(use-package! svg-tag-mode
  :commands svg-tag-mode)
```

5.2.2 Bespoke themes

```
(package! bespoke-themes
  :recipe (:host github
           :repo "mclear-tools/bespoke-themes"))

(package! bespoke-modeline
  :recipe (:host github
           :repo "mclear-tools/bespoke-modeline"))
```

5.2.3 Focus

Dim the font color of text in surrounding paragraphs, focus only on the current line.

```
(package! focus)

(use-package! focus
  :commands focus-mode)
```

5.2.4 Window title

I'd like to have just the buffer name, then if applicable the project folder

```
(setq frame-title-format
      '(""
        (:eval
         (if (s-contains-p org-roam-directory (or buffer-file-name ""))
             (replace-regexp-in-string
              ".*/[0-9]*-?" " "
              (subst-char-in-string ?_ ?  buffer-file-name))
           "%b"))
        (:eval
         (let ((project-name (projectile-project-name)))
           (unless (string= "-" project-name)
             (format (if (buffer-modified-p) "  %s" "    %s") project-name))))))
```

5.3 Features

5.3.1 ESS

View data frames better with

```
(package! ess-view)
```

5.3.2 Very large files

The *very large files* mode loads large files in chunks, allowing one to open ridiculously large files.

```
(package! vlf)
```

To make VLF available without delaying startup, we'll just load it in quiet moments.

```
(use-package! vlf-setup
  :defer-incrementally vlf-tune vlf-base vlf-write vlf-search vlf-occur vlf-follow vlf-ediff vlf)
```

5.3.3 Ebook reading

Then for reading them, the only currently viable options seem to be nov.el.

```
(package! nov
  :pin "b3c7cc28e95fe25ce7b443e5f49e2e45360944a3")
```

Together these should give me a rather good experience reading ebooks.

5.3.4 Org related

```
(package! doct)
(package! org-ref)
(package! org-super-agenda)
(package! org-fragtog)
(package! academic-phrases
  :recipe (:host github
           :repo "nashamri/academic-phrases"))

;; BUG: Not tangled, it seems to be too slow, + an issue with dvipng
(use-package! org-fragtog
  :hook (org-mode . org-fragtog-mode))
```

5.3.5 Info colors

Better colors for manual pages.

```
(package! info-colors)

(use-package! info-colors
  :commands (info-colors-fontify-node))

(add-hook 'Info-selection-hook 'info-colors-fontify-node)
```

5.3.6 Selectric mode

Selectric Every so often, you want everyone else to know that you're typing, or just to amuse oneself. Introducing: typewriter sounds!

```
(package! selectric-mode)

(use-package! selectic-mode
  :commands selectic-mode)
```

5.3.7 Grammarly

```
(package! grammarly
  :recipe (:host github
           :repo "emacs-grammarly/grammarly")
  :disable t) ;; TODO: It messes my org files up, need to investigate

(package! flycheck-grammarly
  :recipe (:host github
           :repo "emacs-grammarly/flycheck-grammarly")
  :disable t) ;; TODO: It messes my org files up, need to investigate

(use-package! flycheck-grammarly
  :config (load! "lisp/private/+grammarly-account.el"))
```

5.4 Programming

5.4.1 Repo

Make sure the repo tool is installed, if not `pacman -S repo` on Arch-based distros, or directly with:

```
REPO_PATH="$HOME/.local/bin/repo"
curl "https://storage.googleapis.com/git-repo-downloads/repo" > ${REPO_PATH}
chmod a+x ${REPO_PATH}

(package! repo)

(use-package! repo
  :commands repo-status)
```

5.4.2 Devdocs

```
(package! devdocs
  :recipe (:host github
           :repo "astoff/devdocs.el"
           :files ("*.el")))

(use-package! devdocs
  :commands (devdocs-lookup devdocs-install)
  :config
  (setq devdocs-data-dir (expand-file-name "devdocs" doom-etc-dir)))
```

5.4.3 Emacs GDB

Use `emacs-gdb` instead of builtin `gdb-mi`.

```
(package! gdb-mi
  :recipe (:host github
           :repo "weirdNox/emacs-gdb"
           :files ("*.el" "*.c" "*.h" "Makefile")))

(use-package! gdb-mi
  :init
  (fmakunbound 'gdb)
  (fmakunbound 'gdb-enable-debug)
  :config
  (setq ;; gdb-window-setup-function #'gdb--setup-windows ; Customize this
        gdb-ignore-gdbinit nil)) ; I use gdbinit to define some useful stuff
```

5.4.4 Magit Delta

```
(package! magit-delta)
```

```
(use-package! magit-delta
  :commands magit-status
  :hook (magit-mode . magit-delta-mode))
```

5.4.5 Systemd

For editing systemd unit files

```
(package! systemd)
```

5.4.6 Bitbake (Yocto)

```
;; See https://bitbucket.org/olanilsson/bitbake-modes also
(package! bitbake)
```

```
(use-package bitbake
  :commands (bitbake-mode bitbake-clean bitbake-fetch))
```

5.4.7 Org Roam

Org-roam is nice by itself, but there are so *extra* nice packages which integrate with it.

```
(package! websocket)
(package! org-roam-ui
  :recipe (:host github
           :repo "org-roam/org-roam-ui"
           :files ("*.el" "out")))
```

```
(use-package! websocket
  :after org-roam-ui)

(use-package! org-roam-ui
  :commands org-roam-ui-open
  :config (setq org-roam-ui-sync-theme t
                org-roam-ui-follow t
                org-roam-ui-update-on-save t
                org-roam-ui-open-on-start t))
```

5.4.8 LaTeX

For mathematical convenience, WIP

```
(package! aas
  :recipe (:host github
           :repo "ymarco/auto-activating-snippets"))
```

And some basic config

```
(use-package! aas
  :commands aas-mode)
```

5.4.9 Franca IDL

```
(package! franca-idl
  :recipe (:host github
           :repo "zeph1e/franca-idl.el"))
```

```
(use-package! franca-idl
  :commands franca-idl-mode)
```

5.4.10 Flycheck + projectile

```
(package! flycheck-projectile
  :recipe (:host github
           :repo "nbfalcon/flycheck-projectile"))

(use-package! flycheck-projectile
  :commands flycheck-projectile-list-errors
;  :config
;  (set-popup-rule! "^\\*Flycheck errors\\*$" :side 'bottom :size 0.4 :select t)
)
```

5.4.11 Graphviz

Graphviz is a nice method of visualizing simple graphs, based on plaintext `.dot` / `.gv` files.

```
(package! graphviz-dot-mode)

(use-package! graphviz-dot-mode
  :commands (graphviz-dot-mode graphviz-dot-preview))
```

5.4.12 Maxima

I've been experiencing an annoying bug with the `maxima` package, when I launch the interpreter, it complains about a file not being found, the package searches in `~/.emacs-doom/.local/straight/repos-28.0.90/maxima/keywords/f` instead of `~/.../build-28.0.90/...`, when I investigated this issue, it turned out that, the package (in the `maxima-font-lock.el` file), replaces `build` with `repos` when it sees `straight` in the path (as a workaround to support `straight`), however, Doom Emacs redefine the value of `straight-build-dir` to be `build-<emacs-version>„` so the workaround is no more needed. To fix this, I set the `maxima-font-lock-keywords-directory` with the right path.

```
(package! imaxima)
(package! maxima
  :recipe (:host gitlab
           :repo "sasanidas/maxima"
           :files ("*.el" "keywords")))

(use-package! maxima
  :init
  (require 'org) ;; to use `org-format-latex-options'
  (add-hook 'maxima-mode-hook #'maxima-hook-function)
  (add-hook 'maxima-inferior-mode-hook #'maxima-hook-function)
  (setq maxima-font-lock-keywords-directory
        (expand-file-name "~/.emacs-doom/.local/straight/build-28.0.91/maxima/keywords"))
  (setq maxima-display-maxima-buffer nil)
  ;; (require 'org)
  ;; (setq org-format-latex-options (plist-put org-format-latex-options :scale 2.0))
  :commands (maxima-mode maxima)
  :config
  (require 'company-maxima)
  (add-to-list 'company-backends '(company-maxima-symbols company-maxima-libraries))
  :mode ("\\.mac\\'" . maxima-mode)
  :interpreter ("maxima" . maxima-mode))

(autoload 'imaxima "imaxima" "Frontend of Maxima CAS" t)
(autoload 'imath "imath" "Interactive Math mode" t)
(autoload 'imath-mode "imath" "Interactive Math mode" t)
```

5.4.13 TODO ROS

Check code-iai/ros_emacs_utils for the `rosemacs` integration.

6 Emacs Daemon

6.1 Initialization

When the daemon is running, I almost always want to do a few particular things with it, so I may as well eat the load time at startup. We also want to keep `mu4e` running.

Lastly, while I'm not sure quite why it happens, but after a bit it seems that new Emacs client frames start on the `*scratch*` buffer instead of the dashboard. I prefer the dashboard, so let's ensure that's always switched to in new frames.

```
(defun greedy-on-daemon-startup ()
  (require 'org)
  (when (require 'mu4e nil t)
    (setq mu4e-confirm-quit t)
    (setq +mu4e-lock-greedy t)
    (setq +mu4e-lock-relaxed t)
    (mu4e~start))
  (when (require 'elfeed nil t)
    (run-at-time nil (* 8 60 60) #'elfeed-update)))

(when (daemonp)
  (add-hook  'emacs-startup-hook #'greedy-on-daemon-startup)
  (add-hook! 'server-after-make-frame-hook (doom/reload-theme))
  (add-hook! 'server-after-make-frame-hook
    (unless (string-match-p "\\*draft" (buffer-name))
      (switch-to-buffer +doom-dashboard-name))))
```

6.2 Tweaks

6.2.1 Save recent files

When editing files with Emacs client, the files does not get stored by `recentf`, making Emacs forgets about recently opened files. A quick fix is to hook the `recentf-save-list` command to the `delete-frame-functions` and `delete-terminal-functions` which gets executed each time a frame/terminal is deleted.

```
(when (daemonp)
  (add-hook! '(delete-frame-functions delete-terminal-functions) #'(lambda (arg) (recentf-save-list))))
```

7 Package configuration

7.1 All the icons

Set some custom icons for some file extensions, basically for `.m` files.

```
(after! all-the-icons
  (setcdr (assoc "m" all-the-icons-extension-icon-alist)
          (cdr (assoc "matlab" all-the-icons-extension-icon-alist))))
```

7.2 Centaur tabs

A 'active-bar' is nice, so let's have one of those. If we have it `under` needs us to turn on `x-underline-at-decent` though. For some reason this didn't seem to work inside the `(after! ... )` block ¯_()_/¯.

```
(after! centaur-tabs
  (centaur-tabs-mode -1)
  (setq centaur-tabs-set-icons t
        centaur-tabs-modified-marker ""
        centaur-tabs-close-button "×"
        centaur-tabs-gray-out-icons 'buffer))
```

7.3 Better PDFs in Modeline

First up I'm going to want a segment for just the buffer file name, and a PDF icon. Then we'll redefine two functions used to generate the modeline.

```
(after! doom-modeline
  (doom-modeline-def-segment buffer-name
    "Display the current buffer's name, without any other information."
    (concat
     (doom-modeline-spc)
     (doom-modeline--buffer-name)))

  (doom-modeline-def-segment pdf-icon
    "PDF icon from all-the-icons."
    (concat
     (doom-modeline-spc)
     (doom-modeline-icon 'octicon "file-pdf" nil nil
                         :face (if (doom-modeline--active)
                                   'all-the-icons-red
                                   'mode-line-inactive)
                         :v-adjust 0.02)))

  (defun doom-modeline-update-pdf-pages ()
    "Update PDF pages."
    (setq doom-modeline--pdf-pages
          (let ((current-page-str (number-to-string (eval `(pdf-view-current-page))))
                (total-page-str (number-to-string (pdf-cache-number-of-pages))))
            (concat
             (propertize
              (concat (make-string (- (length total-page-str) (length current-page-str)) ? )
                      " P" current-page-str)
              'face 'mode-line)
             (propertize (concat "/" total-page-str) 'face 'doom-modeline-buffer-minor-mode)))))

  (doom-modeline-def-segment pdf-pages
    "Display PDF pages."
    (if (doom-modeline--active) doom-modeline--pdf-pages
      (propertize doom-modeline--pdf-pages 'face 'mode-line-inactive)))

  (doom-modeline-def-modeline 'pdf
    '(bar window-number pdf-pages pdf-icon buffer-name)
    '(misc-info matches major-mode process vcs)))
```

7.4 Emojify

For starters, twitter's emojis look nicer than emoji-one. Other than that, this is pretty great OOTB .

```
(setq emojify-emoji-set "twemoji-v2")
```

One minor annoyance is the use of emojis over the default character when the default is actually preferred. This occurs with overlay symbols I use in Org mode, such as checkbox state, and a few other miscellaneous cases.

We can accommodate our preferences by deleting those entries from the emoji hash table

```
(defvar emojify-disabled-emojis
  '(;; Org
    "" "" "" "" "" "" "" "" ""
    ;; Terminal powerline
    ""
    ;; Box drawing
    "" "")
  "Characters that should never be affected by `emojify-mode'.")

(defadvice! emojify-delete-from-data ()
  "Ensure `emojify-disabled-emojis' don't appear in `emojify-emojis'."
  :after #'emojify-set-emoji-data
  (dolist (emoji emojify-disabled-emojis)
    (remhash emoji emojify-emojis)))
```

Now, it would be good to have a minor mode which allowed you to type ascii/gh emojis and get them converted to unicode. Let's make one.

```
(defun emojify--replace-text-with-emoji (orig-fn emoji text buffer start end &optional target)
  "Modify `emojify--propertize-text-for-emoji' to replace ascii/github emoticons with unicode emojis, o
  (if (or (not emoticon-to-emoji) (= 1 (length text)))
      (funcall orig-fn emoji text buffer start end target)
    (delete-region start end)
    (insert (ht-get emoji "unicode"))))

(define-minor-mode emoticon-to-emoji
  "Write ascii/gh emojis, and have them converted to unicode live."
  :global nil
  :init-value nil
  (if emoticon-to-emoji
      (progn
        (setq-local emojify-emoji-styles '(ascii github unicode))
        (advice-add 'emojify--propertize-text-for-emoji :around #'emojify--replace-text-with-emoji)
        (unless emojify-mode
          (emojify-turn-on-emojify-mode)))
    (setq-local emojify-emoji-styles (default-value 'emojify-emoji-styles))
    (advice-remove 'emojify--propertize-text-for-emoji #'emojify--replace-text-with-emoji)))
```

This new minor mode of ours will be nice for messages, so let's hook it in for Email and IRC.

```
(add-hook! '(mu4e-compose-mode org-msg-edit-mode circe-channel-mode) (emoticon-to-emoji 1))
```

7.5 Eros-eval

This makes the result of evals with `gr` and `gR` just slightly prettier.

```
(setq eros-eval-result-prefix " ")
```

7.6 Checkers (spell & grammar)

7.6.1 Set the default `ispell` dictionary

With `:checkers spell +hunspell`, `hunspell` needs to be installed:

```
sudo pacman -S hunspell hunspell-en_US hunspell-en_GB hunspell-fr
```

Set `ispell`'s dictionary to English and French by default.

```
(after! ispell
  (setq ispell-program-name "hunspell"      ; Use hunspell to correct mistakes
        ispell-dictionary   "en_US,fr_FR")  ; Default dictionary to use

  ;; ispell-set-spellchecker-params has to be called
  ;; before ispell-hunspell-add-multi-dic will work
  (ispell-set-spellchecker-params)
  (ispell-hunspell-add-multi-dic "en_US,fr_FR")

  ;; Define the personal dictionary path, and use it only when it exists
  (setq ispell-personal-dictionary (expand-file-name ".ispell_personal_dict" doom-private-dir))
  (unless (file-exists-p ispell-personal-dictionary)
    (write-region "" nil ispell-personal-dictionary nil 0)))
```

7.6.2 Lazy flycheck

```
(after! flyspell
  (setq flyspell-lazy-idle-seconds 3
        flyspell-lazy-window-idle-seconds 10))
```

7.6.3 Shortcuts to change dictionary

```
(defun ab-conf/spelldict (lang)
  "Switch between language dictionaries."
  (cond ((eq lang :en)
         (setq flyspell-default-dictionary "en_US"
               ispell-dictionary "en_US")
         (message "Dictionary changed to 'english'"))
        ((eq lang :fr)
         (setq flyspell-default-dictionary "fr_FR"
               ispell-dictionary "fr_FR")
         (message "Dictionary changed to 'francais'"))
        (t (message "No changes have been made.")))
  (flyspell-mode -1)
  (flyspell-mode))

(map! :leader :prefix ("l" . "custom")
      (:when (featurep! :checkers spell)
       :prefix-map ("y" . "dictionary")
       :desc "English (en_US)"    "e" #'(lambda () (interactive) (ab-conf/spelldict :en))
       :desc "Français (fr_FR)"   "f" #'(lambda () (interactive) (ab-conf/spelldict :fr))))
```

7.6.4 Shortcuts to check grammar

```
(map! :leader :prefix ("l" . "custom")
      (:when (featurep! :checkers grammar)
       :prefix-map ("l" . "langtool")
       :desc "Check"                    "l" #'langtool-check
       :desc "Correct buffer"           "b" #'langtool-correct-buffer
       :desc "Stop server"              "s" #'langtool-server-stop
       :desc "Done checking"            "d" #'langtool-check-done
       :desc "Show msg at point"        "m" #'langtool-show-message-at-point
       :desc "Next error"               "n" #'langtool-goto-next-error
```

```
      :desc "Previous error"           "p" #'langtool-goto-previous-error
      :desc "Switch default language"  "L" #'langtool-switch-default-language))
```

7.7 Projectile

Looking at documentation via `SPC h f` and `SPC h v` and looking at the source can add package src directories to projectile. This isn't desirable in my opinion.

```
;; Run `M-x projectile-project-search-path' to reload paths from this variable
;; (setq projectile-project-search-path '("~/PhD/workspace"
;;                                        "~/PhD/workspace-no"
;;                                        "~/PhD/workspace-no/ez-wheel/swd-starter-kit-repo"
;;                                        "~/Projects/foss_projects"))

(setq projectile-ignored-projects '("~/"
                                    "/tmp"
                                    "~/.emacs.d/.local/straight/repos/"))

(defun projectile-ignored-project-function (filepath)
  "Return t if FILEPATH is within any of `projectile-ignored-projects'"
  (or (mapcar (lambda (p) (s-starts-with-p p filepath)) projectile-ignored-projects)))
```

7.8 Tramp

Let's try to make tramp handle prompts better

```
(after! tramp
  (setenv "SHELL" "/bin/bash")
  (setq tramp-shell-prompt-pattern "\\(?:^\\|
\\)[^]#$%>\n]*#?[]#$%>] *\\(^^[\\[[0-9;]*[a-zA-Z] *\\)*")) ;; default +
```

7.9 YASnippet

Nested snippets are good, enable that.

```
(setq yas-triggers-in-field t)
```

7.10 Ligatures

Disable extra ligatures in some programming modes:

```
(setq +ligatures-extras-in-modes '(not c-mode c++-mode rust-mode python-mode))
```

8 Applications

8.1 e-Books `nov`

Use `nov` to read EPUB e-books.

```
(use-package! nov
  :mode ("\\.epub\\'" . nov-mode)
  :config
  (map! :map nov-mode-map
        :n "RET" #'nov-scroll-up)

  (defun doom-modeline-segment--nov-info ()
    (concat " "
```

```
            (propertize (cdr (assoc 'creator nov-metadata)) 'face 'doom-modeline-project-parent-dir)
            " "
            (cdr (assoc 'title nov-metadata))
            " "
            (propertize (format "%d/%d" (1+ nov-documents-index) (length nov-documents)) 'face 'doom-mo

  (advice-add 'nov-render-title :override #'ignore)

  (defun +nov-mode-setup ()
    (face-remap-add-relative 'variable-pitch
                             :family "Merriweather"
                             :height 1.4
                             :width 'semi-expanded)
    (face-remap-add-relative 'default :height 1.3)
    (setq-local line-spacing 0.2
                next-screen-context-lines 4
                shr-use-colors nil)
    (require 'visual-fill-column nil t)
    (setq-local visual-fill-column-center-text t
                visual-fill-column-width 80
                nov-text-width 80)
    (visual-fill-column-mode 1)
    (hl-line-mode -1)

    (add-to-list '+lookup-definition-functions #'+lookup/dictionary-definition)

    (setq-local mode-line-format
                `((:eval
                   (doom-modeline-segment--workspace-name))
                  (:eval
                   (doom-modeline-segment--window-number))
                  (:eval
                   (doom-modeline-segment--nov-info))
                  ,(propertize
                    " %P "
                    'face 'doom-modeline-buffer-minor-mode)
                  ,(propertize
                    " "
                    'face (if (doom-modeline--active) 'mode-line 'mode-line-inactive)
                    'display `((space
                                :align-to
                                (- (+ right right-fringe right-margin)
                                   ,(* (let ((width (doom-modeline--font-width)))
                                         (or (and (= width 1) 1)
                                             (/ width (frame-char-width) 1.0)))
                                       (string-width
                                        (format-mode-line (cons "" '(:eval (doom-modeline-segment--majo
                  (:eval (doom-modeline-segment--major-mode)))))

  (add-hook 'nov-mode-hook #'+nov-mode-setup))
```

8.2 Newsfeed elfeed

Set RSS news feeds

```
(setq elfeed-feeds
      '("https://this-week-in-rust.org/rss.xml"
```

```
        "https://www.omgubuntu.co.uk/feed"
        "https://lwn.net/headlines/rss"))
```

8.3 VPN Config

8.3.1 NetExtender wrapper

I store my NetExtender VPN parameters in a GPG encrypted file. The credentials file contains a line of private parameters to pass to `netExtender`, like this:

```
echo "-u <USERNAME> -d <DOMAINE> -p <PASSWORD> -s <SERVER_IP>" > gpg -c > netExtender-params.gpg
```

Then I like to have a simple script which decrypt the credentials and launch a session via the `netExtender` command.

```
#!/bin/bash

if ! command -v netExtender &> /dev/null
then
  echo "netExtender not found, installing from AUR using 'yay'"
  yay -S netextender
fi

MY_LOGIN_PARAMS_FILE="$HOME/.ssh/netExtender-params.gpg"

echo "Y\n" | netExtender --auto-reconnect $(gpg -q --for-your-eyes-only --no-tty -d ${MY_LOGIN_PARAMS_F
```

8.3.2 Launch NetExtender session from Emacs

```
(setq netextender-process-name "netextender"
      netextender-buffer-name "*netextender*"
      netextender-command '("~/.local/bin/netextender"))

(defun netextender-start ()
  "Launch a NetExtender VPN session"
  (interactive)
  (unless (get-process netextender-process-name)
    (if (make-process :name netextender-process-name
                      :buffer netextender-buffer-name
                      :command netextender-command)
        (message "Started NetExtender VPN session")
      (message "Cannot start NetExtender"))))

(defun netextender-kill ()
  "Kill the created NetExtender VPN session"
  (interactive)
  (when (get-process netextender-process-name)
    (if (kill-buffer netextender-buffer-name)
        (message "Killed NetExtender VPN session")
      (message "Cannot kill NetExtender"))))
```

8.4 Email `mu4e`

Configuring `mu4e` email accounts, note that you need to have a proper `mbsyncrc` file in the right directory.
You will need to:

- Install `mu` and `mbsync-git`
- Set up a proper configuration file for your accounts at `~/config/mu4e/mbsyncrc`

- Run `mu init --maildir=~/Maildir --my-address=user@host.bla`
- Run `mbsync -c ~/.config/mu4e/mbsyncrc -a`
- For sending mails from `mu4e`, add a `~/.authinfo` file, file contains a line in this format `machine mail.example.org port 587 login myuser password mypasswd`
- Encrypt the `~/.authinfo` file using GPG `gpg -c ~/.authinfo` and delete the original unencrypted file.

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

My Email accounts are configured in a private file in `lisp/private/+mu4e-accounts.el`, which will be loaded after the common part:

```
(after! mu4e
  (require 'org-msg)
  (require 'smtpmail)

  ;; Common parameters
  (setq smtpmail-auth-credentials "~/.authinfo.gpg"
        mu4e-maildir "~/Maildir"
        mu4e-update-interval (* 3 60) ;; Every 3 min
        ;; mu4e-get-mail-command "mbsync -a" ;; Not needed, as +mu4e-backend is 'mbsync by default
        mu4e-main-hide-personal-addresses t ;; No need to display a long list of my own addresses!
        mu4e-attachment-dir (expand-file-name "~/Maildir/attachements")
        ;; message-send-mail-function 'smtpmail-send-it ;; Set by default
        mu4e-sent-messages-behavior 'sent ;; Save sent messages
        mu4e-context-policy 'pick-first ;; Start with the first context
        mu4e-compose-context-policy 'ask) ;; Always ask which context to use when composing a new mail

  (setq mu4e-headers-fields '((:flags . 6) ;; 3 flags
                              (:account-stripe . 2)
                              (:from-or-to . 25)
                              (:folder . 10)
                              (:recipnum . 2)
                              (:subject . 80)
                              (:human-date . 8))
        +mu4e-min-header-frame-width 142
        mu4e-headers-date-format "%d/%m/%y"
        mu4e-headers-time-format " %H:%M"
        mu4e-headers-results-limit 1000
        mu4e-index-cleanup t)

  (defvar +mu4e-header--folder-colors nil)
  (appendq! mu4e-header-info-custom
            '((:folder .
               (:name "Folder" :shortname "Folder" :help "Lowest level folder" :function
                (lambda (msg)
                  (+mu4e-colorize-str
                   (replace-regexp-in-string "\\`.*/" "" (mu4e-message-field msg :maildir))
                   '+mu4e-header--folder-colors))))))

  ;; Add shortcut to view yesterday's messages
  (add-to-list 'mu4e-bookmarks
               '(:name "Yesterday's messages" :query "date:2d..1d" :key ?y) t)

  ;; Use a nicer icon in alerts
  (setq mu4e-alert-icon "/usr/share/icons/Papirus/64x64/apps/mail-client.svg")
```

```
  ;; Org-Msg stuff
  ;; org-msg-signature is set for each account separately
  (map! :map org-msg-edit-mode-map
        :after org-msg
        :n "G" #'org-msg-goto-body)

  ;; Load my accounts
  (load! "lisp/private/+mu4e-accounts.el"))
```

The `lisp/private/+mu4e-accounts.el` file includes Doom's mu4e multi-account configuration as follows:

```
(set-email-account! "Work"
                    '((mu4e-sent-folder             . "/work-dir/Sent")
                      (mu4e-drafts-folder           . "/work-dir/Drafts")
                      (mu4e-trash-folder            . "/work-dir/Trash")
                      (mu4e-refile-folder           . "/work-dir/Archive")
                      (mu4e-compose-signature       . "-- SIGNATURE")
                      (smtpmail-smtp-user           . "username@server.com")
                      (smtpmail-stream-type         . ssl)
                      (smtpmail-default-smtp-server . "smtps.server.com")
                      (smtpmail-smtp-server         . "smtps.server.com")
                      (smtpmail-smtp-service        . 465))
                     t)

(set-email-account! "Gmail"
                    '((mu4e-sent-folder             . "/gmail-dir/Sent")
                      (mu4e-drafts-folder           . "/gmail-dir/Drafts")
                      (mu4e-trash-folder            . "/gmail-dir/Trash")
                      (mu4e-refile-folder           . "/gmail-dir/Archive")
                      (mu4e-compose-signature       . "-- SIGNATURE")
                      (smtpmail-smtp-user           . "username@gmail.com")
                      ...))

; Tell Doom's mu4e module to override some commands to fix issues on Gmail accounts
(setq +mu4e-gmail-accounts '(("username@gmail.com" . "/gmail-dir")))
```

9 Programming

9.1 File Templates

For some file types, we overwrite defaults in the snippets directory, others need to have a template assigned.

```
(set-file-template! "\\.tex$" :trigger "__" :mode 'latex-mode)
(set-file-template! "\\.org$" :trigger "__" :mode 'org-mode)
(set-file-template! "/LICEN[CS]E$" :trigger '+file-templates/insert-license)
```

9.2 ROS

Add ROS specific file formats:

```
(add-to-list 'auto-mode-alist '("\\.launch$" . xml-mode))
(add-to-list 'auto-mode-alist '("\\.urdf$"   . xml-mode))
(add-to-list 'auto-mode-alist '("\\.xacro$"  . xml-mode))
(add-to-list 'auto-mode-alist '("\\.rviz$"   . conf-unix-mode))
```

9.3 LSP

9.3.1 Enable some useful UI stuff

LSP mode provides a set of configurable UI stuff, Doom Emacs disables a set of UI components to provide a less intrusive UI, however I like to enable some less intrusive, more useful UI stuff.

```
(after! lsp-ui
  (setq lsp-ui-sideline-enable t
        lsp-ui-sideline-show-code-actions t
        lsp-ui-sideline-show-diagnostics t
        lsp-ui-sideline-show-hover nil
        lsp-log-io nil
        lsp-lens-enable nil ; not working properly with ccls!
        lsp-diagnostics-provider :auto
        lsp-enable-symbol-highlighting t
        lsp-headerline-breadcrumb-enable nil
        lsp-headerline-breadcrumb-segments '(symbols)))
```

9.3.2 Fringe

Increase the left fringe width, to enable rendering breakpoints correctly.

```
(add-hook 'lsp-mode-hook (lambda () (set-fringe-mode '(12 . 12))))
```

9.3.3 Eglot

Eglot uses `project.el` to detect the project root. This is a workaround to make it work with `projectile`:

```
(after! eglot
  ;; A hack to make it works with projectile (from https://github.com/joaotavora/eglot/issues/129#issue
  (defun projectile-project-find-function (dir)
    (let* ((root (projectile-project-root dir)))
      (and root (cons 'transient root))))

  (with-eval-after-load 'project
    (add-to-list 'project-find-functions 'projectile-project-find-function))

  ;; Use clangd with some options
  (set-eglot-client! 'c++-mode '("clangd" "-j=3" "--clang-tidy")))
```

9.3.4 LSP mode with `clangd`

```
;; NOTE: Not tangled, using the default ccls
(after! lsp-clangd
  (setq lsp-clients-clangd-args
        '("-j=3"
          "--background-index"
          "--clang-tidy"
          "--completion-style=detailed"
          "--header-insertion=never"
          "--header-insertion-decorators=0"))
  (set-lsp-priority! 'clangd 2))
```

9.3.5 LSP mode with `ccls`

```
(after! ccls
  (setq ccls-initialization-options
        '(:index (:comments 2
```

```
                :trackDependency 1
                :threads 4)
        :completion (:detailedLabel t)))
  (set-lsp-priority! 'ccls 2)) ; optional as ccls is the default in Doom
```

9.3.6 Enable `lsp` over `tramp`

For Python

```
;; NOTE: WIP: Not tangled
(after! tramp
  (require 'lsp-mode)
  (require 'lsp-pyright)

  (setq lsp-enable-snippet nil
        lsp-log-io nil
        ; To bypass the "lsp--document-highlight fails if textDocument/documentHighlight is not support
        lsp-enable-symbol-highlighting nil)

  (lsp-register-client
    (make-lsp-client
     :new-connection (lsp-tramp-connection (lambda ()
                                             (cons "pyright-langserver"
                                                   lsp-pyright-langserver-command-args)))
     :major-modes '(python-mode)
     :remote? t
     :server-id 'pyright-remote)))

  (add-to-list 'tramp-remote-path 'tramp-own-remote-path)
```

For C/C++ with `ccls`

```
;; NOTE: WIP: Not tangled
(after! tramp
  (require 'lsp-mode)
  (require 'ccls)

  (setq lsp-enable-snippet nil
        lsp-log-io nil
        lsp-enable-symbol-highlighting t)

  (lsp-register-client
   (make-lsp-client
    :new-connection (lsp-tramp-connection (lambda ()
                                            (cons ccls-executable ; executable name on remote machine '
                                                  ccls-args)))
    :major-modes '(c-mode c++-mode objc-mode cuda-mode)
    :remote? t
    :server-id 'ccls-remote))
  ;; :multi-root t
  ;; :priority 3
  ;; :initialization-options (lambda () (ht-merge (lsp-configuration-section "c++")
  ;;                                              (lsp-configuration-section "ccls")))
  ;; :initialized-fn (lambda (workspace)
  ;;                   (with-lsp-workspace workspace
  ;;                     (lsp--set-configuration
  ;;                      (ht-merge (lsp-configuration-section "c++")
```

```
  ;;                                  (lsp-configuration-section "ccls")))))
  ;; :notification-handlers (lsp-ht ("$ccls/publishSkippedRanges" 'cls--publish-skipped-ranges)
  ;;                                  ("$ccls/publishSemanticHighlight" 'cls--publish-semantic-highlight)

  (add-to-list 'tramp-remote-path 'tramp-own-remote-path))
```

For C/C++ with `clangd`

```
(after! tramp
  (require 'lsp-mode)
  (setq lsp-enable-snippet nil
        lsp-log-io nil
        ; To bypass the "lsp--document-highlight fails if textDocument/documentHighlight is not support
        lsp-enable-symbol-highlighting nil)

  (lsp-register-client
    (make-lsp-client
      :new-connection (lsp-tramp-connection (lambda ()
                                              (cons "clangd-12" ; executable name on remote machine 'cc
                                                    lsp-clients-clangd-args)))
      :major-modes '(c-mode c++-mode objc-mode cuda-mode)
      :remote? t
      :server-id 'clangd-remote)))
```

9.4 DAP

```
(after! dap-mode
  (require 'dap-cpptools)

  ;; More minimal UI
  (setq dap-auto-configure-features '(locals tooltip)
        lsp-enable-dap-auto-configure t
        dap-auto-show-output nil) ;; Hide the annoying server output

  ;; Automatically trigger dap-hydra when a program hits a breakpoint.
  (add-hook 'dap-stopped-hook (lambda (arg) (call-interactively #'dap-hydra)))

  ;; Automatically delete session and close dap-hydra when DAP is terminated.
  (add-hook 'dap-terminated-hook
    (lambda (arg)
      (progn (call-interactively #'dap-delete-session)
             (dap-hydra/nil)))))
```

9.4.1 Doom store

Doom Emacs stores session information persistently using the core `store` mechanism. However, relaunching a new session doesn't overwrite the last stored session, to do so, I define a helper function to clear data stored in the `"+debugger"` location. (see `+debugger--get-last-config` function.)

```
(defun +debugger/clear-last-session ()
  "Clear the last stored session"
  (interactive)
  (doom-store-clear "+debugger"))

(map! :leader :prefix ("l" . "custom")
      (:when (featurep! :tools debugger +lsp)
       :prefix-map ("d" . "debugger")
       :desc "Clear last DAP session" "c" #'+debugger/clear-last-session))
```

9.5 TODO Realgud

Debugging C/C++ with DAP mode is Ok, however, Realgud which interfaces directly with GDB gives a richer and more flexible experience.

9.6 GDB

Stolen from `https://stackoverflow.com/a/41326527/3058915`.

```
(setq gdb-many-windows nil)

(defun set-gdb-layout(&optional c-buffer)
  (if (not c-buffer)
      (setq c-buffer (window-buffer (selected-window)))) ;; save current buffer

  ;; from http://stackoverflow.com/q/39762833/846686
  (set-window-dedicated-p (selected-window) nil) ;; unset dedicate state if needed
  (switch-to-buffer gud-comint-buffer)
  (delete-other-windows) ;; clean all

  (let* ((w-source (selected-window)) ;; left top
         (w-gdb (split-window w-source nil 'right)) ;; right bottom
         (w-locals (split-window w-gdb nil 'above)) ;; right middle bottom
         (w-stack (split-window w-locals nil 'above)) ;; right middle top
         (w-breakpoints (split-window w-stack nil 'above)) ;; right top
         (w-io (split-window w-source (floor(* 0.9 (window-body-height))) 'below))) ;; left bottom
    (set-window-buffer w-io (gdb-get-buffer-create 'gdb-inferior-io))
    (set-window-dedicated-p w-io t)
    (set-window-buffer w-breakpoints (gdb-get-buffer-create 'gdb-breakpoints-buffer))
    (set-window-dedicated-p w-breakpoints t)
    (set-window-buffer w-locals (gdb-get-buffer-create 'gdb-locals-buffer))
    (set-window-dedicated-p w-locals t)
    (set-window-buffer w-stack (gdb-get-buffer-create 'gdb-stack-buffer))
    (set-window-dedicated-p w-stack t)

    (set-window-buffer w-gdb gud-comint-buffer)

    (select-window w-source)
    (set-window-buffer w-source c-buffer)))

(defadvice gdb (around args activate)
  "Change the way to gdb works."
  (setq global-config-editing (current-window-configuration)) ;; to restore: (set-window-configuration
  (let ((c-buffer (window-buffer (selected-window)))) ;; save current buffer
    ad-do-it
    (set-gdb-layout c-buffer)))

(defadvice gdb-reset (around args activate)
  "Change the way to gdb exit."
  ad-do-it
  (set-window-configuration global-config-editing))
```

9.6.1 Highlight current line

```
(defvar gud-overlay
  (let* ((ov (make-overlay (point-min) (point-min))))
     (overlay-put ov 'face 'secondary-selection)
```

```
   ov)
    "Overlay variable for GUD highlighting.")

(defadvice gud-display-line (after my-gud-highlight act)
  "Highlight current line."
  (let* ((ov gud-overlay)
         (bf (gud-find-file true-file)))
  (with-current-buffer bf
    (move-overlay ov (line-beginning-position) (line-beginning-position 2)
      ;; (move-overlay ov (line-beginning-position) (line-end-position)
                        (current-buffer)))))

(defun gud-kill-buffer ()
  (if (derived-mode-p 'gud-mode)
    (delete-overlay gud-overlay)))

(add-hook 'kill-buffer-hook 'gud-kill-buffer)
```

9.7 Cppcheck

```
(after! flycheck
  (setq flycheck-cppcheck-checks '("information"
                                   "missingInclude"
                                   "performance"
                                   "portability"
                                   "style"
                                   "unusedFunction"
                                   "warning"))) ;; Actually, we can use "all"
```

9.8 Plain text

It's nice to see ANSI color codes displayed. However, until Emacs 28 it's not possible to do this without modifying the buffer, so let's condition this block on that.

```
(after! text-mode
  (add-hook! 'text-mode-hook
             ;; Apply ANSI color codes
             (with-silent-modifications
               (ansi-color-apply-on-region (point-min) (point-max) t))))
```

9.9 Org

9.9.1 Intro

Because this section is fairly expensive to initialize, we'll wrap it in a `(after! ...)` block.

```
(after! org
  <<org-conf>>
)
```

9.9.2 Behavior

Tweaking defaults

```
(setq org-directory "~/Dropbox/Org/"             ; let's put files here
      org-use-property-inheritance t             ; it's convenient to have properties inherited
      org-log-done 'time                         ; having the time an item is done sounds convenient
      org-list-allow-alphabetical t              ; have a. A. a) A) list bullets
```

```
;;    org-export-in-background t                ; run export processes in external emacs process
;;    org-export-async-debug t
      org-catch-invisible-edits 'smart          ; try not to accidently do weird stuff in invisible r
      org-export-with-sub-superscripts '{})     ; don't treat lone _ / ^ as sub/superscripts, require
```

I also like the `:comments` header-argument, so let's make that a default.

```
(setq org-babel-default-header-args
      '((:session  . "none")
        (:results  . "replace")
        (:exports  . "code")
        (:cache    . "no")
        (:noweb    . "no")
        (:hlines   . "no")
        (:tangle   . "no")
        (:comments . "link")))
```

By default, `visual-line-mode` is turned `on`, and `auto-fill-mode` `off` by a hook. However, this messes with tables in Org-mode, and other plaintext files (e.g. markdown, LaTeX) so I'll turn it off for this, and manually enable it for more specific modes as desired.

```
(remove-hook 'text-mode-hook #'visual-line-mode)
(add-hook 'text-mode-hook #'auto-fill-mode)
```

There also seem to be a few keybindings which use `hjkl`, but miss arrow key equivalents.

```
(map! :map evil-org-mode-map
      :after evil-org
      :n "g <up>" #'org-backward-heading-same-level
      :n "g <down>" #'org-forward-heading-same-level
      :n "g <left>" #'org-up-element
      :n "g <right>" #'org-down-element)
```

Extra functionality

List bullet sequence I think it makes sense to have list bullets change with depth

```
(setq org-list-demote-modify-bullet '(("+" . "-") ("-" . "+") ("*" . "+") ("1." . "a.")))
```

Citation (`org-ref`) Occasionally I want to cite something, and `org-ref` is *the* package for that.

Unfortunately, it ignores the `file = {...} .bib` keys though. Let's fix that. I separate files on `;`, which may just be a Zotero/BetterBibLaTeX thing, but it's a good idea in my case at least.

```
(use-package! org-ref
  :after org
  :config
  (defadvice! org-ref-open-bibtex-pdf-a ()
    :override #'org-ref-open-bibtex-pdf
    (save-excursion
      (bibtex-beginning-of-entry)
      (let* ((bibtex-expand-strings t)
             (entry (bibtex-parse-entry t))
             (key (reftex-get-bib-field "=key=" entry))
             (pdf (or
                   (car (-filter (lambda (f) (string-match-p "\\.pdf$" f))
                                 (split-string (reftex-get-bib-field "file" entry) ";")))
                   (funcall org-ref-get-pdf-filename-function key))))
```

```
        (if (file-exists-p pdf)
            (org-open-file pdf)
          (ding)))))
  (defadvice! org-ref-open-pdf-at-point-a ()
    "Open the pdf for bibtex key under point if it exists."
    :override #'org-ref-open-pdf-at-point
    (interactive)
    (let* ((results (org-ref-get-bibtex-key-and-file))
           (key (car results))
           (pdf-file (funcall org-ref-get-pdf-filename-function key)))
      (with-current-buffer (find-file-noselect (cdr results))
        (save-excursion
          (bibtex-search-entry (car results))
          (org-ref-open-bibtex-pdf))))))
```

Spellcheck I turn off spell checking by default to make Org files open quickly.

```
(add-hook 'org-mode-hook 'turn-off-flyspell)
;;(add-hook 'org-mode-hook 'turn-on-flyspell)
;;(add-hook 'org-mode-hook 'spell-fu-mode-disable)
```

TODOs

```
(setq org-todo-keywords
      '((sequence "TODO(t)" "PROJ(p)" "LOOP(r)" "STRT(s)" "WAIT(w)" "HOLD(h)" "IDEA(i)" "|" "DONE(d)" "
        (sequence "[ ](T)" "[-](S)" "[?](W)" "|" "[X](D)")
        (sequence "|" "OKAY(o)" "YES(y)" "NO(n)")))

;; (defun log-todo-next-creation-date (&rest ignore)
;;  "Log NEXT creation time in the property drawer under the key 'ACTIVATED'"
;;  (when (and (string= (org-get-todo-state) "NEXT")
;;             (not (org-entry-get nil "ACTIVATED")))
;;    (org-entry-put nil "ACTIVATED" (format-time-string "[%Y-%m-%d]"))))
;; (add-hook 'org-after-todo-state-change-hook #'log-todo-next-creation-date)
```

Super agenda Set files for `org-agenda`

```
(setq org-agenda-files '("~/Dropbox/Org/inbox.org"
                         "~/Dropbox/Org/agenda.org"
                         "~/Dropbox/Org/gcal-agenda.org"
                         "~/Dropbox/Org/notes.org"
                         "~/Dropbox/Org/projects.org"))
```

Configure `org-super-agenda`

```
(use-package! org-super-agenda
  :after org-agenda
  :config (org-super-agenda-mode))

(setq org-agenda-skip-scheduled-if-done t
      org-agenda-skip-deadline-if-done t
      org-agenda-include-deadlines t
      org-agenda-block-separator nil
      org-agenda-tags-column 100 ;; from testing this seems to be a good value
      org-agenda-compact-blocks t)

(setq org-agenda-custom-commands
```

```
'(("o" "Overview"
   ((agenda "" ((org-agenda-span 'day)
                (org-super-agenda-groups
                 '((:name "Today"
                    :time-grid t
                    :date today
                    :todo "TODAY"
                    :scheduled today
                    :order 1)))))
    (alltodo "" ((org-agenda-overriding-header "")
                 (org-super-agenda-groups
                  '((:name "Next to do"
                     :todo "NEXT"
                     :order 1)
                    (:name "Important"
                     :tag "Important"
                     :priority "A"
                     :order 6)
                    (:name "Due Today"
                     :deadline today
                     :order 2)
                    (:name "Due Soon"
                     :deadline future
                     :order 8)
                    (:name "Overdue"
                     :deadline past
                     :face error
                     :order 7)
                    (:name "Assignments"
                     :tag "Assignment"
                     :order 10)
                    (:name "Issues"
                     :tag "Issue"
                     :order 12)
                    (:name "Emacs"
                     :tag "Emacs"
                     :order 13)
                    (:name "Projects"
                     :tag "Project"
                     :order 14)
                    (:name "Research"
                     :tag "Research"
                     :order 15)
                    (:name "To read"
                     :tag "Read"
                     :order 30)
                    (:name "Waiting"
                     :todo "WAIT"
                     :order 20)
                    (:name "University"
                     :tag "Univ"
                     :order 32)
                    (:name "Trivial"
                     :priority<= "E"
                     :tag ("Trivial" "Unimportant")
                     :todo ("SOMEDAY")
```

```
                                :order 90)
                               (:discard (:tag ("Chore" "Routine" "Daily")))))))))))
```

Google calendar (`org-gcal`) I store my `org-gcal` config privately, it contains something like this:

```
(after! org-gcal
  (setq org-gcal-client-id "<SOME_ID>.apps.googleusercontent.com"
        org-gcal-client-secret "<SOME_SECRET>"
        org-gcal-fetch-file-alist '(("<username>@gmail.com" . "~/Dropbox/Org/gcal-agenda.org"))))

(load! "lisp/private/+org-gcal.el")
```

Capture Set capture files

```
(setq +org-capture-emails-file (concat org-directory "inbox.org")
      +org-capture-todo-file (concat org-directory "inbox.org")
      +org-capture-projects-file (concat org-directory "projects.org"))
```

Lets set up some org-capture templates, and make them visually nice to access.

```
(use-package! doct
  :commands (doct))

(after! org-capture
  <<prettify-capture>>

  (defun +doct-icon-declaration-to-icon (declaration)
    "Convert :icon declaration to icon"
    (let ((name (pop declaration))
          (set  (intern (concat "all-the-icons-" (plist-get declaration :set))))
          (face (intern (concat "all-the-icons-" (plist-get declaration :color))))
          (v-adjust (or (plist-get declaration :v-adjust) 0.01)))
      (apply set `(,name :face ,face :v-adjust ,v-adjust))))

  (defun +doct-iconify-capture-templates (groups)
    "Add declaration's :icon to each template group in GROUPS."
    (let ((templates (doct-flatten-lists-in groups)))
      (setq doct-templates (mapcar (lambda (template)
                                     (when-let* ((props (nthcdr (if (= (length template) 4) 2 5) templa
                                                 (spec (plist-get (plist-get props :doct) :icon)))
                                       (setf (nth 1 template) (concat (+doct-icon-declaration-to-icon s
                                                                      "\t"
                                                                      (nth 1 template))))
                                     template)
                                   templates))))

  (setq doct-after-conversion-functions '(+doct-iconify-capture-templates))

  (defun set-org-capture-templates ()
    (setq org-capture-templates
          (doct `(("Personal todo" :keys "t"
                   :icon ("checklist" :set "octicon" :color "green")
                   :file +org-capture-todo-file
                   :prepend t
                   :headline "Inbox"
                   :type entry
                   :template ("* TODO %?"
```

```
            "%i %a")
 )
("Personal note" :keys "n"
 :icon ("sticky-note-o" :set "faicon" :color "green")
 :file +org-capture-todo-file
 :prepend t
 :headline "Inbox"
 :type entry
 :template ("* %?"
            "%i %a"))
("Email" :keys "e"
 :icon ("envelope" :set "faicon" :color "blue")
 :file +org-capture-todo-file
 :prepend t
 :headline "Inbox"
 :type entry
 :template ("* TODO %^{type|reply to|contact} %\\3 %? :email:"
            "Send an email %^{urgancy|soon|ASAP|anon|at some point|eventually} to %^{
            "about %^{topic}"
            "%U %i %a"))
("Interesting" :keys "i"
 :icon ("eye" :set "faicon" :color "lcyan")
 :file +org-capture-todo-file
 :prepend t
 :headline "Interesting"
 :type entry
 :template ("* [ ] %{desc}%? :%{i-type}:"
            "%i %a")
 :children (("Webpage" :keys "w"
             :icon ("globe" :set "faicon" :color "green")
             :desc "%(org-cliplink-capture) "
             :i-type "read:web"
             )
            ("Article" :keys "a"
             :icon ("file-text" :set "octicon" :color "yellow")
             :desc ""
             :i-type "read:reaserch"
             )
            ("Information" :keys "i"
             :icon ("info-circle" :set "faicon" :color "blue")
             :desc ""
             :i-type "read:info"
             )
            ("Idea" :keys "I"
             :icon ("bubble_chart" :set "material" :color "silver")
             :desc ""
             :i-type "idea"
             )))
("Tasks" :keys "k"
 :icon ("inbox" :set "octicon" :color "yellow")
 :file +org-capture-todo-file
 :prepend t
 :headline "Tasks"
 :type entry
 :template ("* TODO %? %^G%{extra}"
            "%i %a")
```

```
 :children (("General Task" :keys "k"
             :icon ("inbox" :set "octicon" :color "yellow")
             :extra ""
             )
            ("Task with deadline" :keys "d"
             :icon ("timer" :set "material" :color "orange" :v-adjust -0.1)
             :extra "\nDEADLINE: %^{Deadline:}t"
             )
            ("Scheduled Task" :keys "s"
             :icon ("calendar" :set "octicon" :color "orange")
             :extra "\nSCHEDULED: %^{Start time:}t"
             )
            ))
("Project" :keys "p"
 :icon ("repo" :set "octicon" :color "silver")
 :prepend t
 :type entry
 :headline "Inbox"
 :template ("* %{time-or-todo} %?"
            "%i"
            "%a")
 :file ""
 :custom (:time-or-todo "")
 :children (("Project-local todo" :keys "t"
             :icon ("checklist" :set "octicon" :color "green")
             :time-or-todo "TODO"
             :file +org-capture-project-todo-file)
            ("Project-local note" :keys "n"
             :icon ("sticky-note" :set "faicon" :color "yellow")
             :time-or-todo "%U"
             :file +org-capture-project-notes-file)
            ("Project-local changelog" :keys "c"
             :icon ("list" :set "faicon" :color "blue")
             :time-or-todo "%U"
             :heading "Unreleased"
             :file +org-capture-project-changelog-file))
 )
("\tCentralised project templates"
 :keys "o"
 :type entry
 :prepend t
 :template ("* %{time-or-todo} %?"
            "%i"
            "%a")
 :children (("Project todo"
             :keys "t"
             :prepend nil
             :time-or-todo "TODO"
             :heading "Tasks"
             :file +org-capture-central-project-todo-file)
            ("Project note"
             :keys "n"
             :time-or-todo "%U"
             :heading "Notes"
             :file +org-capture-central-project-notes-file)
            ("Project changelog"
```

```
                                :keys "c"
                                :time-or-todo "%U"
                                :heading "Unreleased"
                                :file +org-capture-central-project-changelog-file))
                    )))))

  (set-org-capture-templates)
  (unless (display-graphic-p)
    (add-hook 'server-after-make-frame-hook
              (defun org-capture-reinitialise-hook ()
                (when (display-graphic-p)
                  (set-org-capture-templates)
                  (remove-hook 'server-after-make-frame-hook
                               #'org-capture-reinitialise-hook))))))
```

It would also be nice to improve how the capture dialogue looks

```
(defun org-capture-select-template-prettier (&optional keys)
  "Select a capture template, in a prettier way than default
Lisp programs can force the template by setting KEYS to a string."
  (let ((org-capture-templates
         (or (org-contextualize-keys
              (org-capture-upgrade-templates org-capture-templates)
              org-capture-templates-contexts)
             '(("t" "Task" entry (file+headline "" "Tasks")
                "* TODO %?\n  %u\n  %a")))))
    (if keys
        (or (assoc keys org-capture-templates)
            (error "No capture template referred to by \"%s\" keys" keys))
      (org-mks org-capture-templates
               "Select a capture template\n"
               "Template key: "
               `(("q" ,(concat (all-the-icons-octicon "stop" :face 'all-the-icons-red :v-adjust 0.01) "
(advice-add 'org-capture-select-template :override #'org-capture-select-template-prettier)

(defun org-mks-pretty (table title &optional prompt specials)
  "Select a member of an alist with multiple keys. Prettified.

TABLE is the alist which should contain entries where the car is a string.
There should be two types of entries.

1. prefix descriptions like (\"a\" \"Description\")
   This indicates that `a' is a prefix key for multi-letter selection, and
   that there are entries following with keys like \"ab\", \"ax\"...

2. Select-able members must have more than two elements, with the first
   being the string of keys that lead to selecting it, and the second a
   short description string of the item.

The command will then make a temporary buffer listing all entries
that can be selected with a single key, and all the single key
prefixes.  When you press the key for a single-letter entry, it is selected.
When you press a prefix key, the commands (and maybe further prefixes)
under this key will be shown and offered for selection.

TITLE will be placed over the selection in the temporary buffer,
PROMPT will be used when prompting for a key.  SPECIALS is an
```

```
alist with (\"key\" \"description\") entries.  When one of these
is selected, only the bare key is returned."
  (save-window-excursion
    (let ((inhibit-quit t)
          (buffer (org-switch-to-buffer-other-window "*Org Select*"))
          (prompt (or prompt "Select: "))
          case-fold-search
          current)
      (unwind-protect
          (catch 'exit
            (while t
              (setq-local evil-normal-state-cursor (list nil))
              (erase-buffer)
              (insert title "\n\n")
              (let ((des-keys nil)
                    (allowed-keys '("\C-g"))
                    (tab-alternatives '("\s" "\t" "\r"))
                    (cursor-type nil))
                ;; Populate allowed keys and descriptions keys
                ;; available with CURRENT selector.
                (let ((re (format "\\`%s\\(.\\)\\'"
                                  (if current (regexp-quote current) "")))
                      (prefix (if current (concat current " ") "")))
                  (dolist (entry table)
                    (pcase entry
                      ;; Description.
                      (`(,(and key (pred (string-match re))) ,desc)
                       (let ((k (match-string 1 key)))
                         (push k des-keys)
                         ;; Keys ending in tab, space or RET are equivalent.
                         (if (member k tab-alternatives)
                             (push "\t" allowed-keys)
                           (push k allowed-keys))
                         (insert (propertize prefix 'face 'font-lock-comment-face) (propertize k 'face
                      ;; Usable entry.
                      (`(,(and key (pred (string-match re))) ,desc . ,_)
                       (let ((k (match-string 1 key)))
                         (insert (propertize prefix 'face 'font-lock-comment-face) (propertize k 'face
                         (push k allowed-keys)))
                      (_ nil))))
                ;; Insert special entries, if any.
                (when specials
                  (insert "\n")
                  (pcase-dolist (`(,key ,description) specials)
                    (insert (format "%s   %s\n" (propertize key 'face '(bold all-the-icons-red)) descri
                    (push key allowed-keys)))
                ;; Display UI and let user select an entry or
                ;; a sublevel prefix.
                (goto-char (point-min))
                (unless (pos-visible-in-window-p (point-max))
                  (org-fit-window-to-buffer))
                (let ((pressed (org--mks-read-key allowed-keys
                                                  prompt
                                                  (not (pos-visible-in-window-p (1- (point-max)))))))
                  (setq current (concat current pressed))
                  (cond
```

```
                    ((equal pressed "\C-g") (user-error "Abort"))
                    ;; Selection is a prefix: open a new menu.
                    ((member pressed des-keys))
                    ;; Selection matches an association: return it.
                    ((let ((entry (assoc current table)))
                       (and entry (throw 'exit entry))))
                    ;; Selection matches a special entry: return the
                    ;; selection prefix.
                    ((assoc current specials) (throw 'exit current))
                    (t (error "No entry available")))))))
        (when buffer (kill-buffer buffer))))))
(advice-add 'org-mks :override #'org-mks-pretty)
```

The org-capture bin is rather nice, but I'd be nicer with a smaller frame, and no modeline.

```
(setf (alist-get 'height +org-capture-frame-parameters) 15)
;; (alist-get 'name +org-capture-frame-parameters) " Capture") ;; ATM hardcoded in other places, so cha
(setq +org-capture-fn
      (lambda ()
        (interactive)
        (set-window-parameter nil 'mode-line-format 'none)
        (org-capture)))
```

Roam

Basic settings

```
(setq org-roam-directory "~/Dropbox/Org/slip-box")
(setq org-roam-db-location "~/Dropbox/Org/slip-box/org-roam.db")
(setq org-roam-index-file "~/Dropbox/Org/slip-box/index.org")
(setq org-roam-directory "~/Dropbox/Org/slip-box/")
```

That said, if the directory doesn't exist we likely don't want to be using roam. Since we don't want to trigger errors (which will happen as soon as roam tries to initialize), let's not load roam.

```
(package! org-roam :disable t)
```

Modeline file name All those numbers! It's messy. Let's adjust this similarly that I have in the window title

```
(defadvice! doom-modeline--buffer-file-name-roam-aware-a (orig-fun)
  :around #'doom-modeline-buffer-file-name ; takes no args
  (if (s-contains-p org-roam-directory (or buffer-file-name ""))
      (replace-regexp-in-string
       "\\(?:^\\|.*/\\)\\([0-9]\\{4\\}\\)\\([0-9]\\{2\\}\\)\\([0-9]\\{2\\}\\)[0-9]*-"
       "(\\1-\\2-\\3) "
       (subst-char-in-string ?_ ?  buffer-file-name))
    (funcall orig-fun)))
```

Org Roam Capture template

```
(after! org-roam
   (setq org-roam-capture-ref-templates
         '(("r" "ref" plain "%?"
            :if-new (file+head "web/%<%Y%m%d%H%M%S>-${slug}.org" "#+title: ${title}\n#+created: %U\n\n$
            :unnarrowed t))))
```

Snippet Helpers I often want to set `src-block` headers, and it's a pain to

- type them out
- remember what the accepted values are
- oh, and specifying the same language again and again

We can solve this in three steps

- having one-letter snippets, conditioned on `(point)` being within a src header
- creating a nice prompt showing accepted values and the current default
- pre-filling the `src-block` language with the last language used

For header args, the keys I'll use are

- `r` for `:results`
- `e` for `:exports`
- `v` for `:eval`
- `s` for `:session`
- `d` for `:dir`

```
(defun +yas/org-src-header-p ()
  "Determine whether `point' is within a src-block header or header-args."
  (pcase (org-element-type (org-element-context))
    ('src-block (< (point) ; before code part of the src-block
                   (save-excursion (goto-char (org-element-property :begin (org-element-context)))
                                   (forward-line 1)
                                   (point))))
    ('inline-src-block (< (point) ; before code part of the inline-src-block
                          (save-excursion (goto-char (org-element-property :begin (org-element-context)))
                                          (search-forward "]{")
                                          (point))))
    ('keyword (string-match-p "^header-args" (org-element-property :value (org-element-context))))))
```

Now let's write a function we can reference in yasnippets to produce a nice interactive way to specify header args.

```
(defun +yas/org-prompt-header-arg (arg question values)
  "Prompt the user to set ARG header property to one of VALUES with QUESTION.
The default value is identified and indicated. If either default is selected,
or no selection is made: nil is returned."
  (let* ((src-block-p (not (looking-back "^#\\+property:[ \t]+header-args:.*" (line-beginning-position))))
         (default
           (or
            (cdr (assoc arg
                        (if src-block-p
                            (nth 2 (org-babel-get-src-block-info t))
                          (org-babel-merge-params
                           org-babel-default-header-args
                           (let ((lang-headers
                                  (intern (concat "org-babel-default-header-args:"
                                                  (+yas/org-src-lang)))))
                             (when (boundp lang-headers) (eval lang-headers t)))))))
            "")))
```

```
        default-value)
    (setq values (mapcar
                  (lambda (value)
                    (if (string-match-p (regexp-quote value) default)
                        (setq default-value
                              (concat value " "
                                      (propertize "(default)" 'face 'font-lock-doc-face)))
                      value))
                  values))
    (let ((selection (consult--read question values :default default-value)))
      (unless (or (string-match-p "(default)$" selection)
                  (string= "" selection))
        selection))))
```

Finally, we fetch the language information for new source blocks.

Since we're getting this info, we might as well go a step further and also provide the ability to determine the most popular language in the buffer that doesn't have any `header-args` set for it (with `#+properties`).

```
(defun +yas/org-src-lang ()
  "Try to find the current language of the src/header at `point'.
Return nil otherwise."
  (let ((context (org-element-context)))
    (pcase (org-element-type context)
      ('src-block (org-element-property :language context))
      ('inline-src-block (org-element-property :language context))
      ('keyword (when (string-match "^header-args:\\([^ ]+\\)" (org-element-property :value context))
                  (match-string 1 (org-element-property :value context)))))))

(defun +yas/org-last-src-lang ()
  "Return the language of the last src-block, if it exists."
  (save-excursion
    (beginning-of-line)
    (when (re-search-backward "^[ \t]*#\\+begin_src" nil t)
      (org-element-property :language (org-element-context)))))

(defun +yas/org-most-common-no-property-lang ()
  "Find the lang with the most source blocks that has no global header-args, else nil."
  (let (src-langs header-langs)
    (save-excursion
      (goto-char (point-min))
      (while (re-search-forward "^[ \t]*#\\+begin_src" nil t)
        (push (+yas/org-src-lang) src-langs))
      (goto-char (point-min))
      (while (re-search-forward "^[ \t]*#\\+property: +header-args" nil t)
        (push (+yas/org-src-lang) header-langs)))

    (setq src-langs
          (mapcar #'car
                  ;; sort alist by frequency (desc.)
                  (sort
                   ;; generate alist with form (value . frequency)
                   (cl-loop for (n . m) in (seq-group-by #'identity src-langs)
                            collect (cons n (length m)))
                   (lambda (a b) (> (cdr a) (cdr b))))))

    (car (cl-set-difference src-langs header-langs :test #'string=))))
```

Translate capital keywords (old) to lower case (new) Everyone used to use `#+CAPITAL` keywords. Then people realised that `#+lowercase` is actually both marginally easier and visually nicer, so now the capital version is just used in the manual.

> Org is standardized on lower case. Uppercase is used in the manual as a poor man's bold, and supported for historical reasons. — Nicolas Goaziou on the Org ML

To avoid sometimes having to choose between the hassle out of updating old documents and using mixed syntax, I'll whip up a basic transcode-y function. It likely misses some edge cases, but should mostly work.

```
(defun org-syntax-convert-keyword-case-to-lower ()
  "Convert all #+KEYWORDS to #+keywords."
  (interactive)
  (save-excursion
    (goto-char (point-min))
    (let ((count 0)
          (case-fold-search nil))
      (while (re-search-forward "^[ \t]*#\\+[A-Z_]+" nil t)
        (unless (s-matches-p "RESULTS" (match-string 0))
          (replace-match (downcase (match-string 0)) t)
          (setq count (1+ count))))
      (message "Replaced %d occurances" count))))
```

Fix problematic hooks When one of the `org-mode-hook` functions errors, it halts the hook execution. This is problematic, and there are two hooks in particular which cause issues. Let's make their failure less eventful.

```
(defadvice! shut-up-org-problematic-hooks (orig-fn &rest args)
  :around #'org-fancy-priorities-mode
  :around #'org-superstar-mode
  :around #'dap-mode-hook
  (ignore-errors (apply orig-fn args)))
```

9.9.3 Custom links

Subfig This defines a new link type `subfig` to enable exporting sub-figures to LaTeX, taken form Export subfigures to LaTeX (and HTML).

```
(org-link-set-parameters
 "subfig"
 :follow (lambda (file) (find-file file))
 :face '(:foreground "chocolate" :weight bold :underline t)
 :display 'full
 :export (lambda (file desc backend)
           (when (eq backend 'latex)
             (if (string-match ">(\\(.+\\))" desc)
                 (concat "\\begin{subfigure}[b]"
                         "\\caption{"
                         (replace-regexp-in-string "\s+>(.+)" "" desc)
                         "}"
                         "\\includegraphics"
                         "["
                         (match-string 1 desc)
                         "]"
                         "{"
                         file
                         "}"
                         "\\end{subfigure}")
               (format "\\begin{subfigure}\\includegraphics{%s}\\end{subfigure}" desc file)))))
```

Example of usage:

```
#+caption: Lorem impsum dolor
#+attr_latex: :options \centering
#+begin_figure
[[subfig:img1.jpg][Caption of img1 >(width=.3\textwidth)]]

[[subfig:img2.jpg][Caption of img2 >(width=.3\textwidth)]]

[[subfig:img3.jpg][Caption of img3 >(width=.6\textwidth)]]
#+end_figure
```

9.9.4 Visuals

Here I try to do two things: improve the styling of the various documents, via font changes etc., and also propagate colours from the current theme.

Font Display

Org Pretty Mode Activate `+org-pretty-mode`.

Headings Let's make the title and the headings a bit bigger:

```
(custom-set-faces!
  '(org-document-title :height 1.2))

(custom-set-faces!
  '(outline-1 :weight extra-bold :height 1.25)
  '(outline-2 :weight bold :height 1.15)
  '(outline-3 :weight bold :height 1.12)
  '(outline-4 :weight semi-bold :height 1.09)
  '(outline-5 :weight semi-bold :height 1.06)
  '(outline-6 :weight semi-bold :height 1.03)
  '(outline-8 :weight semi-bold)
  '(outline-9 :weight semi-bold))
```

Deadlines It seems reasonable to have deadlines in the error face when they're passed.

```
(setq org-agenda-deadline-faces
      '((1.001 . error)
        (1.0 . org-warning)
        (0.5 . org-upcoming-deadline)
        (0.0 . org-upcoming-distant-deadline)))
```

Font styling We can then have quote blocks stand out a bit more by making them *italic*.

```
(setq org-fontify-quote-and-verse-blocks t)
```

While `org-hide-emphasis-markers` is very nice, it can sometimes make edits which occur at the border a bit more fiddley. We can improve this situation without sacrificing visual amenities with the `org-appear` package.

```
(use-package! org-appear
  :hook (org-mode . org-appear-mode)
  :config
  (setq org-appear-autoemphasis t
        org-appear-autosubmarkers t
```

```
      org-appear-autolinks nil)
  ;; for proper first-time setup, `org-appear--set-elements'
  ;; needs to be run after other hooks have acted.
  (run-at-time nil nil #'org-appear--set-elements))
```

Fontifying inline src blocks Org does lovely things with `#+begin_src` blocks, like using font-lock for language's major-mode behind the scenes and pulling out the lovely colorful results. By contrast, inline `src_` blocks are somewhat neglected.

I am not the first person to feel this way, thankfully others have taken to stackexchange to voice their desire for inline src fontification. I was going to steal their work, but unfortunately they didn't perform *true* source code fontification, but simply applied the `org-code` face to the content.

We can do better than that, and we shall! Using `org-src-font-lock-fontify-block` we can apply language-appropriate syntax highlighting. Then, continuing on to `{{{results(...)}}}` , it can have the `org-block` face applied to match, and then the value-surrounding constructs hidden by mimicking the behavior of `prettify-symbols-mode`.

This currently only highlights a single inline src block per line. I have no idea why it stops, but I'd rather it didn't. If you have any idea what's going on or how to fix this *please* get in touch.

```
(defvar org-prettify-inline-results t
  "Whether to use (ab)use prettify-symbols-mode on {{{results(...)}}}.
Either t or a cons cell of strings which are used as substitutions
for the start and end of inline results, respectively.")

(defvar org-fontify-inline-src-blocks-max-length 200
  "Maximum content length of an inline src block that will be fontified.")

(defun org-fontify-inline-src-blocks (limit)
  "Try to apply `org-fontify-inline-src-blocks-1'."
  (condition-case nil
      (org-fontify-inline-src-blocks-1 limit)
    (error (message "Org mode fontification error in %S at %d"
                    (current-buffer)
                    (line-number-at-pos)))))

(defun org-fontify-inline-src-blocks-1 (limit)
  "Fontify inline src_LANG blocks, from `point' up to LIMIT."
  (let ((case-fold-search t)
        (initial-point (point)))
    (while (re-search-forward "\\_<src_\\([^ \t\n[{]+\\)[{[]?" limit t) ; stolen from `org-element-inli
      (let ((beg (match-beginning 0))
            pt
            (lang-beg (match-beginning 1))
            (lang-end (match-end 1)))
        (remove-text-properties beg lang-end '(face nil))
        (font-lock-append-text-property lang-beg lang-end 'face 'org-meta-line)
        (font-lock-append-text-property beg lang-beg 'face 'shadow)
        (font-lock-append-text-property beg lang-end 'face 'org-block)
        (setq pt (goto-char lang-end))
        ;; `org-element--parse-paired-brackets' doesn't take a limit, so to
        ;; prevent it searching the entire rest of the buffer we temporarily
        ;; narrow the active region.
        (save-restriction
          (narrow-to-region beg (min (point-max) limit (+ lang-end org-fontify-inline-src-blocks-max-le
          (when (ignore-errors (org-element--parse-paired-brackets ?\[))
            (remove-text-properties pt (point) '(face nil))
            (font-lock-append-text-property pt (point) 'face 'org-block)
```

```
            (setq pt (point)))
          (when (ignore-errors (org-element--parse-paired-brackets ?\{))
            (remove-text-properties pt (point) '(face nil))
            (font-lock-append-text-property pt (1+ pt) 'face '(org-block shadow))
            (unless (= (1+ pt) (1- (point)))
              (if org-src-fontify-natively
                  (org-src-font-lock-fontify-block (buffer-substring-no-properties lang-beg lang-end) (
                (font-lock-append-text-property (1+ pt) (1- (point)) 'face 'org-block)))
            (font-lock-append-text-property (1- (point)) (point) 'face '(org-block shadow))
            (setq pt (point))))
        (when (and org-prettify-inline-results (re-search-forward "\\= {{{results(" limit t))
          (font-lock-append-text-property pt (1+ pt) 'face 'org-block)
          (goto-char pt))))
    (when org-prettify-inline-results
      (goto-char initial-point)
      (org-fontify-inline-src-results limit))))

(defun org-fontify-inline-src-results (limit)
  (while (re-search-forward "{{{results(\\(.+?\\))}}}" limit t)
    (remove-list-of-text-properties (match-beginning 0) (point)
                                    '(composition
                                      prettify-symbols-start
                                      prettify-symbols-end))
    (font-lock-append-text-property (match-beginning 0) (match-end 0) 'face 'org-block)
    (let ((start (match-beginning 0)) (end (match-beginning 1)))
      (with-silent-modifications
        (compose-region start end (if (eq org-prettify-inline-results t) "〈" (car org-prettify-inline-r
        (add-text-properties start end `(prettify-symbols-start ,start prettify-symbols-end ,end))))
    (let ((start (match-end 1)) (end (point)))
      (with-silent-modifications
        (compose-region start end (if (eq org-prettify-inline-results t) "〉" (cdr org-prettify-inline-r
        (add-text-properties start end `(prettify-symbols-start ,start prettify-symbols-end ,end))))))

(defun org-fontify-inline-src-blocks-enable ()
  "Add inline src fontification to font-lock in Org.
Must be run as part of `org-font-lock-set-keywords-hook'."
  (setq org-font-lock-extra-keywords
        (append org-font-lock-extra-keywords '((org-fontify-inline-src-blocks)))))

(add-hook 'org-font-lock-set-keywords-hook #'org-fontify-inline-src-blocks-enable)
```

Symbols It's also nice to change the character used for collapsed items (by default ...), I think is better for indicating 'collapsed section'. and add an extra `org-bullet` to the default list of four. I've also added some fun alternatives, just commented out.

```
(after! org-superstar
  (setq org-superstar-headline-bullets-list '("" "" "" "" "" "" "" "")
        org-superstar-prettify-item-bullets t))

(setq org-ellipsis "  "
      org-hide-leading-stars t
      org-priority-highest ?A
      org-priority-lowest ?E
      org-priority-faces
      '((?A . 'all-the-icons-red)
        (?B . 'all-the-icons-orange)
```

```
        (?C . 'all-the-icons-yellow)
        (?D . 'all-the-icons-green)
        (?E . 'all-the-icons-blue)))
```

It's also nice to make use of the Unicode characters for check boxes, and other commands.

```
(appendq! +ligatures-extra-symbols
          '(:checkbox      ""
            :pending       ""
            :checkedbox    ""
            :list_property ""
            :em_dash       "-"
            :ellipses      "..."
            :arrow_right   "→"
            :arrow_left    "←"
            :title         ""
            :subtitle      ""
            :author        ""
            :date          ""
            :property      ""
            :options       ""
            :startup       ""
            :macro         ""
            :html_head     ""
            :html          ""
            :latex_class   ""
            :latex_header  ""
            :beamer_header ""
            :latex         ""
            :attr_latex    ""
            :attr_html     ""
            :attr_org      ""
            :begin_quote   ""
            :end_quote     ""
            :caption       ""
            :header        ">"
            :results       ""
            :begin_export  ""
            :end_export    ""
            :filetags      "#"
            :created       ""
            :include       ""
            :setupfile     ""
            :properties    ""
            :end           ""
            :priority_a   ,(propertize "" 'face 'all-the-icons-red)
            :priority_b   ,(propertize "" 'face 'all-the-icons-orange)
            :priority_c   ,(propertize "" 'face 'all-the-icons-yellow)
            :priority_d   ,(propertize "" 'face 'all-the-icons-green)
            :priority_e   ,(propertize "" 'face 'all-the-icons-blue)))
(set-ligatures! 'org-mode
  :merge t
  :checkbox      "[ ]"
  :pending       "[-]"
  :checkedbox    "[X]"
  :list_property "::"
  :em_dash       "---"
```

```
  :ellipsis      "..."
  :arrow_right   "->"
  :arrow_left    "<-"
  :title         "#+title:"
  :subtitle      "#+subtitle:"
  :author        "#+author:"
  :date          "#+date:"
  :property      "#+property:"
  :options       "#+options:"
  :startup       "#+startup:"
  :macro         "#+macro:"
  :html_head     "#+html_head:"
  :html          "#+html:"
  :latex_class   "#+latex_class:"
  :latex_header  "#+latex_header:"
  :beamer_header "#+beamer_header:"
  :latex         "#+latex:"
  :attr_latex    "#+attr_latex:"
  :attr_html     "#+attr_html:"
  :attr_org      "#+attr_org:"
  :begin_quote   "#+begin_quote"
  :end_quote     "#+end_quote"
  :caption       "#+caption:"
  :header        "#+header:"
  :begin_export  "#+begin_export"
  :end_export    "#+end_export"
  :filetags      "#+filetags:"
  :created       "#+created:"
  :include       "#+include:"
  :setupfile     "#+setupfile:"
  :results       "#+RESULTS:"
  :property      ":PROPERTIES:"
  :end           ":END:"
  :priority_a    "[#A]"
  :priority_b    "[#B]"
  :priority_c    "[#C]"
  :priority_d    "[#D]"
  :priority_e    "[#E]")
(plist-put +ligatures-extra-symbols :name "")
```

LaTeX Fragments

Prettier highlighting First off, we want those fragments to look good.

```
(setq org-highlight-latex-and-related '(native script entities))
```

However, by using `native` highlighting the `org-block` face is added, and that doesn't look too great — particularly when the fragments are previewed.

Ideally `org-src-font-lock-fontify-block` wouldn't add the `org-block` face, but we can avoid advising that entire function by just adding another face with `:inherit default` which will override the background color.

Inspecting `org-do-latex-and-related` shows that `"latex"` is the language argument passed, and so we can override the background as discussed above.

```
(require 'org-src)
(add-to-list 'org-src-block-faces '("latex" (:inherit default :extend t)))
```

Prettier rendering It's nice to customize the look of LaTeX fragments, so they fit better in the text — like this $\sqrt{\beta^2+3}-\sum_{\phi=1}^{\infty}\frac{x^{\phi}-1}{\Gamma(a)}$. Let's start by adding a sans font. I'd also like to use some functionality from `bmc-maths`, so we'll load that too.

```
(setq org-format-latex-header "\\documentclass{article}
\\usepackage[usenames]{xcolor}

\\usepackage[T1]{fontenc}

\\usepackage{booktabs}

\\pagestyle{empty}             % do not remove
% The settings below are copied from fullpage.sty
\\setlength{\\textwidth}{\\paperwidth}
\\addtolength{\\textwidth}{-3cm}
\\setlength{\\oddsidemargin}{1.5cm}
\\addtolength{\\oddsidemargin}{-2.54cm}
\\setlength{\\evensidemargin}{\\oddsidemargin}
\\setlength{\\textheight}{\\paperheight}
\\addtolength{\\textheight}{-\\headheight}
\\addtolength{\\textheight}{-\\headsep}
\\addtolength{\\textheight}{-\\footskip}
\\addtolength{\\textheight}{-3cm}
\\setlength{\\topmargin}{1.5cm}
\\addtolength{\\topmargin}{-2.54cm}
% my custom stuff
% \\usepackage[nofont,plaindd]{bmc-maths}
\\usepackage{arev}
")
```

Since we can, instead of making the background color match the `default` face, let's make it transparent.

```
(setq org-format-latex-options
      (plist-put org-format-latex-options :background "Transparent"))

(setq org-format-latex-options
      (plist-put org-format-latex-options :scale 0.5))

;; Can be dvipng, dvisvgm, imagemagick
(setq org-preview-latex-default-process 'dvipng)
```

Org Plot We can use some variables in `org-plot` to use the current doom theme colors.

```
(after! org-plot
  (defun org-plot/generate-theme (_type)
    "Use the current Doom theme colours to generate a GnuPlot preamble."
    (format "
fgt = \"textcolor rgb '%s'\" # foreground text
fgat = \"textcolor rgb '%s'\" # foreground alt text
fgl = \"linecolor rgb '%s'\" # foreground line
fgal = \"linecolor rgb '%s'\" # foreground alt line

# foreground colors
set border lc rgb '%s'
# change text colors of  tics
set xtics @fgt
set ytics @fgt
```

```
# change text colors of labels
set title @fgt
set xlabel @fgt
set ylabel @fgt
# change a text color of key
set key @fgt

# line styles
set linetype 1 lw 2 lc rgb '%s' # red
set linetype 2 lw 2 lc rgb '%s' # blue
set linetype 3 lw 2 lc rgb '%s' # green
set linetype 4 lw 2 lc rgb '%s' # magenta
set linetype 5 lw 2 lc rgb '%s' # orange
set linetype 6 lw 2 lc rgb '%s' # yellow
set linetype 7 lw 2 lc rgb '%s' # teal
set linetype 8 lw 2 lc rgb '%s' # violet

# palette
set palette maxcolors 8
set palette defined ( 0 '%s',\
1 '%s',\
2 '%s',\
3 '%s',\
4 '%s',\
5 '%s',\
6 '%s',\
7 '%s' )
"
            (doom-color 'fg)
            (doom-color 'fg-alt)
            (doom-color 'fg)
            (doom-color 'fg-alt)
            (doom-color 'fg)
            ;; colours
            (doom-color 'red)
            (doom-color 'blue)
            (doom-color 'green)
            (doom-color 'magenta)
            (doom-color 'orange)
            (doom-color 'yellow)
            (doom-color 'teal)
            (doom-color 'violet)
            ;; duplicated
            (doom-color 'red)
            (doom-color 'blue)
            (doom-color 'green)
            (doom-color 'magenta)
            (doom-color 'orange)
            (doom-color 'yellow)
            (doom-color 'teal)
            (doom-color 'violet)
            ))
  (defun org-plot/gnuplot-term-properties (_type)
    (format "background rgb '%s' size 1050,650"
            (doom-color 'bg)))
  (setq org-plot/gnuplot-script-preamble #'org-plot/generate-theme)
```

```
  (setq org-plot/gnuplot-term-extra #'org-plot/gnuplot-term-properties))
```

9.9.5 Exporting

General settings By default Org only exports the first three levels of headings as . . . headings. This is rather unfortunate as my documents frequently stray far beyond three levels of depth. The two main formats I care about exporting to are LaTeX and HTML. When using an `article` class, LaTeX headlines go from `\section`, `\subsection`, `\subsubsection`, and `\paragraph` to `\subgraph` — *five* levels. HTML5 has six levels of headings (`<h1>` to `<h6>`), but first level Org headings get exported as `<h2>` elements — leaving *five* usable levels.

As such, it would seem to make sense to recognize the first *five* levels of Org headings when exporting.

```
(setq org-export-headline-levels 5) ; I like nesting
```

I'm also going to make use of an item in `ox-extra` so that I can add an `:ignore:` tag to headings for the content to be kept, but the heading itself ignored (unlike `:noexport:` which ignored both heading and content). This is useful when I want to use headings to provide a structure for writing that doesn't appear in the final documents.

```
(require 'ox-extra)
(ox-extras-activate '(ignore-headlines))
```

Since I (roughly) track Org `HEAD`, it makes sense to include the git version in the creator string.

```
(setq org-export-creator-string
      (format "Emacs %s (Org mode %s %s)" emacs-version (org-release) (org-git-version)))
```

LaTeX Export

Compiling By default Org uses `pdflatex` × 3 + `bibtex`. This simply won't do in our modern world. `latexmk` + `biber` (which is used automatically with `latexmk`) is a simply superior combination.

```
;; `org-latex-compilers' contains a list of possible values ("pdflatex" "xelatex" "lualatex")
;; for the `%latex' argument.
(setq org-latex-pdf-process '("latexmk -f -pdf -%latex -shell-escape -interaction=nonstopmode -output-d
;;(setq org-latex-pdf-process (list "latexmk -shell-escape -bibtex -f -pdf %f"))
;(setq org-latex-pdf-process
;      '("pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"
;        "pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"
;        "pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"))
```

While `org-latex-pdf-process` does support a function, and we could use that instead, this would no longer use the log buffer — it's a bit blind, you give it the file name and expect it to do its thing.

The default values of `org-latex-compilers` is given in commented form to see how `org-latex-pdf-process` works with them.

While the `-%latex` above is slightly hacky (`-pdflatex` expects to be given a value) it allows us to leave `org-latex-compilers` unmodified. This is nice in case I open an org file that uses `#+LATEX_COMPILER` for example, it should still work.

Export PDFs with syntax highlighting

```
(require 'ox-latex)

(add-to-list 'org-latex-packages-alist '("" "minted"))

(setq org-latex-listings 'minted
      org-src-fontify-natively t)
```

10 System configuration

10.1 Mime types

10.1.1 Org Mode files

Org mode isn't recognized as its own mime type by default, but that can easily be changed with the following file. For system-wide changes try `/usr/share/mime/packages/org.xml`.

```
<mime-info xmlns='http://www.freedesktop.org/standards/shared-mime-info'>
  <mime-type type="text/org">
    <comment>Emacs Org-mode File</comment>
    <glob pattern="*.org"/>
    <alias type="text/org"/>
  </mime-type>
</mime-info>
```

What's nice is that Papirus now has an icon for `text/org`. One simply needs to refresh their mime database

```
update-mime-database ~/.local/share/mime
```

Then set Emacs as the default editor:

```
xdg-mime default emacs-client.desktop text/org
```

10.1.2 Registering `org-protocol://`

The recommended method of registering a protocol is by registering a desktop application, which seems reasonable.

To associate `org-protocol://` links with the desktop file:

```
xdg-mime default org-protocol.desktop x-scheme-handler/org-protocol
```

10.1.3 Configuring Chrome/Brave

As specified in the official documentation, we would like to invoke the `org-protocol://` without confirmation. To do this, we need to add this system-wide configuration.

```
echo "Setting Chrome/Brave to show the 'Always open ...' checkbox, to be used with the 'org-protocol://

sudo mkdir -p /etc/opt/chrome/policies/managed/

sudo tee /etc/opt/chrome/policies/managed/external_protocol_dialog.json >/dev/null <<'EOF'
{
  "ExternalProtocolDialogShowAlwaysOpenCheckbox": true
}
EOF

sudo chmod 644 /etc/opt/chrome/policies/managed/external_protocol_dialog.json
```

Then add a bookmarklet in your browser with this code:

```
javascript:location.href =
    'org-protocol://roam-ref?template=r&ref='
    + encodeURIComponent(location.href)
    + '&title='
    + encodeURIComponent(document.title)
    + '&body='
    + encodeURIComponent(window.getSelection())
```

10.2 Git

10.2.1 Git diffs

Based on this gist and this article.

Then adding a regex for it to `~/.config/git/config`

10.2.2 Apache Tika App wrapper

Apache Tika is a content detection and analysis framework. It detects and extracts metadata and text from over a thousand different file types. We will be using the Tika App in command-line mode to show some meaningful diff information for some binary files.

First, let's add a custom script to run `tika-app`:

```
#!/bin/sh
APACHE_TIKA_JAR="$HOME/.local/share/tika/tika-app.jar"

if [ -f ${APACHE_TIKA_JAR} ]
then
  exec java -Dfile.encoding=UTF-8 -jar ${APACHE_TIKA_JAR} "$@" 2>/dev/null
else
  echo "JAR file not found at ${APACHE_TIKA_JAR}"
fi
```

Add `tika`'s installation instructions to the `setup.sh` file.

```
update_apache_tika () {
  TIKA_JAR_PATH=$HOME/.local/share/tika

  if [ ! -d ${TIKA_JAR_PATH} ]
  then
    mkdir -p ${TIKA_JAR_PATH}
  fi

  TIKA_BASE_URL=https://archive.apache.org/dist/tika/
  TIKA_JAR_LINK="${TIKA_JAR_PATH}/tika-app.jar"

  echo -n "Checking for new Apache Tika App version... "

  # Get the lastest version
  TIKA_VERSION=$(
    curl -s ${TIKA_BASE_URL} | # Get the page
    pandoc -f html -t plain | # Convert HTML page to plain text.
    awk '/([0-9]+\.)+[0-1]\// {print substr($1, 0, length($1)-1)}' | # Get the versions directories (pa
    sort -rV | # Sort versions, the newest first
    head -n 1 # Get the first (newest) version
  )

  if [ -z ${TIKA_VERSION} ]
  then
    echo "Failed, check your internet connection."
    exit 1
  fi

  echo "Lastest version is ${TIKA_VERSION}"

  TIKA_JAR="${TIKA_JAR_PATH}/tika-app-${TIKA_VERSION}.jar"
  TIKA_JAR_URL="${TIKA_BASE_URL}${TIKA_VERSION}/tika-app-${TIKA_VERSION}.jar"
```

```
  if [ ! -f ${TIKA_JAR} ]
  then
    echo "New version available!"
    read -p "Do you want to download Apache Tika App v${TIKA_VERSION}? [Y | N]: " INSTALL_CONFIRM
    if [[ $INSTALL_CONFIRM == "Y" ]]
    then
      curl -o ${TIKA_JAR} ${TIKA_JAR_URL} && echo "Apache Tika App v${TIKA_VERSION} downloaded successf
    fi
  else
    echo "Apache Tika App is up-to-date, version ${TIKA_VERSION} already downloaded to '${TIKA_JAR}'"
  fi

  # Check the existance of the symbolic link
  if [ -L ${TIKA_JAR_LINK} ]
  then
    unlink ${TIKA_JAR_LINK}
  fi

  # Create a symbolic link to the installed version
  ln -s ${TIKA_JAR} ${TIKA_JAR_LINK}
}

update_apache_tika;
```

When it detects that Tesseract is installed, Tika App will try to extract text from some file types. For some reason, it tries to use Tesseract with some compressed files like `*.bz2`, `*.apk`... etc. I would like to disable this feature by exporting an XML config file which will be used when launching the Tika App (using `--config=<tika-config.xml>`).

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
  <parsers>
    <parser class="org.apache.tika.parser.DefaultParser">
      <parser-exclude class="org.apache.tika.parser.ocr.TesseractOCRParser"/>
    </parser>
  </parsers>
</properties>
```

10.3 Emacs' Systemd Daemon

Let's define a Systemd service to launch Emacs server automatically.

Which is then enabled by:

```
systemctl --user enable emacs.service
```

For some reason if a frame isn't opened early in the initialization process, the daemon doesn't seem to like opening frames later — hence the `&& emacsclient` part of the `ExecStart` value.

10.4 Emacs Client

10.4.1 Desktop Integration

It can now be nice to use this as a 'default app' for opening files. If we add an appropriate desktop entry, and enable it in the desktop environment.

10.4.2 Command-line Wrapper

A wrapper around `emacsclient`:

- Accepting `stdin` by putting it in a temporary file and immediately opening it.
- Guessing that the `tty` is a good idea when `$DISPLAY` is unset (relevant with SSH sessions, among other things).
- With a whiff of 24-bit color support, sets `TERM` variable to a `terminfo` that (probably) announces 24-bit color support.
- Changes GUI `emacsclient` instances to be non-blocking by default (`--no-wait`), and instead take a flag to suppress this behavior (`-w`).

I would use `sh`, but using arrays for argument manipulation is just too convenient, so I'll raise the requirement to `bash`. Since arrays are the only 'extra' compared to `sh`, other shells like `ksh` etc. should work too.

```
#!/usr/bin/env bash
force_tty=false
force_wait=false
stdin_mode=""

args=()

usage () {
  echo -e "Usage: e [-t] [-m MODE] [OPTIONS] FILE [-]

Emacs client convenience wrapper.

Options:
-h, --help            Show this message
-t, -nw, --tty        Force terminal mode
-w, --wait            Don't supply --no-wait to graphical emacsclient
-                     Take stdin (when last argument)
-m MODE, --mode MODE  Mode to open stdin with
-mm, --maximized      Start Emacs client in maximized window

Run emacsclient --help to see help for the emacsclient."
}

while :
do
  case "$1" in
    -t | -nw | --tty)
      force_tty=true
      shift ;;
    -w | --wait)
      force_wait=true
      shift ;;
    -m | --mode)
      stdin_mode=" ($2-mode)"
      shift 2 ;;
    -mm | --maximized)
        args+=("--frame-parameters='(fullscreen . maximized)")
        shift ;;
    -h | --help)
      usage
      exit 0 ;;
```

```
    --*=*)
      set -- "$@" "${1%%=*}" "${1#*=}"
      shift ;;
    *)
      [ "$#" = 0 ] && break
      args+=("$1")
      shift ;;
  esac
done

if [ ! "${#args[*]}" = 0 ] && [ "${args[-1]}" = "-" ]
then
  unset 'args[-1]'
  TMP="$(mktemp /tmp/emacsstdin-XXX)"
  cat > "$TMP"
  args+=(--eval "(let ((b (generate-new-buffer \"*stdin*\"))) (switch-to-buffer b) (insert-file-content
fi

if [ -z "$DISPLAY" ] || $force_tty
then
  # detect terminals with sneaky 24-bit support
  if { [ "$COLORTERM" = truecolor ] || [ "$COLORTERM" = 24bit ]; } \
    && [ "$(tput colors 2>/dev/null)" -lt 257 ]
  then
    if echo "$TERM" | grep -q "^\w\+-[0-9]"
    then
      termstub="${TERM%%-*}"
    else
      termstub="${TERM#*-}"
    fi

    if infocmp "$termstub-direct" >/dev/null 2>&1
    then
      TERM="$termstub-direct"
    else
      TERM="xterm-direct"
    fi # should be fairly safe
  fi

  emacsclient --tty -create-frame --alternate-editor="" "${args[@]}"
else
  if ! $force_wait
  then
    args+=(--no-wait)
  fi

  emacsclient -create-frame --alternate-editor="" "${args[@]}"
fi
```

Useful aliases Now, to set an alias to use `e` with `magit`, and then for maximum laziness we can set aliases for the terminal-forced variants.

```
# Alias to run emacs client in terminal mode
alias et="e -t"

# Aliases to run emacs+magit
```

```
alias magit='e --eval "(progn (magit-status) (delete-other-windows))"'
alias magitt='e -t --eval "(progn (magit-status) (delete-other-windows))"'

# Aliases to run emacs+mu4e
alias emu='e --eval "(progn (=mu4e) (delete-other-windows))"'
alias emut='e -t --eval "(progn (=mu4e) (delete-other-windows))"'
```

10.5 TODO tmux

Configure remote/local mixed `tmux` configuration, an example in this repo and this article.

10.6 AppImage

Install/update the `appimageupdatetool.AppImage` tool:

```
update_appimageupdatetool () {
  TOOL_NAME=appimageupdatetool
  MACHINE_ARCH=$(uname -m)
  APPIMAGE_UPDATE_TOOL_PATH="$HOME/.local/bin/${TOOL_NAME}"
  APPIMAGE_UPDATE_TOOL_URL="https://github.com/AppImage/AppImageUpdate/releases/download/continuous/${T

  if [ -f ${APPIMAGE_UPDATE_TOOL_PATH} ] && $APPIMAGE_UPDATE_TOOL_PATH -j ${APPIMAGE_UPDATE_TOOL_PATH}
  then
    echo "${TOOL_NAME} already up to date"
  else
    if [ -f ${APPIMAGE_UPDATE_TOOL_PATH} ]
    then
      echo "Update available, downloading latest ${MACHINE_ARCH} version to ${APPIMAGE_UPDATE_TOOL_PATH
      mv ${APPIMAGE_UPDATE_TOOL_PATH} "${APPIMAGE_UPDATE_TOOL_PATH}.backup"
    else
      echo "${TOOL_NAME} not found, downloading latest ${MACHINE_ARCH} version to ${APPIMAGE_UPDATE_TOO
    fi
    wget -O ${APPIMAGE_UPDATE_TOOL_PATH} ${APPIMAGE_UPDATE_TOOL_URL} &&
        echo "Downloaded ${TOOL_NAME}-${MACHINE_ARCH}.AppImage" &&
        [ -f "${APPIMAGE_UPDATE_TOOL_PATH}.backup" ] &&
        rm "${APPIMAGE_UPDATE_TOOL_PATH}.backup"
    chmod a+x ${APPIMAGE_UPDATE_TOOL_PATH}
  fi
}

update_appimageupdatetool;
```

10.7 Custom environment

I would like to customize my Linux environment in a separate file, which I source from my `~/.zshrc` file.

I like to define MacOS-like commands (`pbcopy` and `pbpaste`) to copy and paste in terminal (from `stdin`, to `stdout`). The `pbcopy` and `pbpaste` are defined using either `xclip` or `xsel`, you would need to install these tools, otherwise we wouldn't define the aliases.

```
# Define aliases to 'pbcopy' and 'pbpaste'
if command -v xclip &> /dev/null
then
  # Define aliases using xclip
  alias pbcopy='xclip -selection clipboard'
  alias pbpaste='xclip -selection clipboard -o'
elif command -v xsel &> /dev/null
then
```

```
  # Define aliases using xsel
  alias pbcopy='xsel --clipboard --input'
  alias pbpaste='xsel --clipboard --output'
fi
```

And then define `gsuon` and `gsuoff` aliases to run graphical apps from terminal with root permissions, this requires `xhost`.

```
# To run GUI apps from terminal with root permissions
if command -v xhost &> /dev/null
then
  alias gsuon='xhost si:localuser:root'
  alias gsuoff='xhost -si:localuser:root'
fi
```

Define a `netpaste` command to paste to https://ptpb.pw.

```
# To copy the output of a command to ptpb.pw
alias netpaste='curl -F c=@- https://ptpb.pw'
```

Use NeoVIM instead of VIM to provide `vi` and `vim` commands.

```
# NeoVim
if command -v nvim &> /dev/null
then
  alias vim="nvim"
  alias vi="nvim"
fi
```

Add some aliases to work with the ESP-IDF framework.

```
if [ -d $HOME/sources-and-libs/esp-idf/ ]
then
  alias esp-prepare-env='source $HOME/sources-and-libs/esp-idf/export.sh'
  alias esp-update='echo "Updating ESP-IDF framework..." && cd $HOME/sources-and-libs/esp-idf && git pu
else
  echo "esp-idf repo not found. You can clone the esp-idf repo using 'git clone https://github.com/espr
fi
```

For the moment, I'm not using a particular tool to manage my dotfiles, instead, I use a bare Git repository to manage files, when the workspace is set to the home directory. To be able to add/commit files to the dotfiles repository, I define an alias to `git` which takes the bare repository as `--git-dir`, and my home directory as `--work-tree`.

```
alias dotfiles='git --git-dir=$HOME/Projects/dotfiles.git --work-tree=$HOME'
```

Define an alias to get weather information for my city:

```
export WTTRIN_CITY=Orsay

alias wttrin='curl wttr.in/$WTTRIN_CITY'
alias wttrin2='curl v2.wttr.in/$WTTRIN_CITY'
```

Enable Meta key and colors in `minicom`:

```
export MINICOM='-m -c on'
```

Define Rust sources path, and add packages installed from `cargo` to the `PATH`.

```
export RUST_SRC_PATH=$HOME/.rustup/toolchains/stable-x86_64-unknown-linux-gnu/lib/rustlib/src/rust/src/
export PATH=$PATH:$HOME/.cargo/bin
```

I'm using the AUR package `clang-format-static-bin`, which provide multiple versions of Clang-format, I use it with some work projects requiring a specific version of Clang-format.

```
export PATH=/opt/clang-format-static:$PATH
```

Add my manually installed libraries to CMake and `PATH`.

```
export CMAKE_PREFIX_PATH=$HOME/sources-and-libs/build_installs
export PATH=$PATH:$HOME/.cargo/bin:$HOME/sources-and-libs/build_installs/bin
```

Set NPM installation path to local:

```
NPM_PACKAGES="${HOME}/.npm-packages"

# Export NPM bin path
export PATH="$PATH:$NPM_PACKAGES/bin"

# Preserve MANPATH if you already defined it somewhere in your config.
# Otherwise, fall back to `manpath` so we can inherit from `/etc/manpath`.
export MANPATH="${MANPATH-$(manpath)}:$NPM_PACKAGES/share/man"
```

Some useful stuff (`fzf`, `opam`, Doom Emacs...)

```
# FZF
[ -f ~/.fzf.zsh ] && source ~/.fzf.zsh

# opam configuration
[[ ! -r $HOME/.opam/opam-init/init.zsh ]] || source $HOME/.opam/opam-init/init.zsh  > /dev/null 2> /dev

# Add ~/.emacs-doom.d/bin to path (for DOOM Emacs stuff)
export PATH=$PATH:$HOME/.emacs-doom/bin
```

I like to use `tmux` by default, even on my local sessions, I like to start a `tmux` in a `default` session on the first time I launch a terminal, and then, attach any other terminal to this default session:

```
if command -v tmux &> /dev/null && [ -z "$TMUX" ]
then
    tmux attach -t default || tmux new -s default
fi
```

10.8 Zotero UI trick

Zotero does not support dark mode (ATM), when using a system-wide dark theme (at least on KDE), Zotero UI gets messed up, to fix this, we can force Zotero to use its default GTK theme by defining the `GTK_THEME=Default`.

10.9 Rust format

For Rust code base, the file `$HOME/.rustfmt.toml` contains the global format settings, I like to set it to:

10.10 GDB

10.10.1 Early init

I like to disable the initial message (containing copyright info and other stuff), the right way to do this is either by starting `gdb` with `-q` option, or (since GDB v11 I think), by setting in `~/.gdbearlyinit`.

10.10.2 Init

GDB loads `$HOME/.gdbinit` at startup, I like to define some default options in this file, this is a WIP, but won't evolve too much, as it is recommended to keep the `.gdbinit` simple. For the moment, it does just enable pretty printing, and defines `c` and `n` commands to wrap `continue` and `next` with a post `refresh`, this is just to avoid the annoying TUI when the program outputs to the stdout.

WIP: Guile Scheme per program/project script I often debug programs with a lot of arguments, I like to be able to set the arguments and the binary file to be launched in a per project script (currently using Guile Scheme). This bit of code checks if the `gdb.scm` file exists in the working directory, and if so, loads it.

A more flexible way is to provide a per program config files (to debug a program named `fft`, I like to create a script named `fft.scm` which gets loaded after the file). The following is a WIP, for the moment, I need to call my custom command `dbg-guile` when GDB done loading symbols from the file, otherwise, the used `(current-progspace)` returns an object with no filename. I need a mechanism to hook the `(dbg-find-and-load)` to GDB's load file functionality.

```
;; Set the Scheme implementation to Guile to avoid annoying
;; message when working with babel
;; (setq geiser-scheme-implementation 'guile)

(use-modules (gdb))

(define (dbg-check-and-load filename)
  (if (file-exists? filename)
    (begin (display (string-append "Found a Guile Scheme script, loading file " filename "\n"))
           (load filename)
           #t)
    #f))

(define (dbg-find-and-load)
  ;; Get the program name from the current progspace
  ;; For a program named "prog", the priorities goes like this:
  ;; 1. a script with the same program name (prog.scm) exists in the current directory
  ;; 2. a script with the same program name (prog.scm) exists in the program directory
  ;; 3. a script with the name (gdb.scm) exists in the current directory
  (let ((dbg-prg-filename (progspace-filename (current-progspace))))
    (if dbg-prg-filename
      (or (dbg-check-and-load (string-append (basename dbg-prg-filename) ".scm"))
          (dbg-check-and-load (string-append dbg-prg-filename ".scm")))
      (dbg-check-and-load "gdb.scm"))))

;; Run by default
(dbg-find-and-load)

;; Define a command to load binary specific config
(register-command! (make-command "dbg-guile" #:invoke (lambda (self arg from-tty) (dbg-find-and-load))))
```

In my project, I create a `gdb.scm` (or `<program-name>.scm`) with something like this:

```
;; Load program executable
(execute "file ./build/bin/my_program")

;; Load program arguments
(execute (string-join '("set args "
                        "arg1"
                        "--param=arg2")))
```

10.11 Latex

Set `texmf` path, to be used to install `https://github.com/tecosaur/BMC`

```
export TEXMFHOME=$HOME/.texmf
```

Clone and install the BMC repo, check for updates from time to time.

```
check_and_install_bmc () {
  [ ! -d $TEXMFHOME ] && mkdir -p $TEXMFHOME

 cd $TEXMFHOME

  RUN_TEXHASH=true
  BMC_DIR="${TEXMFHOME}/BMC"

  if [ -d $BMC_DIR ]
  then
    echo "BMC already installed in $BMC_DIR, checking for updates..."
    cd $BMC_DIR

    if git pull origin master | grep -iq "up to date"
    then
      echo "BMC is up-to-date!"
      RUN_TEXHASH=false
    else
      echo "BMC updated successfully!"
    fi
  else
    echo "BMC not installed!, installing to $BMC_DIR"
    git clone https://github.com/tecosaur/BMC.git $BMC_DIR
  fi

  if $RUN_TEXHASH
  then
    texhash
  fi
}

check_and_install_bmc
```