Doom Emacs Configuration

Emacs configuration for work and life!

Abdelhak Bougouffa*

May 23, 2022

Contents

*abougouffa@fedoraproject.org

1 This repository

This repository (abougouffa/dotfiles) contains my configuration files for **Zsh**, **Emacs**, **Vim**, **Alacritty** and other Linux related stuff.

If you want to reuse some of these configurations, you will need to modify some directories and add some user specific information (usernames, passwords…)

This is the main configuration file .doom.d/config.org, (available also as a PDF file, it contains the literal configuration for Doom Emacs, and I use it to generate some other user configuration files (define aliases, environment variables, user tools, Git configuration…).

1.1 How to install

Since commit 55c92810, I'm using **chezmoi** to manage my Dotfiles.

Now the Dotfiles can be installed using:

```
sudo pacman -S chezmoi
chezmoi init --apply abougouffa
```

1.2 Emacs stuff

- You will need to install Chemacs2 to `.emacs.d`:

```
[ -f ~/.emacs ] && mv ~/.emacs ~/.emacs.bak
[ -d ~/.emacs.d ] && mv ~/.emacs.d ~/.emacs.default

git clone https://github.com/plexus/chemacs2.git ~/.emacs.d
```

- Install Doom Emacs to `~/.config/emacs.doom` instead of `.emacs.d`:

```
git clone https://github.com/hlissner/doom-emacs.git ~/.config/emacs.doom

~/.config/emacs.doom/bin/doom install
```

- You can install other Emacs distributions like Spacemacs; lets clone it to `~/.config/emacs.spacemacs`, however, I'm not using it anymore, it can just be a fall back solution if I mess up my Doom config and I want to do something quickly:

  `git clone https://github.com/syl20bnr/spacemacs.git ~/.config/emacs.spacemacs`

- You can now customize `~/.emacs-profiles.el` if you want to.

2 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 one.

2.1 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.

```
#!/bin/bash

# This is an automatically generated setup file, it installes some missing
# dependencies, configure system services, set system settings form better
# desktop integration... etc.
# Abdelhak BOUGOUFFA (c) 2022
```

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`).

```
#!/bin/zsh

# This is an automatically generated file, it should be sourced from `~/.zshrc',
# it defines some useful aliases, and customize some environment variables for
# better defaults.
# Abdelhak BOUGOUFFA (c) 2022
```

3 Doom configuration files

3.1 Pseudo early-init

This file will be loaded before the content of Doom's private `init.el`, I add some special stuff which I want to load very early.

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

;; Fix for #2386 until further investigation
;; From https://git.sr.ht/~gagbo/doom-config
(when noninteractive
  (after! undo-tree
    (global-undo-tree-mode -1)))
```

3.2 Doom 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.

3.2.1 File skeleton

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

```
;;; 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.

;; I add some special stuff wich I want to load very early.
(load! "pseudo-early-init.el")

(doom!
  :input
  <<doom-input>>

  :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>>

)
```

3.2.2 Input (:input)

Enable bidirectional languages support (bidi).

```
bidi
```

3.2.3 General (:config)

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

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

3.2.4 Completion (:completion)

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

```
(vertico +icons)
company
```

3.2.5 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
doom
doom-dashboard
hl-todo
hydra
modeline
vc-gutter
workspaces
zen
ophints
nav-flash
(window-select +numbers)
(ligatures +extra)
(popup +all +defaults)
(emoji +ascii +unicode +github)
(treemacs +lsp)
;;tabs
;;unicode
;;neotree
;;doom-quit
;;indent-guides
;;minimap
;;vi-tilde-fringe
```

3.2.6 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.

```
(evil +everywhere)
file-templates
fold
format
multiple-cursors
parinfer
snippets
word-wrap
```

```
;;lispy
;;(objed +manual)
;;god
;;rotate-text
```

3.2.7 Emacs builtin stuff (:emacs)

Beautify Emacs builtin packages.

```
(dired +icons)
(ibuffer +icons)
(undo +tree)
vc
;;electric
```

3.2.8 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
vterm
shell
term
```

3.2.9 Checkers (:checkers)

I like to check my documents for errors while I'm typing. The `grammar` module enables LanguageTool support.

```
(syntax +childframe)
(spell +aspell)
grammar
```

3.2.10 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.

```
direnv
editorconfig
ein
gist
make
pdf
rgb
tmux
upload
(lsp +peek)
(debugger +lsp)
(docker +lsp)
(eval +overlay)
(lookup +docsets +dictionary +offline)
(magit +forge)
;;pass
;;biblio
```

```
;;ansible
;;prodigy
;;taskrunner
;;terraform
```

3.2.11 Operating system (:os)

I enable tty for better support of terminal editing.

```
(tty +osc)
```

3.2.12 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
emacs-lisp
common-lisp
markdown
data
qt
(cc +lsp)
(json +lsp)
(julia +lsp)
(latex +lsp +latexmk +fold)
(rust +lsp)
(ess +lsp)
(yaml +lsp)
(sh +lsp)
(python +lsp +pyenv +conda +pyright)
(racket +lsp +xp)
(scheme +mit +guile +racket +chez)
(org +dragndrop +gnuplot +jupyter +pandoc +noter +hugo +present +pomodoro +roam2)
;;rst                 ; ReST in peace
;;(lua +lsp)          ; one-based indices? one-based indices
;;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
```

3.2.13 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)
```

3.2.14 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
emms
everywhere
(rss +org)
;;twitter
```

3.3 Additional packages (packages.el)

This section generates Doom's `packages.el`, with the associated configurations (`use-package!` blocks).

This file shouldn't be byte compiled.

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

4 General Emacs settings

4.1 User information

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

4.2 Secrets

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

```
(setq auth-sources '("~/.authinfo.gpg")
      auth-source-do-cache t
      auth-source-cache-expiry 86400 ; All day, defaut is 2h (7200)
      password-cache t
      password-cache-expiry 86400)

;; Set my GPG key as the default key
(setq-default epa-file-encrypt-to '("F808A020A3E1AC37"))
```

4.3 Better defaults

4.3.1 File deletion

Delete files by moving them to trash.

```
(setq-default delete-by-moving-to-trash t
              trash-directory nil) ;; Use freedesktop.org trashcan
```

4.3.2 Window

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

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

Stick to buffer tail, useful with `*Messages*` buffer. Derived from this answer.

```
(defvar +messages-buffer-auto-tail--enabled nil)

(defun +messages-buffer-auto-tail--advice (&rest arg)
  "Make *Messages* buffer auto-scroll to the end after each message."
  (let* ((buf-name (buffer-name (messages-buffer)))
         ;; Create *Messages* buffer if it does not exist
         (buf (get-buffer-create buf-name)))
    ;; Activate this advice only if the point is _not_ in the *Messages* buffer
    ;; to begin with. This condition is required; otherwise you will not be
    ;; able to use `isearch' and other stuff within the *Messages* buffer as
    ;; the point will keep moving to the end of buffer :P
    (when (not (string= buf-name (buffer-name)))
      ;; Go to the end of buffer in all *Messages* buffer windows that are
      ;; *live* (`get-buffer-window-list' returns a list of only live windows).
      (dolist (win (get-buffer-window-list buf-name nil :all-frames))
        (with-selected-window win
          (goto-char (point-max))))
      ;; Go to the end of the *Messages* buffer even if it is not in one of
      ;; the live windows.
      (with-current-buffer buf
        (goto-char (point-max))))))

(defun +messages-buffer-toggle-auto-tail ()
  "Auto tail the '*Messages*' buffer."
  (interactive)
  ;; Add/remove an advice from the 'message' function.
  (cond (+messages-buffer-auto-tail--enabled
         (advice-remove 'message '+messages-buffer-auto-tail--advice)
         (setq +messages-buffer-auto-tail--enabled nil)
         (message "+messages-buffer-auto-tail: Disabled."))
        (t
         (advice-add 'message :after '+messages-buffer-auto-tail--advice)
         (setq +messages-buffer-auto-tail--enabled t)
         (message "+messages-buffer-auto-tail: Enabled."))))
```

Split defaults 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))
```

4.3.3 Undo and auto-save

There is a package `https://github.com/bbatsov/super-save`, maybe better than the default `auto-save-mode`.

```
(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
```

4.3.4 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)
```

4.3.5 Emacs sources

```
(setq source-directory
      (expand-file-name "~/Softwares/aur/emacs-git/src/emacs-git"))
```

4.3.6 Frame

```
;; 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))
```

Maximizing

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), 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.

```
;; (set-frame-parameter nil 'internal-border-width 15)

;; Add frame borders and window dividers
(modify-all-frames-parameters '((right-divider-width . 10)
                                (internal-border-width . 10)))

(dolist (face '(window-divider window-divider-first-pixel window-divider-last-pixel))
  (face-spec-reset-face face)
  (set-face-foreground face (face-attribute 'default :background)))

(set-face-background 'fringe (face-attribute 'default :background))
```

Margins

4.4 Chemacs2

Add Chemacs2 profiles for a set of Emacs configurations. I'm using DOOM Emacs, however, I like to try other configs to get inspired!

```
(("doom"      . ((user-emacs-directory . "~/.config/emacs.doom")
                 (env . (("DOOMDIR" . "~/.doom.d")))))
 ("rational"  . ((user-emacs-directory . "~/.config/emacs.rational")
                 (env . (("RATIONAL_EMACS_HOME" . "~/.config/emacs.rational")))))
 ("scimax"    . ((user-emacs-directory . "~/.config/emcas.scimax")))
 ("spacemacs" . ((user-emacs-directory . "~/.config/emacs.spacemacs"))))
```

Make `doom` the default profile.

```
doom
```

5 Emacs daemon

5.1 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 +greedily-do-daemon-setup ()
  (require 'org)
  (when (require 'mu4e nil t)
    (setq mu4e-confirm-quit t
          +mu4e-lock-greedy t
          +mu4e-lock-relaxed t)
    (+mu4e-lock-start 'mu4e~start))
  (when (require 'elfeed nil t)
    (run-at-time nil (* 8 60 60) #'elfeed-update)))

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

5.2 Tweaks

5.2.1 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))))
```

6 Package configuration

6.1 Check for external tools

Some of the added packages require external tools, I like to check for these tools and store the result in global constants.

```
(defun bool (val) (not (null val))) ;; Convert a value to boolean

(defconst +zotero-ok-p (bool (executable-find "zotero")))
(defconst +ag-ok-p (bool (executable-find "ag")))
(defconst +chezmoi-ok-p (bool (executable-find "chezmoi")))
(defconst +bitwarden-ok-p (bool (executable-find "bw")))
(defconst +repo-ok-p (bool (executable-find "repo")))
(defconst +delta-ok-p (bool (executable-find "delta")))
(defconst +maxima-ok-p (bool (executable-find "maxima")))
(defconst +eaf-ok-p (bool (file-directory-p (expand-file-name "lisp/emacs-application-framework"))))
(defconst +quarto-ok-p (bool (executable-find "quarto")))
(defconst +clang-format-ok-p (bool (executable-find "clang-format")))
(defconst +inkscape-ok-p (bool (executable-find "inkscape")))
(defconst +rosbag-ok-p (bool (executable-find "rosbag")))

(defconst +netextender-ok-p
  (let ((ok (bool (and (executable-find "netExtender")
                       (file-exists-p "~/.local/bin/netextender")
                       (file-exists-p "~/.ssh/netExtender-params.gpg")))))
    (unless ok (warn "Missing netExtender dependencies."))
    ok)
  "Evaluates to 't' when a valid netExtender configuration is present, 'nil' otherwise.")

(defconst +mpd-ok-p
  (let ((ok (bool (and (executable-find "mpc") (executable-find "mpd")))))
    (unless ok (warn "Missing MPD or MPC. Falling back to the EMMS default backend."))
```

```
    ok)
  "Evaluates to 't' when MPD and MPC commands are present, 'nil' otherwise.")

(defconst +mpv-ok-p
  (let ((ok (bool (and +mpd-ok-p
                       (executable-find "mpv")
                       (executable-find "youtube-dl")))))
    (unless ok (warn "Missing MPV or youtube-dl."))
    (and nil ok)) ;; NOTE: disabled
  "Evaluates to 't' when MPV and youtube-dl commands are present, 'nil' otherwise.")
```

6.2 User interface

6.2.1 Font

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)
- Nerd fonts
  - `FantasqueSansMono Nerd Font Mono`
  - `mononoki Nerd Font Mono`
  - `CaskaydiaCove Nerd Font Mono`
- `Cascadia Code`
- `Fantasque Sans Mono`
- `JuliaMono` (good Unicode support)
- `IBM Plex Mono`
- `JetBrains Mono`
- `Roboto Mono`
- `Source Code Pro`
- `Input Mono Narrow`
- `Fira Code`

```
(setq doom-font (font-spec :family "FantasqueSansMono Nerd Font Mono" :size 20)
      doom-variable-pitch-font (font-spec :family "Andika") ;; inherits the :size from doom-font
      doom-unicode-font (font-spec :family "JuliaMono")
      doom-serif-font (font-spec :family "FantasqueSansMono Nerd Font Mono" :weight 'light))
```

6.2.2 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-vibrant)
(remove-hook 'window-setup-hook #'doom-init-theme-h)
(add-hook 'after-init-hook #'doom-init-theme-h 'append)
(delq! t custom-theme-load-path)

;; By default 'doom-vibrant' uses red faces to mark modified file in modeline,
;; lets change it to orange.
(custom-set-faces!
  '(doom-modeline-buffer-modified :foreground "orange"))
```

6.2.3 Mode line

Clock Display time and set the format to 24h.

```
(after! doom-modeline
  (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.

```
(after! doom-modeline
  (let ((battery-str (battery)))
     (unless (or (equal "Battery status not available" battery-str)
                 (string-match-p (regexp-quote "unknown") battery-str)
                 (string-match-p (regexp-quote "N/A") battery-str))
      (display-battery-mode 1))))
```

```
(setq doom-modeline-major-mode-icon t
      doom-modeline-major-mode-color-icon t
      doom-modeline-buffer-state-icon t
      doom-modeline-github t)
```

Mode line customization

6.2.4 Set transparency

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

6.2.5 Dashboard

Custom Splash Image Change the logo to an image, a set of beautiful images can be found in `assets`.

File
emacs-e.svg
gnu-emacs-white.svg
gnu-emacs-flat.svg
blackhole-lines.svg
doom-emacs-white.svg
doom-emacs-dark.svg

```
(setq fancy-splash-image (expand-file-name "assets/emacs-e.png" doom-private-dir))
```

```
;; (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))

(defun +doom/open-private-config-org ()
  (interactive)
  (when (file-directory-p doom-private-dir)
    (find-file (expand-file-name "config.org" doom-private-dir))))

(setq +doom-dashboard-menu-sections
  '(("Reload last session"
     :icon (all-the-icons-octicon "history" :face 'doom-dashboard-menu-title)
     :when (cond ((featurep! :ui workspaces)
                  (file-exists-p (expand-file-name persp-auto-save-fname persp-save-dir)))
                 ((require 'desktop nil t)
                  (file-exists-p (desktop-full-file-name))))
     :face (:inherit (doom-dashboard-menu-title bold))
     :action doom/quickload-session)
    ("Open mailbox"
     :icon (all-the-icons-octicon "mail" :face 'doom-dashboard-menu-title)
     :action =mu4e)
    ("Open org-agenda"
     :icon (all-the-icons-octicon "calendar" :face 'doom-dashboard-menu-title)
     :when (fboundp 'org-agenda)
     :action org-agenda)
    ("Recently opened files"
     :icon (all-the-icons-octicon "file-text" :face 'doom-dashboard-menu-title)
     :action recentf-open-files)
    ("Open project"
     :icon (all-the-icons-octicon "briefcase" :face 'doom-dashboard-menu-title)
     :action projectile-switch-project)
    ("Jump to bookmark"
     :icon (all-the-icons-octicon "bookmark" :face 'doom-dashboard-menu-title)
     :action bookmark-jump)
    ("Open config.org"
     :icon (all-the-icons-fileicon "config" :face 'doom-dashboard-menu-title)
     :when (file-directory-p doom-private-dir)
     :action +doom/open-private-config-org)))
```

```
39
40 (defun +doom-dashboard-setup-modified-keymap ()
41   (setq +doom-dashboard-mode-map (make-sparse-keymap))
42   (map! :map +doom-dashboard-mode-map
43         :desc "Find file" :ne "f" #'find-file
44         :desc "Recent files" :ne "r" #'consult-recent-file
45         :desc "Config dir" :ne "C" #'doom/open-private-config
46         :desc "Open config.org" :ne "c" #'+doom/open-private-config-org
47         :desc "Open dotfile" :ne "." (cmd! (doom-project-find-file "~/.config/"))
48         :desc "Notes (roam)" :ne "n" #'org-roam-node-find
49         :desc "Switch buffer" :ne "b" #'+vertico/switch-workspace-buffer
50         :desc "Switch buffers (all)" :ne "B" #'consult-buffer
51         :desc "IBuffer" :ne "i" #'ibuffer
52         :desc "Previous buffer" :ne "p" #'previous-buffer
53         :desc "Email" :ne "m" #'=mu4e
54         :desc "Quit" :ne "Q" #'save-buffers-kill-terminal
55         :desc "Show keybindings" :ne "h" (cmd! (which-key-show-keymap '+doom-dashboard-mode-map))))
56
57 (add-transient-hook! #'+doom-dashboard-mode (+doom-dashboard-setup-modified-keymap))
58 (add-transient-hook! #'+doom-dashboard-mode :append (+doom-dashboard-setup-modified-keymap))
59 (add-hook! 'doom-init-ui-hook :append (+doom-dashboard-setup-modified-keymap))
60
61 (map! :leader :desc "Dashboard" "d" #'+doom-dashboard/open)
```

Dashboard

6.2.6 Which key

Make `which-key` popup faster.

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

I stol this chunk from tecosaur's config, it helps replacing the `evil-` prefix with a unicode char, making `which-key`'s candidate list less verbose.

```
1 (setq which-key-allow-multiple-replacements t)
2
3 (after! which-key
4   (pushnew! which-key-replacement-alist
5             '(("" . "\\`+?evil[-:]?\\(?:a-\\)?\\(.*\\)") . (nil . " \\1"))
6             '(("\\`g s" . "\\`evilem--?motion-\\(.*\\)") . (nil . " \\1"))))
```

6.2.7 Window title

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

```
1 (setq frame-title-format
2       '(""
3         (:eval
4          (if (s-contains-p org-roam-directory (or buffer-file-name ""))
5              (replace-regexp-in-string
6               ".*/[0-9]*-?" "  "
7               (subst-char-in-string ?_ ?  buffer-file-name))
8            "%b"))
9         (:eval
10         (let ((project-name (projectile-project-name)))
11           (unless (string= "-" project-name)
12             (format (if (buffer-modified-p) "  %s" "  %s") project-name))))))
```

6.2.8 Fringe

Increase the left fringe width, to enable rendering breakpoints (in debug modes) correctly.

```
;; (after! lsp-mode
;;   (add-hook 'lsp-mode-hook (lambda () (set-fringe-mode '(15 . 15)))))

(setq-default left-fringe-width 25
              right-fringe-width 25)
```

6.2.9 Vertico

Since doom-emacs@ece4a74, Doom supports the `+childframe` for `:completion vertico`. This can be used to adjust the left and right fringes.

```
(after! vertico-posframe
  (setq vertico-posframe-parameters '((left-fringe . 12) (right-fringe . 14))
        vertico-posframe-border-width 3))
```

6.2.10 Company

I do not find `company` useful in Org files.

```
(setq company-global-modes
      '(not erc-mode
            circe-mode
            message-mode
            help-mode
            gud-mode
            vterm-mode
            org-mode))
```

6.2.11 SVG Tag mode

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

```
(use-package! svg-tag-mode
  :commands svg-tag-mode
  :config
  (setq svg-tag-tags
      '(("^\\*.* .* \\(:[A-Za-z0-9]+\\)" .
         ((lambda (tag) (svg-tag-make)
                    tag
                    :beg 1
                    :font-family "Roboto Mono"
                    :font-size 6
                    :height 0.6
                    :padding 0
                    :margin 0)))
        ("\\(:[A-Za-z0-9]+:\\)$" .
         ((lambda (tag) (svg-tag-make)
                    tag
                    :beg 1
                    :end -1
                    :font-family "Roboto Mono"
                    :font-size 6
                    :height 0.6
                    :padding 0
                    :margin 0))))))
```

6.2.12 Focus

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

```
(package! focus)
```

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

6.2.13 Smooth scrolling

```
(when (<= emacs-major-version 28)
  (package! good-scroll))
```

```
(if (> emacs-major-version 28)
    (pixel-scroll-precision-mode 1)
  (use-package! good-scroll
    :config (good-scroll-mode 1)))
```

6.2.14 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))))
```

6.3 Editing

6.3.1 Scratch buffer

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

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

6.3.2 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)
```

6.3.3 Page break lines

A feature that displays ugly form feed characters as tidy horizontal rules. Inspired by M-EMACS.

```
(package! page-break-lines)
```

```
(use-package! page-break-lines
  :diminish
  :init (global-page-break-lines-mode))
```

6.3.4 Binary files

Taken from this answer.

```
(defun +hexl/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/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 (+hexl/buffer-binary-p)
      (hexl-mode))))

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

6.3.5 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)
```

6.3.6 Evil

I'm not using `evil-escape`, lets disable it.

```
(package! evil-escape :disable t)
```

```
(after! evil
  (setq evil-kill-on-visual-paste nil)) ; Don't put overwritten text in the kill ring
```

6.3.7 Aggressive indent

```
(package! aggressive-indent)
```

```
(use-package! aggressive-indent
  :commands (aggressive-indent-mode))
```

6.3.8 YASnippet

Nested snippets are good, enable that.

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

6.4 Literate configuration

6.4.1 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)
```

6.4.2 Asynchronous 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.

```
(defvar +literate-tangle--proc nil)
(defvar +literate-tangle--proc-start-time nil)

(defadvice! +literate-tangle-async-h ()
  "A very simplified version of `+literate-tangle-h', but async."
  :override #'+literate-tangle-h
  (unless (getenv "__NOTANGLE")
    (let ((default-directory doom-private-dir))
      (when +literate-tangle--proc
        (message "Killing outdated tangle process...")
        (set-process-sentinel +literate-tangle--proc #'ignore)
        (kill-process +literate-tangle--proc)
        (sit-for 0.3)) ; ensure the message is seen for a bit
      (setq +literate-tangle--proc-start-time (float-time)
            +literate-tangle--proc
            (start-process "tangle-config"
                           (get-buffer-create " *tangle config*")
                           "emacs" "--batch" "--eval"
                           (format "(progn \
(require 'ox) \
(require 'ob-tangle) \
(setq org-confirm-babel-evaluate nil \
      org-inhibit-startup t \
      org-mode-hook nil \
      write-file-functions nil \
      before-save-hook nil \
      after-save-hook nil \
      vc-handled-backends nil \
```

```
      org-startup-folded nil \
      org-startup-indented nil) \
(org-babel-tangle-file \"%s\" \"%s\"))"
                                  +literate-config-file
                                  (expand-file-name (concat doom-module-config-file ".el")))))
      (set-process-sentinel +literate-tangle--proc #'+literate-tangle--sentinel)
      (run-at-time nil nil (lambda () (message "Tangling config.org"))) ; ensure shown after a save message
      "Tangling config.org..."))

(defun +literate-tangle--sentinel (process signal)
  (cond
   ((and (eq 'exit (process-status process))
         (= 0 (process-exit-status process)))
    (message "Tangled config.org sucessfully (took %.1fs)"
             (- (float-time) +literate-tangle--proc-start-time))
    (setq +literate-tangle--proc nil))
   ((memq (process-status process) (list 'exit 'signal))
    (pop-to-buffer (get-buffer " *tangle config*"))
    (message "Failed to tangle config.org (after %.1fs)"
             (- (float-time) +literate-tangle--proc-start-time))
    (setq +literate-tangle--proc nil))))

(defun +literate-tangle-check-finished ()
  (when (and (process-live-p +literate-tangle--proc)
             (yes-or-no-p "Config is currently retangling, would you please wait a few seconds?"))
    (switch-to-buffer " *tangle config*")
    (signal 'quit nil)))

(add-hook! 'kill-emacs-hook #'+literate-tangle-check-finished)
```

6.5 IDE

6.5.1 Tabs

Stolen from siren-tab-bar.el.

```
(use-package! tab-bar
  :custom
  (tab-bar-close-button-show nil)
  (tab-bar-format '(tab-bar-format-tabs-groups tab-bar-separator))
  (tab-bar-history-limit 25)
  (tab-bar-new-tab-choice "*scratch*")
  (tab-bar-show 1)
  (tab-bar-tab-group-format-function #'+tab-bar-tab-group-format-default)
  (tab-bar-tab-hints t)
  (tab-bar-tab-name-format-function #'+tab-bar-tab-name-format-default)

  :config
  (+tab-bar-setup)

  :preface
  (defun +tab-bar-setup ()
    (tab-bar-mode)
    (tab-bar-history-mode))

  (defgroup +tab-bar nil
    "Siren specific tweaks to tar-bar-mode."
    :group 'tab-bar)

  (defcustom +tab-bar-echo-tab-list t
    "When t and print list of tabs in echo area when changing tabs."
    :type 'boolean
    :group '+tab-bar)

  (defface +tab-bar-echo-default
    '((t :inherit default))
    "Face for tab names in echo area."
```

```
32      :group '+tab-bar)
33
34    (defface +tab-bar-echo-current
35      '((t :inherit font-lock-keyword-face))
36      "Face for current tab name in echo area."
37      :group '+tab-bar)
38
39    (defface +tab-bar-echo-index
40      '((t :inherit font-lock-comment-face))
41      "Face for index numbers in echo area."
42      :group '+tab-bar)
43
44    (defface +tab-bar-tab
45      `((t :inherit 'tab-bar-tab
46           :foreground ,(face-attribute 'font-lock-keyword-face :foreground nil t)))
47      "Face for active tab in tab-bar."
48      :group '+tab-bar)
49
50    (defface +tab-bar-tab-hint
51      `((t :inherit '+tab-bar-tab
52           :foreground ,(face-attribute 'tab-bar-tab-inactive :foreground nil t)))
53      "Face for active tab hint in tab-bar."
54      :group '+tab-bar)
55
56    (defface +tab-bar-tab-inactive
57      `((t :inherit 'tab-bar-tab-inactive
58           :foreground ,(face-attribute 'font-lock-comment-face :foreground nil t)))
59      "Face for inactive tab in tab-bar."
60      :group '+tab-bar)
61
62    (defface +tab-bar-tab-hint-inactive
63      `((t :inherit '+tab-bar-tab-inactive
64           :foreground ,(face-attribute 'tab-bar-tab-inactive :foreground nil t)))
65      "Face for inactive tab hint in tab-bar."
66      :group '+tab-bar)
67
68    (defun +tab-bar-tab-name-format-default (tab i)
69      (let* ((current-p (eq (car tab) 'current-tab))
70             (tab-face (if current-p
71                           '+tab-bar-tab
72                         '+tab-bar-tab-inactive))
73             (hint-face (if current-p
74                            '+tab-bar-tab-hint
75                          '+tab-bar-tab-hint-inactive)))
76        (concat (propertize "  " 'face tab-face)
77                (if tab-bar-tab-hints (propertize
78                                       (format "%d:" (- i 1)) 'face hint-face))
79                (propertize
80                 (concat
81                  (alist-get 'name tab)
82                  (or (and tab-bar-close-button-show
83                           (not (eq tab-bar-close-button-show
84                                    (if current-p 'non-selected 'selected)))
85                           tab-bar-close-button)
86                      "")
87                  "  ")
88                 'face tab-face))))
89
90    (defun +tab-bar-tab-group-format-default (tab i)
91      (propertize
92       (concat (if tab-bar-tab-hints (format "%d:" (- i 1)) "")
93               (funcall tab-bar-tab-group-function tab))
94       'face 'tab-bar-tab-group-inactive))
95
96    (defun +tab-bar-switch-to-or-create-tab (name)
97      "Switch to or create a tab by NAME."
98      (interactive
99       (let* ((recent-tabs (mapcar (lambda (tab) (alist-get 'name tab))
100                                   (tab-bar--tabs-recent))))
101        (list (completing-read "Switch to tab by name (default recent): "
```

```
                              recent-tabs nil nil nil nil recent-tabs))))
    (let ((tab-names (mapcar (lambda (tab) (alist-get 'name tab))
                             (funcall tab-bar-tabs-function))))
      (if (member name tab-names
           (tab-bar-switch-to-tab name))
          (+tab-bar-new-named-tab name)))
    (tab-bar-select-tab (1+ (or (tab-bar--tab-index-by-name name) 0))))

  (defun +tab-bar-new-named-tab (name)
    "Create a new tab named NAME."
    (interactive "MName for new tab (leave blank for automatic naming): ")
    (tab-new 99999)
    (if (not (string= name ""))
        (tab-rename name)))

  (defun +tab-bar-switch-to-index (&optional arg)
    "Switch to tab with index ARG.
When this command is bound to a numeric key, calling it without
an argument will translate its bound numeric key to the numeric
argument.
ARG counts from 1."
    (interactive "P")
    (unless (integerp arg)
      (let ((key (event-basic-type last-command-event)))
        (setq arg (if (and (characterp key) (>= key ?0) (<= key ?9))
                      (- key ?0)
                    0))))

    (tab-bar-select-tab (1+ arg)))

  (defun +tab-bar-move-tab-left ()
    "Move current tab to the left."
    (interactive)
    (tab-move -1))

  (defun +tab-bar-move-tab-right ()
    "Move current tab to the right."
    (interactive)
    (tab-move 1))

  (defun +tab-bar-echo-tab-list ()
    "Echo list of tabs"
    (interactive)
    (let* ((tabs (funcall tab-bar-tabs-function))
           (current-index (or (tab-bar--current-tab-index tabs) 0))
           (output '())
           (index 0))
      (dolist (tab tabs)
        (add-to-list 'output
                     (concat (propertize (format "%d:" index)
                                         'face '+tab-bar-echo-index)
                             (propertize (alist-get 'name tab)
                                         'face (if (eq index current-index)
                                                   '+tab-bar-echo-current
                                                 '+tab-bar-echo-default)))
                     t)
        (setq index (1+ index)))

      (message "tabs: %s" (string-join output " "))))

  (defun +tab-bar-echo-tab-list-advice (&rest _)
    (when +tab-bar-echo-tab-list
      (+tab-bar-echo-tab-list)))

  (advice-add 'tab-bar-close-tab :after #'+tab-bar-echo-tab-list-advice)
  (advice-add 'tab-bar-move-tab-to :after #'+tab-bar-echo-tab-list-advice)
  (advice-add 'tab-bar-new-tab-to :after #'+tab-bar-echo-tab-list-advice)
  (advice-add 'tab-bar-rename-tab :after #'+tab-bar-echo-tab-list-advice)
  (advice-add 'tab-bar-select-tab :after #'+tab-bar-echo-tab-list-advice)
  (advice-add 'tab-switcher-select :after #'+tab-bar-echo-tab-list-advice)
```

```
  (advice-add 'display-buffer-in-new-tab :after #'+tab-bar-echo-tab-list-advice)
  (advice-add 'tab-bar-change-tab-group :after #'+tab-bar-echo-tab-list-advice))
```

6.5.2 Centaur tabs

Disabled, not working correctly with Emacs Daemon + EmacsClient.

```
(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))
```

6.5.3 Treemacs

```
(unpin! treemacs)
(unpin! lsp-treemacs)
```

```
(after! treemacs
  (require 'dired)

  ;; My custom stuff (from tecosaur's config)
  (setq +treemacs-file-ignore-extensions
        '(;; LaTeX
          "aux" "ptc" "fdb_latexmk" "fls" "synctex.gz" "toc"
          ;; LaTeX - bibliography
          "bbl"
          ;; LaTeX - glossary
          "glg" "glo" "gls" "glsdefs" "ist" "acn" "acr" "alg"
          ;; LaTeX - pgfplots
          "mw"
          ;; LaTeX - pdfx
          "pdfa.xmpi"
          ;; Python
          "pyc"))

  (setq +treemacs-file-ignore-globs
        '(;; LaTeX
          "*/_minted-*"
          ;; AucTeX
          "*/.auctex-auto"
          "*/_region_.log"
          "*/_region_.tex"
          ;; Python
          "*/__pycache__"))

  ;; Reload treemacs theme
  (setq doom-themes-treemacs-enable-variable-pitch nil
        doom-themes-treemacs-theme "doom-colors")
  (doom-themes-treemacs-config)

  (setq treemacs-show-hidden-files nil
        treemacs-hide-dot-git-directory t
        treemacs-width 30)

  (defvar +treemacs-file-ignore-extensions '()
    "File extension which `treemacs-ignore-filter' will ensure are ignored")

  (defvar +treemacs-file-ignore-globs '()
    "Globs which will are transformed to `+treemacs-file-ignore-regexps' which `+treemacs-ignore-filter' will
↪  ensure are ignored")
```

```
(defvar +treemacs-file-ignore-regexps '()
  "RegExps to be tested to ignore files, generated from `+treeemacs-file-ignore-globs'")

(defun +treemacs-file-ignore-generate-regexps ()
  "Generate `+treemacs-file-ignore-regexps' from `+treemacs-file-ignore-globs'"
  (setq +treemacs-file-ignore-regexps (mapcar 'dired-glob-regexp +treemacs-file-ignore-globs)))

(unless (equal +treemacs-file-ignore-globs '())
  (+treemacs-file-ignore-generate-regexps))

(defun +treemacs-ignore-filter (file full-path)
  "Ignore files specified by `+treemacs-file-ignore-extensions', and `+treemacs-file-ignore-regexps'"
  (or (member (file-name-extension file) +treemacs-file-ignore-extensions)
      (let ((ignore-file nil))
        (dolist (regexp +treemacs-file-ignore-regexps ignore-file)
          (setq ignore-file (or ignore-file (if (string-match-p regexp full-path) t nil)))))))

(add-to-list 'treemacs-ignored-file-predicates #'+treemacs-ignore-filter))
```

6.5.4 Projectile

```
;; 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"
        "~/.cache"
        "~/.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)))
```

6.5.5 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 +
```

6.5.6 Eros-eval

This makes the result of evals slightly prettier.

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

6.6 Symbols

6.6.1 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, on the
↪  fly."
  (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))
```

6.6.2 Ligatures

Extra ligatures are good, however, I'd like to see my keywords! Lets disable them in C/C++, Rust and Python modes.

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

6.7 Checkers (spell & grammar)

6.7.1 Install back-end

For `flyspell` + `hunspell`

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

For `spell-fu`

```
sudo pacman -S aspell aspell-en aspell-fr
```

6.7.2 Spell-Fu

Now, `spell-fu` supports multiple languages! Lets add English, French and Arabic. So I can "mélanger les langues sans avoir de problèmes!".

```
(after! spell-fu
  (defun +spell-fu-register-dictionary (lang)
    "Add `LANG` to spell-fu multi-dict, with a personal dictionary."
    ;; Add the dictionary
    (spell-fu-dictionary-add (spell-fu-get-ispell-dictionary lang))
    (let ((personal-dict-file (expand-file-name (format "aspell.%s.pws" lang) doom-private-dir)))
      ;; Create an empty personal dictionary if it doesn't exists
      (unless (file-exists-p personal-dict-file) (write-region "" nil personal-dict-file))
      ;; Add the personal dictionary
      (spell-fu-dictionary-add (spell-fu-get-personal-dictionary (format "%s-personal" lang)
↪  personal-dict-file))))

  (add-hook 'spell-fu-mode-hook
            (lambda ()
              (+spell-fu-register-dictionary "en")
              (+spell-fu-register-dictionary "fr"))))
```

6.7.3 Guess language

```
(package! guess-language
  :recipe (:host github
           :repo "tmalsburg/guess-language.el"))
```

```
(use-package! guess-language
  :config
  (setq guess-language-languages '(en fr ar)
        guess-language-min-paragraph-length 35
        guess-language-langcodes '((en . ("en_US" "English" " " " English"))
                                   (fr . ("francais" "French" " " " Français"))
                                   (ar . ("arabic" "Arabic" " " " Arabic"))))
  ;; :hook (text-mode . guess-language-mode)
  :commands (guess-language
             guess-language-mode
             guess-language-region
             guess-language-mark-lines))
```

6.7.4 Grammarly

Use either eglot-grammarly or lsp-grammarly.

```
(package! grammarly
  :recipe (:host github
           :repo "emacs-grammarly/grammarly"))

;; Install the suitable LSP frontend
(if (featurep! :tools lsp +eglot)
  (package! eglot-grammarly
    :recipe (:host github
             :repo "emacs-grammarly/eglot-grammarly"))
  (package! lsp-grammarly
    :recipe (:host github
             :repo "emacs-grammarly/lsp-grammarly")))
```

```
(use-package! grammarly
  :config
  (grammarly-load-from-authinfo))

(if (featurep! :tools lsp +eglot)
    (use-package! eglot-grammarly
      :commands (+lsp-grammarly-load)
      :init
      (defun +lsp-grammarly-load ()
        "Load Grammarly LSP server."
        (interactive)
        (require 'eglot-grammarly)
        (call-interactively #'eglot)))
  (use-package! lsp-grammarly
    :commands (+lsp-grammarly-load)
    :init
    (defun +lsp-grammarly-load ()
      "Load Grammarly LSP server."
      (require 'lsp-grammarly)
      (lsp-deferred))) ;; or (lsp)
```

6.7.5 Grammalecte

```
(package! flycheck-grammalecte
  :recipe (:host github
           :repo "milouse/flycheck-grammalecte"))
```

```
(use-package! flycheck-grammalecte
  :commands (flycheck-grammalecte-correct-error-at-point
             grammalecte-conjugate-verb
             grammalecte-define
             grammalecte-define-at-point
             grammalecte-find-synonyms
             grammalecte-find-synonyms-at-point)
  :init
  (setq grammalecte-settings-file (expand-file-name "grammalecte/grammalecte-cache.el" doom-etc-dir)
        grammalecte-python-package-directory (expand-file-name "grammalecte/grammalecte" doom-etc-dir))
  (setq flycheck-grammalecte-report-spellcheck t
        flycheck-grammalecte-report-grammar t
        flycheck-grammalecte-report-apos nil
        flycheck-grammalecte-report-esp nil
        flycheck-grammalecte-report-nbsp nil
        flycheck-grammalecte-filters
        '("(?m)^# ?-*-.+$"
          ;; Ignore LaTeX equations (inline and block)
          "\\$.*?\\$"
          "(?s)\\\\begin{equation}.*?\\\\end{equation}"))

  (map! :leader :prefix ("l" . "custom")
```

```
        (:prefix-map ("g" . "grammalecte")
         :desc "Correct error at point"     "p" #'flycheck-grammalecte-correct-error-at-point
         :desc "Conjugate a verb"           "V" #'grammalecte-conjugate-verb
         :desc "Define a word"              "W" #'grammalecte-define
         :desc "Conjugate a verb at point"  "w" #'grammalecte-define-at-point
         :desc "Find synonyms"              "S" #'grammalecte-find-synonyms
         :desc "Find synonyms at point"     "s" #'grammalecte-find-synonyms-at-point))

  :config
  (grammalecte-download-grammalecte)
  (flycheck-grammalecte-setup)
  (add-to-list 'flycheck-grammalecte-enabled-modes 'fountain-mode))
```

6.7.6 Flyspell

```
;; NOTE: Not tangled, using spell-fu instead
(after! (ispell flyspell)
  (setq ispell-dictionary "en_US,fr_FR")

  ;; 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)))
```

```
(after! flyspell
  (setq flyspell-lazy-idle-seconds 2
        flyspell-lazy-window-idle-seconds 5))
```

Lazy flyspell

```
;; NOTE: Not tangled, using spell-fu with multiple dictionaries
(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))))
```

Shortcuts to change dictionary

6.7.7 LanguageTool

Watch the LanguageTool LSP and Eglot implementations. This section defines some shortcuts to check the 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))
```

6.8 System tools

6.8.1 Disk usage

```
(package! disk-usage)
```

```
(use-package! disk-usage
  :commands (disk-usage))
```

6.8.2 Chezmoi

```
(package! chezmoi)
```

```
(use-package! chezmoi
  :when +chezmoi-ok-p
  :commands (chezmoi-write
             chezmoi-magit-status
             chezmoi-diff
             chezmoi-ediff
             chezmoi-find
             chezmoi-write-files
             chezmoi-open-other
             chezmoi-template-buffer-display
             chezmoi-mode)
  :config
  ;; Company integration
  (when (featurep! :completion company)
    (defun +chezmoi--company-backend-h ()
      (require 'chezmoi-company)
      (if chezmoi-mode
          (add-to-list 'company-backends 'chezmoi-company-backend)
        (delete 'chezmoi-company-backend 'company-backends)))

    (add-hook 'chezmoi-mode-hook #'+chezmoi--company-backend-h))

  ;; Integrate with evil mode by toggling template display when entering insert mode.
  (when (featurep! :editor evil)
    (defun +chezmoi--evil-insert-state-enter-h ()
      "Run after evil-insert-state-entry."
      (chezmoi-template-buffer-display nil (point))
      (remove-hook 'after-change-functions #'chezmoi-template--after-change 1))
```

```

    (defun +chezmoi--evil-insert-state-exit-h ()
      "Run after evil-insert-state-exit."
      (chezmoi-template-buffer-display nil)
      (chezmoi-template-buffer-display t)
      (add-hook 'after-change-functions #'chezmoi-template--after-change nil 1))

    (defun +chezmoi--evil-h ()
      (if chezmoi-mode
          (progn
            (add-hook 'evil-insert-state-entry-hook #'+chezmoi--evil-insert-state-enter-h nil 1)
            (add-hook 'evil-insert-state-exit-hook #'+chezmoi--evil-insert-state-exit-h nil 1))
        (progn
          (remove-hook 'evil-insert-state-entry-hook #'+chezmoi--evil-insert-state-enter-h 1)
          (remove-hook 'evil-insert-state-exit-hook #'+chezmoi--evil-insert-state-exit-h 1))))

    (add-hook 'chezmoi-mode-hook #'+chezmoi--evil-h)))
```

6.8.3 Aweshell

```
(package! aweshell
  :recipe (:host github
           :repo "manateelazycat/aweshell"))
```

```
(use-package! aweshell
  :commands (aweshell-new aweshell-dedicated-open))
```

6.8.4 Lemon

```
(package! lemon
  :recipe (:host gitlab
           :repo "ieure/lemon"))
```

```
(use-package! lemon
  :commands (lemon-mode lemon-display)
  :config
  (require 'lemon-cpu)
  (require 'lemon-memory)
  (require 'lemon-network)
  (setq lemon-delay 5
        lemon-refresh-rate 2
        lemon-monitors(list '((lemon-cpufreq-linux :display-opts '(:sparkline (:type gridded)))
                              (lemon-cpu-linux)
                              (lemon-memory-linux)
                              (lemon-linux-network-tx)
                              (lemon-linux-network-rx)))))
```

6.9 Features

6.9.1 Weather

```
;; lisp/wttrin/wttrin.el 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)
```

6.9.2 OpenStreetMap

```
(package! osm)
```

```
(use-package! osm
  :commands (osm-home
             osm-search
             osm-server
             osm-goto
             osm-gpx-show
             osm-bookmark-jump)

  :custom
  ;; Take a look at the customization group `osm' for more options.
  (osm-server 'default) ;; Configure the tile server
  (osm-copyright t)     ;; Display the copyright information

  :init
  (setq osm-tile-directory (expand-file-name "osm" doom-etc-dir))
  ;; Load Org link support
  (with-eval-after-load 'org
    (require 'osm-ol)))
```

6.9.3 Islamic prayer times

```
(package! awqat
  :recipe (:host github
           :repo "zkry/awqat"))
```

```
(use-package! awqat
  :load-path "~/Projects/foss_projects/awqat"
  :commands (awqat-display-prayer-time-mode awqat-times-for-day)
  :config
  ;; Make sure `calendar-latitude' and `calendar-longitude' are set,
  ;; otherwise, set them here.
  (setq awqat-asr-hanafi nil
        awqat-mode-line-format "   ${prayer} (${hours}h${minutes}m) ")
  (awqat-set-preset-french-muslims))
```

6.9.4 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)
```

6.9.5 Zotero Zotxt

```
(package! zotxt)
```

```
(use-package! zotxt
  :when +zotero-ok-p
  :commands org-zotxt-mode)
```

6.9.6 CRDT

Collaborative editing for geeks! `crdt.el` adds support for *Conflict-free Replicated Data Type*.

```
(package! crdt)
```

```
(use-package! crdt
  :commands (crdt-share-buffer
             crdt-connect
             crdt-visualize-author-mode
             crdt-org-sync-overlay-mode))
```

6.9.7 The Silver Searcher

An Emacs front-end to *The Silver Searcher*, first we need to install `ag` using `sudo pacman -S the_silver_searcher`.

```
(package! ag)
```

```
(use-package! ag
  :when +ag-ok-p
  :commands (ag
             ag-files
             ag-regexp
             ag-project
             ag-project-files
             ag-project-regexp))
```

6.9.8 Emacs Application Framework

EAF is presented as: *A free/libre and open-source extensible framework that revolutionizes the graphical capabilities of Emacs.* Or the key to ultimately *Live in Emacs*.

First, install EAF as specified in the project's readme. To update EAF, we need to run `git pull ; ./install-eaf.py` in `lisp/emacs-application-framework` and `(M-x eaf-install-and-update)` in Emacs. This updates EAF, applications and their dependencies.

```
(use-package! eaf
  ;; EAF doesn't work with LUCID build, however, I found LUCID more stable for
  ;; Emacs daemon + emacsclient usage. So, this section will not be used for LUCID builds.
  :when (and +eaf-ok-p (not (string-search "LUCID" system-configuration-features)))
  :load-path "lisp/emacs-application-framework"
  :commands (eaf-open eaf-open-browser eaf-open-jupyter eaf-open-mail-as-html)
  :init
  (defvar +eaf-enabled-apps
    '(org mail browser mindmap jupyter org-previewer markdown-previewer))
```

```
;; mindmap file-manager file-browser
;; file-sender music-player video-player
;; git image-viewer

:config
;; Generic
(setq eaf-start-python-process-when-require t
      eaf-kill-process-after-last-buffer-closed t
      eaf-fullscreen-p nil)

;; Debug
(setq eaf-enable-debug nil)

;; Web engine
(setq eaf-webengine-font-family "FantasqueSansMono Nerd Font Mono"
      eaf-webengine-fixed-font-family "FantasqueSansMono Nerd Font Mono"
      eaf-webengine-serif-font-family "FantasqueSansMono Nerd Font Mono"
      eaf-webengine-font-size 14
      eaf-webengine-fixed-font-size 14
      eaf-webengine-download-path "~/Downloads"
      eaf-webengine-enable-plugin t
      eaf-webengine-enable-javascript t
      eaf-webengine-enable-javascript-access-clipboard t
      eaf-webengine-enable-scrollbar t
      eaf-webengine-default-zoom 1.25
      eaf-webengine-scroll-step 200)

(when (display-graphic-p)
  (require 'eaf-all-the-icons))

;; Browser settings
(when (member 'browser +eaf-enabled-apps)
  (setq eaf-browser-continue-where-left-off t
        eaf-browser-dark-mode "follow"
        eaf-browser-enable-adblocker t
        eaf-browser-enable-autofill nil
        eaf-browser-remember-history t
        eaf-browser-ignore-history-list '("google.com/search" "file://")
        eaf-browser-text-selection-color "auto"
        eaf-browser-translate-language "fr"
        eaf-browser-blank-page-url "https://www.duckduckgo.com"
        eaf-browser-chrome-history-file "~/.config/google-chrome/Default/History"
        eaf-browser-default-search-engine "duckduckgo"
        eaf-browser-continue-where-left-off nil)

  (require 'eaf-browser)

  ;; Make EAF Browser my default browser
  (setq browse-url-browser-function #'eaf-open-browser)
  (defalias 'browse-web #'eaf-open-browser))


;; File manager settings
(when (member 'file-manager +eaf-enabled-apps)
  (setq eaf-file-manager-show-preview nil
        eaf-find-alternate-file-in-dired t
        eaf-file-manager-show-hidden-file t
        eaf-file-manager-show-icon t)
  (require 'eaf-file-manager))

;; File Browser
(when (member 'file-browser +eaf-enabled-apps)
  (require 'eaf-file-browser))

;; PDF Viewer settings
(when (member 'pdf-viewer +eaf-enabled-apps)
  (setq eaf-pdf-dark-mode "follow"
        eaf-pdf-show-progress-on-page nil
        eaf-pdf-dark-exclude-image t
        eaf-pdf-notify-file-changed t)
```

```
80      (require 'eaf-pdf-viewer)
81
82      (after! org
83        ;; Use EAF PDF Viewer in Org
84        (defun +eaf-org-open-file-fn (file &optional link)
85          "An wrapper function on `eaf-open'."
86          (eaf-open file))
87
88        ;; use `emacs-application-framework' to open PDF file: link
89        (add-to-list 'org-file-apps '("\\.pdf\\'" . +eaf-org-open-file-fn)))
90
91      (after! latex
92        ;; Link EAF with the LaTeX compiler in emacs. When a .tex file is open,
93        ;; the Command>Compile and view (C-c C-a) option will compile the .tex
94        ;; file into a .pdf file and display it using EAF. Double clicking on the
95        ;; PDF side jumps to editing the clicked section.
96        (add-to-list 'TeX-command-list '("XeLaTeX" "%`xelatex --synctex=1%(mode)%' %t" TeX-run-TeX nil t))
97        (add-to-list 'TeX-view-program-list '("eaf" eaf-pdf-synctex-forward-view))
98        (add-to-list 'TeX-view-program-selection '(output-pdf "eaf"))))
99
100   ;; Org
101   (when (member 'rss-reader +eaf-enabled-apps)
102     (setq eaf-rss-reader-split-horizontally nil
103           eaf-rss-reader-web-page-other-window t)
104     (require 'eaf-org))
105
106   ;; Org
107   (when (member 'org +eaf-enabled-apps)
108     (require 'eaf-org))
109
110   ;; Mail
111   (when (member 'mail +eaf-enabled-apps)
112     (require 'eaf-mail))
113
114   ;; Org Previewer
115   (when (member 'org-previewer +eaf-enabled-apps)
116     (setq eaf-org-dark-mode "follow")
117     (require 'eaf-org-previewer))
118
119   ;; Markdown Previewer
120   (when (member 'markdown-previewer +eaf-enabled-apps)
121     (setq eaf-markdown-dark-mode "follow")
122     (require 'eaf-markdown-previewer))
123
124   ;; Jupyter
125   (when (member 'jupyter +eaf-enabled-apps)
126     (setq eaf-jupyter-dark-mode "follow"
127           eaf-jupyter-font-family "JuliaMono"
128           eaf-jupyter-font-size 13)
129     (require 'eaf-jupyter))
130
131   ;; Mindmap
132   (when (member 'mindmap +eaf-enabled-apps)
133     (setq eaf-mindmap-dark-mode "follow"
134           eaf-mindmap-save-path "~/Dropbox/Mindmap")
135     (require 'eaf-mindmap))
136
137   ;; File Sender
138   (when (member 'file-sender +eaf-enabled-apps)
139     (require 'eaf-file-sender))
140
141   ;; Music Player
142   (when (member 'music-player +eaf-enabled-apps)
143     (require 'eaf-music-player))
144
145   ;; Video Player
146   (when (member 'video-player +eaf-enabled-apps)
147     (require 'eaf-video-player))
148
149   ;; Image Viewer
```

```
  (when (member 'image-viewer +eaf-enabled-apps)
    (require 'eaf-image-viewer))

  ;; Git
  (when (member 'git +eaf-enabled-apps)
    (require 'eaf-git))

  ;; EVIL keybindings for Doom
  (after! evil
    (require 'eaf-evil)
    (define-key key-translation-map (kbd "SPC")
      (lambda (prompt)
        (if (derived-mode-p 'eaf-mode)
            (pcase eaf--buffer-app-name
              ("browser" (if (eaf-call-sync "execute_function" eaf--buffer-id "is_focus")
                             (kbd "SPC")
                           (kbd eaf-evil-leader-key)))
              ("pdf-viewer" (kbd eaf-evil-leader-key))
              ("image-viewer" (kbd eaf-evil-leader-key))
              ("music-player" (kbd eaf-evil-leader-key))
              ("video-player" (kbd eaf-evil-leader-key))
              (_  (kbd "SPC")))
          (kbd "SPC"))))))
```

6.9.9 Bitwarden

```
(package! bitwarden
  :recipe (:host github
           :repo "seanfarley/emacs-bitwarden"))
```

```
(use-package! bitwarden
  ;;:config
  ;;(bitwarden-auth-source-enable)
  :when +bitwarden-ok-p
  :init
  (setq bitwarden-automatic-unlock
        (lambda ()
          (require 'auth-source)
          (if-let* ((matches (auth-source-search :host "bitwarden.com" :max 1))
                    (entry (nth 0 matches))
                    (email (plist-get entry :user))
                    (pass (plist-get entry :secret)))
              (progn
                (setq bitwarden-user email)
                (if (functionp pass) (funcall pass) pass))
            "")))))
```

6.9.10 PDF tools

```
(unpin! pdf-tools)
```

Dark mode The `pdf-tools` package supports dark mode (midnight), I use Emacs often to write and read PDF documents, so lets make it dark by default, this can be toggled using the `m z`.

```
(after! pdf-tools
  (add-hook! 'pdf-view-mode-hook (pdf-view-midnight-minor-mode 1)))
```

Better PDFs in mode line 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)))
```

6.10 Fun

6.10.1 Speed Type

A game to practice speed typing in Emacs.

```
(package! speed-type)
```

```
(use-package! speed-type
  :commands (speed-type-text))
```

6.10.2 2048 Game

```
(package! 2048-game)
```

```
(use-package! 2048-game
  :commands (2048-game))
```

6.10.3 Snow

Let it snow in Emacs!

```
(package! snow)
```

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

6.10.4 `xkcd`

```
(package! xkcd
  :recipe (:host github
           :repo "vibhavp/emacs-xkcd"))
```

```
(use-package! xkcd
  :commands (xkcd-get xkcd)
  :config
  (setq xkcd-cache-dir (expand-file-name "xkcd/" doom-cache-dir)
        xkcd-cache-latest (expand-file-name "xkcd/latest" doom-cache-dir)))
```

7 Applications

7.1 Calendar

```
(setq calendar-latitude 48.7
      calendar-longitude 2.17
      calendar-location-name "Orsay, FR"
      calendar-time-display-form '(24-hours ":" minutes
                                            (if time-zone " (") time-zone (if time-zone ")")))
```

7.2 Dirvish :heart: Dired

DIRVISH is a minimalist yet versatile file manager based on Dired. Faster than `ranger.el`.

```
(package! dirvish)
```

```
(use-package! dirvish
  :after dired
  :init (dirvish-override-dired-mode)
  :hook (dired-mode . dired-omit-mode)
  :config
  (setq dirvish-cache-dir (concat doom-cache-dir "dirvish/")
        dirvish-hide-details nil
        dired-omit-files (concat dired-omit-files "\\|^\\..*$"))
  (map! :map dirvish-mode-map
        :n "b" #'dirvish-goto-bookmark
        :n "z" #'dirvish-show-history
        :n "f" #'dirvish-file-info-menu
        :n "F" #'dirvish-toggle-fullscreen
        :n "l" #'dired-find-file
        :n "h" #'dired-up-directory
        :localleader
        "h" #'dired-omit-mode))
```

7.3 e-Books nov

```
(package! 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-modeline-info)))

  (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--major-mode))))))))))
                  (:eval (doom-modeline-segment--major-mode)))))

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

7.4 News feed elfeed

Set RSS news feeds

```
(setq elfeed-feeds
      '("https://this-week-in-rust.org/rss.xml"
        "https://www.omgubuntu.co.uk/feed"
        "https://itsfoss.com/feed"
        "https://linuxhandbook.com/feed"
        "https://spectrum.ieee.org/rss/robotics/fulltext"
        "https://spectrum.ieee.org/rss/aerospace/fulltext"
        "https://spectrum.ieee.org/rss/computing/fulltext"
        "https://spectrum.ieee.org/rss/blog/automaton/fulltext"
        "https://developers.redhat.com/blog/feed"
        "https://lwn.net/headlines/rss"))
```

7.5 VPN configuration

7.5.1 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_FILE})
```

7.5.2 Launch NetExtender session from Emacs

```
(when +netextender-ok-p
  (defvar +netextender-process-name "netextender")
  (defvar +netextender-buffer-name "*netextender*")
  (defvar +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"))))
```

```
15
16  (defun +netextender-kill ()
17    "Kill the created NetExtender VPN session"
18    (interactive)
19    (when (get-process +netextender-process-name)
20      (if (kill-buffer +netextender-buffer-name)
21          (message "Killed NetExtender VPN session")
22        (message "Cannot kill NetExtender")))))
```

7.6 Email mu4e

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

7.6.1 mbsync

You will need to:

- Install mu and isync (sudo pacman -S mu isync)
- Set up a proper configuration file for your accounts at ~/.mbsyncrc
- Run mu init --maildir=~/Maildir --my-address=user@host1 --my-address=user@host2
- Run mbsync -c ~/.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 MY_USER password MY_PASSWORD
- Encrypt the ~/.authinfo file using GPG gpg -c ~/.authinfo and delete the original unencrypted file.

I use a mbsyncrc file for multi-accounts, with some hacks for Gmail accounts (to rename the [Gmail]/... folders). Here is an explained configuration example.

```
1  # mbsync config file
2  # GLOBAL OPTIONS
3  BufferLimit 50mb              # Global option:   Default buffer size is 10M, too small for modern machines.
4  Sync All                      # Channels global: Sync everything "Pull Push New ReNew Delete Flags" (default
   ↪  option)
5  Create Both                   # Channels global: Automatically create missing mailboxes on both sides
6  Expunge Both                  # Channels global: Delete messages marked for deletion on both sides
7  CopyArrivalDate yes           # Channels global: Propagate arrival time with the messages
8
9  # SECTION (IMAP4 Accounts)
10 IMAPAccount work              # IMAP Account name
11 Host mail.host.ccc            # The host to connect to
12 User user@host.ccc            # Login user name
13 SSLVersions TLSv1.2 TLSv1.1   # Supported SSL versions
14 # Extract password from encrypted ~/.authinfo.gpg
15 # File format: "machine <SERVER> login <LOGIN> port <PORT> password <PASSWORD>"
16 # This uses sed to extract <PASSWORD> from line matching the account's <SERVER>
17 PassCmd "echo $(gpg --no-tty -qd ~/.authinfo.gpg 2> /dev/null | sed -n 's,^machine smtp\\.host\\.ccc .*password
   ↪  \\(..*\\)$,\\1,p')"
18 AuthMechs *                   # Authentication mechanisms
19 SSLType IMAPS                 # Protocol (STARTTLS/IMAPS)
20 CertificateFile /etc/ssl/certs/ca-certificates.crt
21 # END OF SECTION
22 # IMPORTANT NOTE: you need to keep the blank line after each section
23
24 # SECTION (IMAP Stores)
25 IMAPStore work-remote         # Remote storage name
26 Account work                  # Associated account
27 # END OF SECTION
28
29 # SECTION (Maildir Stores)
30 MaildirStore work-local       # Local storage (create directories with mkdir -p ~/Maildir/<ACCOUNT-NAME>)
```

```
Path ~/Maildir/work/          # The local store path
Inbox ~/Maildir/work/Inbox    # Location of the INBOX
SubFolders Verbatim           # Download all sub-folders
# END OF SECTION

# Connections specify links between remote and local folders
# they are specified using patterns, which match remote mail
# folders. Some commonly used patters include:
#
# - "*" to match everything
# - "!DIR" to exclude "DIR"
# - "DIR" to match DIR
#
# SECTION (Channels)
Channel work                  # Channel name
Far :work-remote:             # Connect remote store
Near :work-local:             # to the local one
Patterns "INBOX" "Drafts" "Sent" "Archives/*" "Spam" "Trash"
SyncState *                   # Save state in near side mailbox file ".mbsyncstate"
# END OF SECTION

# ============================================================================

IMAPAccount gmail
Host imap.gmail.com
User user@gmail.com
PassCmd "echo $(gpg --no-tty -qd ~/.authinfo.gpg 2> /dev/null | sed -n 's,^machine smtp\\.googlemail\\.com
↪  .*password \\(..*\\)$,\\1,p')"
AuthMechs LOGIN
SSLType IMAPS
CertificateFile /etc/ssl/certs/ca-certificates.crt

IMAPStore gmail-remote
Account gmail

MaildirStore gmail-local
Path ~/Maildir/gmail/
Inbox ~/Maildir/gmail/Inbox

# For Gmail, I like to make multiple channels, one for each remote directory
# this is a trick to rename remote "[Gmail]/mailbox" to "mailbox"
Channel gmail-inbox
Far :gmail-remote:
Near :gmail-local:
Patterns "INBOX"
SyncState *

Channel gmail-trash
Far :gmail-remote:"[Gmail]/Trash"
Near :gmail-local:"Trash"
SyncState *

Channel gmail-drafts
Far :gmail-remote:"[Gmail]/Drafts"
Near :gmail-local:"Drafts"
SyncState *

Channel gmail-sent
Far :gmail-remote:"[Gmail]/Sent Mail"
Near :gmail-local:"Sent Mail"
SyncState *

Channel gmail-all
Far :gmail-remote:"[Gmail]/All Mail"
Near :gmail-local:"All Mail"
SyncState *

Channel gmail-starred
Far :gmail-remote:"[Gmail]/Starred"
Near :gmail-local:"Starred"
```

```
SyncState *

Channel gmail-spam
Far :gmail-remote:"[Gmail]/Spam"
Near :gmail-local:"Spam"
SyncState *

# GROUPS PUT TOGETHER CHANNELS, SO THAT WE CAN INVOKE
# MBSYNC ON A GROUP TO SYNC ALL CHANNELS
#
# FOR INSTANCE: "mbsync gmail" GETS MAIL FROM
# "gmail-inbox", "gmail-sent", and "gmail-trash"
#
# SECTION (Groups)
Group gmail
Channel gmail-inbox
Channel gmail-sent
Channel gmail-trash
Channel gmail-drafts
Channel gmail-all
Channel gmail-starred
Channel gmail-spam
# END OF SECTION
```

7.6.2 mu4e

Add `mu4e` to path if it exists on the file system.

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

I configure my email accounts in a private file in `lisp/private/+mu4e-accounts.el`, which will be loaded after this common part:

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

  ;; Common parameters
  (setq mu4e-update-interval (* 3 60) ;; Every 3 min
        +mu4e-backend 'mbsync
        mu4e-index-update-error-warning nil ;; Do not show warning after update
        ;; 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 ;; Not needed, it is 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:1d..today" :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)

  ;; I like to always BCC myself
  (defun +bbc-me ()
    "Add my email to BCC."
    (save-excursion (message-add-header (format "Bcc: %s\n" user-mail-address))))

  (add-hook 'mu4e-compose-mode-hook '+bbc-me)

  ;; I constantly get a non systematic error after sending a mail.
  ;; >> Error (message-sent-hook): Error running hook "undo" because:
  ;; >> (error Unrecognized entry in undo list undo-tree-canary)
  ;; It is triggered by the 'message-sent-hook', so lets remove the 'undo'
  ;; command from the hook, we can do this before sending the message via
  ;; the 'message-send-hook'.
  (add-hook 'message-send-hook ;; Befor sending the message
            ;; Remove the problematic 'undo' hook.
            (lambda () (remove-hook 'message-sent-hook 'undo t)))

  ;; 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" ;; Account label

 ;; Mu4e folders
 '((mu4e-sent-folder             . "/work-dir/Sent")
   (mu4e-drafts-folder           . "/work-dir/Drafts")
   (mu4e-trash-folder            . "/work-dir/Trash")
   (mu4e-refile-folder           . "/work-dir/Archive")

   ;; Org-msg template (signature and greeting)
   (org-msg-greeting-fmt         . "Hello%s,")
   (org-msg-signature            . "

Regards,

#+begin_signature
-----
*Abdelhak BOUGOUFFA* \\\\
/PhD. Candidate in Robotics | R&D Engineer/ \\\\
/Paris-Saclay University - SATIE/MOSS | ez-Wheel/ \\\\
#+end_signature")

   ;; 'smtpmail' options
   (smtpmail-smtp-user           . "username@server.com")
   (smtpmail-smtp-server         . "smtps.server.com")
   (smtpmail-stream-type         . ssl)
   (smtpmail-smtp-service        . 465)

   ;; By default, `smtpmail' will try to send mails without authentication, and if rejected,
```

```
    ;; it tries to send credentials. This behavior broke my configuration. So I set this
    ;; variable to tell 'smtpmail' to require authentication for our server (using a regex).
    (smtpmail-servers-requiring-authorization . "smtps\\.server\\.com"))

 t) ;; Use as default/fallback account

;; Set another account
(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")
   (org-msg-greeting-fmt         . "Hello%s,")
   (org-msg-signature            . "-- SIGNATURE")
   (smtpmail-smtp-user           . "username@gmail.com")
   (smtpmail-smtp-server         . "smtp.googlemail.com")
   (smtpmail-stream-type         . starttls)
   (smtpmail-smtp-service        . 587)
   ...))

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

7.7 Multimedia

I like to use a MPD powered EMMS, so when I restart Emacs I do not lost my music.

7.7.1 MPD, MPC, and MPV

```
;; Not sure if it is required!
(after! mpc
  (setq mpc-host "localhost:6600"))
```

I like to launch the music daemon `mpd` using Systemd, lets define some commands in Emacs to start/kill the server:

```
(defun +mpd-daemon-start ()
  "Start MPD, connects to it and syncs the metadata cache."
  (interactive)
  (let ((mpd-daemon-running-p (+mpd-daemon-running-p)))
    (unless mpd-daemon-running-p
      ;; Start the daemon if it is not already running.
      (setq mpd-daemon-running-p (zerop (call-process "systemctl" nil nil nil "--user" "start" "mpd.service"))))
    (cond ((+mpd-daemon-running-p)
           (+mpd-mpc-update)
           (emms-player-mpd-connect)
           (emms-cache-set-from-mpd-all)
           (message "Connected to MPD!"))
          (t
           (warn "An error occured when trying to start Systemd mpd.service.")))))

(defun +mpd-daemon-stop ()
  "Stops playback and kill the MPD daemon."
  (interactive)
  (emms-stop)
  (call-process "systemctl" nil nil nil "--user" "stop" "mpd.service")
  (message "MPD stopped!"))

(defun +mpd-daemon-running-p ()
  "Check if the MPD service is running."
  (zerop (call-process "systemctl" nil nil nil "--user" "is-active" "--quiet" "mpd.service")))

(defun +mpd-mpc-update ()
```

```
  "Updates the MPD database synchronously."
  (interactive)
  (if (zerop (call-process "mpc" nil nil nil "update"))
      (message "MPD database updated!")
    (warn "An error occured when trying to update MPD database.")))
```

7.7.2 EMMS

Now, we configure EMMS to use MPD if it is present; otherwise, it uses whatever default backend EMMS is using.

```
(after! emms
  ;; EMMS basic configuration
  (require 'emms-setup)

  (when +mpd-ok-p
    (require 'emms-player-mpd))

  (emms-all)
  (emms-default-players)

  (setq emms-source-file-default-directory "~/Music/"
        ;; Load cover images
        emms-browser-covers 'emms-browser-cache-thumbnail-async
        emms-seek-seconds 5)

  (if +mpd-ok-p
      ;; If using MPD as backend
      (setq emms-player-list '(emms-player-mpd)
            emms-info-functions '(emms-info-mpd)
            emms-player-mpd-server-name "localhost"
            emms-player-mpd-server-port "6600"
            emms-player-mpd-music-directory (expand-file-name "~/Music"))
    ;; Use whatever backend EMMS is using by default (VLC in my machine)
    (setq emms-info-functions '(emms-info-tinytag))) ;; use Tinytag, or '(emms-info-exiftool) for Exiftool

  ;; Keyboard shortcuts
  (global-set-key (kbd "<XF86AudioPrev>")  'emms-previous)
  (global-set-key (kbd "<XF86AudioNext>")  'emms-next)
  (global-set-key (kbd "<XF86AudioPlay>")  'emms-pause)
  (global-set-key (kbd "<XF86AudioPause>") 'emms-pause)
  (global-set-key (kbd "<XF86AudioStop>")  'emms-stop)

  ;; Try to start MPD or connect to it if it is already started.
  (when +mpd-ok-p
    (emms-player-set emms-player-mpd 'regex
                     (emms-player-simple-regexp
                      "m3u" "ogg" "flac" "mp3" "wav" "mod" "au" "aiff"))
    (add-hook 'emms-playlist-cleared-hook 'emms-player-mpd-clear)
    (+mpd-daemon-start))

  ;; Activate EMMS in mode line
  (emms-mode-line 1)

  ;; More descriptive track lines in playlists
  ;; From: https://www.emacswiki.org/emacs/EMMS#h5o-15
  (defun +better-emms-track-description (track)
    "Return a somewhat nice track description."
    (let ((artist (emms-track-get track 'info-artist))
          (album (emms-track-get track 'info-album))
          (tracknumber (emms-track-get track 'info-tracknumber))
          (title (emms-track-get track 'info-title)))
      (cond
       ((or artist title)
        (concat
         (if (> (length artist) 0) artist "Unknown artist") ": "
         (if (> (length album) 0) album "Unknown album") " - "
```

```
        (if (> (length tracknumber) 0) (format "%02d. " (string-to-number tracknumber)) "")
        (if (> (length title) 0) title "Unknown title")))
      (t
       (emms-track-simple-description track)))))

(setq emms-track-description-function '+better-emms-track-description)

;; Manage notifications, inspired by:
;; https://www.emacswiki.org/emacs/EMMS#h5o-9
;; https://www.emacswiki.org/emacs/EMMS#h5o-11
(cond
 ;; Choose D-Bus to disseminate messages, if available.
 ((and (require 'dbus nil t) (dbus-ping :session "org.freedesktop.Notifications"))
  (setq +emms-notifier-function '+notify-via-freedesktop-notifications)
  (require 'notifications))
 ;; Try to make use of KNotify if D-Bus isn't present.
 ((and window-system (executable-find "kdialog"))
  (setq +emms-notifier-function '+notify-via-kdialog))
 ;; Use the message system otherwise
 (t
  (setq +emms-notifier-function '+notify-via-messages)))

(setq +emms-notification-icon "/usr/share/icons/Papirus/64x64/apps/enjoy-music-player.svg")

(defun +notify-via-kdialog (title msg icon)
  "Send notification with TITLE, MSG, and ICON via `KDialog'."
  (call-process "kdialog"
                nil nil nil
                "--title" title
                "--passivepopup" msg "5"
                "--icon" icon))

(defun +notify-via-freedesktop-notifications (title msg icon)
  "Send notification with TITLE, MSG, and ICON via `D-Bus'."
  (notifications-notify
   :title title
   :body msg
   :app-icon icon
   :urgency 'low))

(defun +notify-via-messages (title msg icon)
  "Send notification with TITLE, MSG to message. ICON is ignored."
  (message "%s %s" title msg))

(add-hook 'emms-player-started-hook
          (lambda () (funcall +emms-notifier-function
                              "EMMS is now playing:"
                              (emms-track-description (emms-playlist-current-selected-track))
                              +emms-notification-icon)))

;; MPV and Youtube integration
(when +mpv-ok-p
  (add-to-list 'emms-player-list 'emms-player-mpv t)
  (emms-player-set
   emms-player-mpv
   'regex
   (rx (or (: "https://" (* nonl) "youtube.com" (* nonl))
           (+ (? (or "https://" "http://"))
              (* nonl)
              (regexp (eval (emms-player-simple-regexp
                             "mp4" "mov" "wmv" "webm" "flv" "avi" "mkv")))))))

  (setq +youtube-dl-quality-list
        '("bestvideo[height<=720]+bestaudio/best[height<=720]"
          "bestvideo[height<=480]+bestaudio/best[height<=480]"
          "bestvideo[height<=1080]+bestaudio/best[height<=1080]"))

  (setq +default-emms-player-mpv-parameters
        '("--quiet" "--really-quiet" "--no-audio-display"))

```

```
    (defun +set-emms-mpd-youtube-quality (quality)
      (interactive "P")
      (unless quality
        (setq quality (completing-read "Quality: " +youtube-dl-quality-list nil t)))
      (setq emms-player-mpv-parameters
            `(,@+default-emms-player-mpv-parameters ,(format "--ytdl-format=%s" quality))))

    (+set-emms-mpd-youtube-quality (car +youtube-dl-quality-list))

    (defun +get-youtube-url (link)
      (let ((watch-id (cadr
                       (assoc "watch?v"
                              (url-parse-query-string
                               (substring
                                (url-filename
                                 (url-generic-parse-url link))
                                1))))))
        (concat "https://www.youtube.com/watch?v=" watch-id)))))

;; Example, to be used in an EMMS Playlist
;; (let ((track (emms-track 'url (+get-youtube-url
↪  "https://www.youtube.com/watch?v=Wh-7Kg-jVLg&list=PLBsIgVubrncChqmejIOyA-Xp_dcywQQln"))))
;;   (emms-track-set track 'info-title "Vid")
;;   (emms-playlist-insert-track track))
```

7.7.3 Elfeed :heart: MPV

https://sqrtminusone.xyz/posts/2021-09-07-emms/

Install `yt-rss` from Git:

```
YT_RSS_DIR=~/.local/share/yt-rss
git clone https://github.com/SqrtMinusOne/yt-rss.git $YT_RSS_DIR
cd $TY_RSS_DIR
pip install -r requirements.txt
gunicorn main:app
```

```
(after! (elfeed emms)
  (when +mpv-ok-p
    ;; Integration with Elfeed
    (define-emms-source elfeed (entry)
      (let ((track (emms-track
                    'url (+get-youtube-url (elfeed-entry-link entry)))))
        (emms-track-set track 'info-title (elfeed-entry-title entry))
        (emms-playlist-insert-track track)))

    (defun +elfeed-add-emms-youtube ()
      (interactive)
      (emms-add-elfeed elfeed-show-entry)
      (elfeed-tag elfeed-show-entry 'watched)
      (elfeed-show-refresh))

    (defun +elfeed-search-filter-source (entry)
      "Filter elfeed search buffer by the feed under cursor."
      (interactive (list (elfeed-search-selected :ignore-region)))
      (when (elfeed-entry-p entry)
        (elfeed-search-set-filter
         (concat
          "@6-months-ago "
          "+unread "
          "="
          (replace-regexp-in-string
           (rx "?" (* not-newline) eos)
           ""
           (elfeed-feed-url (elfeed-entry-feed entry))))))))))
```

7.7.4 Keybindings

Lastly, lets define keybindings for these commands, under `<leader> l m`.

```
(map! :leader :prefix ("l" . "custom")
      (:when (featurep! :app emms)
       :prefix-map ("m" . "media")
       :desc "Playlist go"                "g" #'emms-playlist-mode-go
       :desc "Add playlist"               "D" #'emms-add-playlist
       :desc "Toggle random playlist"     "r" #'emms-toggle-random-playlist
       :desc "Add directory"              "d" #'emms-add-directory
       :desc "Add file"                   "f" #'emms-add-file
       :desc "Smart browse"               "b" #'emms-smart-browse
       :desc "Play/Pause"                 "p" #'emms-pause
       :desc "Start"                      "S" #'emms-start
       :desc "Start"                      "S" #'emms-start
       :desc "Stop"                       "s" #'emms-stop))

```

Then we add MPD related keybindings if MPD is used.

```
(map! :leader
      :prefix ("l m")
      (:when (and (featurep! :app emms) +mpd-ok-p)
       :prefix-map ("m" . "mpd/mpc")
       :desc "Start daemon"               "s" #'+mpd-daemon-start
       :desc "Stop daemon"                "k" #'+mpd-daemon-stop
       :desc "EMMS player (MPD update)"   "R" #'emms-player-mpd-update-all-reset-cache
       :desc "Update database"            "u" #'+mpd-mpc-update))
```

7.7.5 Cycle song information in mode line

I found a useful package named **emms-mode-line-cycle** which permits to do this; however, it has not been updated since a while, it uses some obsolete functions to draw icon in mode line, so I forked it, got rid of the problematic parts, and added some minor stuff.

```
(package! emms-mode-line-cycle
  :recipe (:host github
           :repo "abougouffa/emms-mode-line-cycle"))
```

```
(use-package! emms-mode-line-cycle
  :after emms
  :config
  (setq emms-mode-line-cycle-max-width 15
        emms-mode-line-cycle-additional-space-num 4
        emms-mode-line-cycle-any-width-p nil
        emms-mode-line-cycle-velocity 4)

  ;; Some music files do not have metadata, by default, the track title
  ;; will be the full file path, so, if I detect what seems to be an absolute
  ;; path, I trim the directory part and get only the file name.
  (setq emms-mode-line-cycle-current-title-function
        (lambda ()
          (let ((name (emms-track-description (emms-playlist-current-selected-track))))
            (if (file-name-absolute-p name) (file-name-base name) name))))

  ;; Mode line formatting settings
  ;; This format complements the 'emms-mode-line-format' one.
  (setq emms-mode-line-format "  %s "  ;;
        ;; To hide the playing time without stopping the cycling.
        emms-playing-time-display-format "")

  (defun +emms-mode-line-toggle-format-hook ()
```

```
    "Toggle the 'emms-mode-line-fotmat' string, when playing or paused."
    (setq emms-mode-line-format (concat "  " (if emms-player-paused-p " " " ") " %s "))
    ;; Force a sync to get the right song name over MPD in mode line
    (when +mpd-ok-p (emms-player-mpd-sync-from-mpd))
    ;; Trigger a forced update of mode line (useful when pausing)
    (emms-mode-line-alter-mode-line))

      ;; Hook the function to the 'emms-player-paused-hook'
  (add-hook 'emms-player-paused-hook '+emms-mode-line-toggle-format-hook)

  (emms-mode-line-cycle 1))
```

7.8 Maxima

The Maxima CAS comes bundled with three Emacs modes: `maxima`, `imaxima` and `emaxima`; installed by default in `"/usr/share/emacs/site-lisp/maxima"`.

7.8.1 Maxima

The emacsmirror/maxima seems more up-to-date, and supports completion via Company, so lets install it from Github. Note that, normally, we don't need to specify a recipe; however, installing it directly seems to not install `company-maxima.el` and `poly-maxima.el`.

```
(package! maxima
  :recipe (:host github
           :repo "emacsmirror/maxima"
           :files (:defaults
                   "keywords"
                   "company-maxima.el"
                   "poly-maxima.el")))
```

```
(use-package! maxima
  :when +maxima-ok-p
  :commands (maxima-mode maxima-inferior-mode maxima)
  :init
  (require 'straight) ;; to use `straight-build-dir' and `straight-base-dir'
  (setq maxima-font-lock-keywords-directory ;; a workaround to undo the straight workaround!
        (expand-file-name (format "straight/%s/maxima/keywords" straight-build-dir) straight-base-dir))

  ;; The `maxima-hook-function' setup `company-maxima'.
  (add-hook 'maxima-mode-hook #'maxima-hook-function)
  (add-hook 'maxima-inferior-mode-hook #'maxima-hook-function)
  (add-to-list 'auto-mode-alist '("\\.ma[cx]\\'" . maxima-mode)))
```

7.8.2 IMaxima

For the `imaxima` (Maxima with image support), the emacsattic/imaxima seems outdated compared to the `imaxima` package of the official Maxima distribution, so lets install `imaxima` from the source code of Maxima, hosted on Sourceforge git.code.sf.net/p/maxima/code. The package files are stored in the repository's sub-directory `interfaces/emacs/imaxima`.

```
;; Use the `imaxima' package bundled with the official Maxima distribution.
(package! imaxima
  :recipe (:host nil ;; Unsupported host, we will specify the complete repo link
           :repo "https://git.code.sf.net/p/maxima/code"
           :files ("interfaces/emacs/imaxima/*")))
```

```
(use-package! imaxima
  :when +maxima-ok-p
  :commands (imaxima imath-mode)
  :init
  (setq imaxima-use-maxima-mode-flag nil ;; otherwise, it don't render equations with LaTeX.
        imaxima-scale-factor 2.0)

  ;; Hook the `maxima-inferior-mode' to get Company completion.
  (add-hook 'imaxima-startup-hook #'maxima-inferior-mode))
```

8 Programming

8.1 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)
```

8.2 CSV rainbow

Stolen from here.

```
(after! csv-mode
  ;; TODO: Need to fix the case of two commas, example "a,b,,c,d"
  (require 'cl-lib)
  (require 'color)

  (map! :localleader
        :map csv-mode-map
        "R" #'+csv-rainbow)

  (defun +csv-rainbow (&optional separator)
    (interactive (list (when current-prefix-arg (read-char "Separator: "))))
    (font-lock-mode 1)
    (let* ((separator (or separator ?\,))
           (n (count-matches (string separator) (point-at-bol) (point-at-eol)))
           (colors (cl-loop for i from 0 to 1.0 by (/ 2.0 n)
                            collect (apply #'color-rgb-to-hex
                                           (color-hsl-to-rgb i 0.3 0.5)))))
      (cl-loop for i from 2 to n by 2
               for c in colors
               for r = (format "^\\([^%c\n]+%c\\)\\{%d\\}" separator separator i)
               do (font-lock-add-keywords nil `((,r (1 '(face (:foreground ,c))))))))))

;; provide CSV mode setup
;; (add-hook 'csv-mode-hook (lambda () (+csv-rainbow)))
```

8.3 ESS

View data frames better with

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

8.4 GNU Octave

Files with `.m` extension gets recognized automatically as Objective C files. Lets change this to be recognized as Octave/Matlab files.

```
(add-to-list 'auto-mode-alist '("\\.m\\'" . octave-mode))
```

8.5 ROS

8.5.1 Extensions

Add ROS specific file formats:

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

;; msg and srv files: for now use gdb-script-mode
(add-to-list 'auto-mode-alist '("\\.msg\\'"     . gdb-script-mode))
(add-to-list 'auto-mode-alist '("\\.srv\\'"     . gdb-script-mode))
(add-to-list 'auto-mode-alist '("\\.action\\'" . gdb-script-mode))
```

8.5.2 ROS bags

Mode to view ROS `.bag` files. Taken from code-iai/ros_emacs_utils.

```
(when +rosbag-ok-p
  (define-derived-mode rosbag-view-mode
    fundamental-mode "Rosbag view mode"
    "Major mode for viewing ROS bag files."
    (let ((f (buffer-file-name)))
      (let ((buffer-read-only nil))
        (erase-buffer)
        (message "Calling rosbag info")
        (call-process "rosbag" nil (current-buffer) nil
                      "info" f)
        (set-buffer-modified-p nil))
      (view-mode)
      (set-visited-file-name nil t)))

  ;; rosbag view mode
  (add-to-list 'auto-mode-alist '("\\.bag$" . rosbag-view-mode)))
```

8.5.3 `ros.el`

I found this awesome `ros.el` package made by Max Beutelspacher, which facilitate working with ROS machines, supports ROS1 and ROS2, with local workspaces or remote ones (over Trump!).

```
;; `ros.el' depends on `with-shell-interpreter' among other packages
;; See: https://github.com/DerBeutlin/ros.el/blob/master/Cask
(package! with-shell-interpreter)
(package! ros
  :recipe (:host github
           :repo "DerBeutlin/ros.el"))
```

Now, we configure the ROS1/ROS2 workspaces to work on. But before that, we need to install some tools on the ROS machine, and build the workspace for the first time using `colcon build`, the repository contains example Docker files for Noetic and Foxy.

```
(use-package! ros
  :init (map! :leader
              :prefix ("l" . "custom")
              :desc "Hydra ROS" "r" #'hydra-ros-main/body)
  :commands (hydra-ros-main/body ros-set-workspace)
  :config
  (setq ros-workspaces
        (list (ros-dump-workspace
               :tramp-prefix (format "/docker:%s@%s:" "ros" "ros-machine")
               :workspace "~/ros_ws"
               :extends '("/opt/ros/noetic/"))
              (ros-dump-workspace
               :tramp-prefix (format "/ssh:%s@%s:" "swd_sk" "172.16.96.42")
               :workspace "~/ros_ws"
               :extends '("/opt/ros/noetic/"))
              (ros-dump-workspace
               :tramp-prefix (format "/ssh:%s@%s:" "swd_sk" "172.16.96.42")
               :workspace "~/ros2_ws"
               :extends '("/opt/ros/foxy/")))))
```

8.6 Embedded systems

8.6.1 Embed.el

Some embedded systems development tools.

TODO: Try to integrate embedded debuggers adapters with `dap-mode`:

- probe-rs-debugger
- cortex-debug
- esp-debug-adapter

```
(package! embed
  :recipe (:host github
           :repo "sjsch/embed-el"))
```

```
(use-package! embed
  :commands (embed-openocd-start
             embed-openocd-stop
             embed-openocd-gdb
             embed-openocd-flash)

  :init
  (map! :leader :prefix ("l" . "custom")
        (:when (featurep! :tools debugger +lsp)
         :prefix-map ("e" . "embedded")
         :desc "Start OpenOCD"    "o" #'embed-openocd-start
         :desc "Stop OpenOCD"     "O" #'embed-openocd-stop
         :desc "OpenOCD GDB"      "g" #'embed-openocd-gdb
         :desc "OpenOCD flash"    "f" #'embed-openocd-flash)))
```

8.6.2 Bitbake (Yocto)

Add support for Yocto Project files.

```
(package! bitbake-modes
  :recipe (:host bitbucket
           :repo "olanilsson/bitbake-modes"))
```

```
(use-package bitbake-modes
  :commands (bitbake-mode
             conf-bitbake-mode
             bb-scc-mode wks-mode
             bitbake-task-log-mode
             bb-sh-mode
             mmm-mode))
```

8.7 Debugging

8.7.1 DAP

I like to use `cpptools` over `webfreak.debug`. So I enable it after loading `dap-mode`. I like also to have a mode minimal UI. And I like to trigger `dap-hydra` when the program hits a break point, and automatically delete the session and close Hydra when DAP is terminated.

```
(unpin! dap-mode)
```

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

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

  ;; 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)
              (call-interactively #'dap-delete-session)
              (dap-hydra/nil))))
```

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))
```

8.7.2 The Grand "Cathedral" Debugger

For C/C++, DAP mode is missing so much features. In my experience, both `cpptools` and `gdb` DAP interfaces aren't mature, it stops and disconnect while debugging, making it a double pain.

Additional commands There is no best than using pure GDB, it makes debugging more flexible. Lets define some missing GDB commands, add them to Hydra keys, and define some reverse debugging commands for usage with rr (which we can use by substituting `gdb` by `rr replay` when starting the session).

```
(after! realgud
  (require 'hydra)

  ;; Add some missing gdb/rr commands
  (defun ab/realgud:cmd-start (arg)
    "start = break main + run"
    (interactive "p")
    (realgud-command "start"))

  (defun ab/realgud:cmd-reverse-next (arg)
    "Reverse next"
    (interactive "p")
    (realgud-command "reverse-next"))

  (defun ab/realgud:cmd-reverse-step (arg)
    "Reverse step"
    (interactive "p")
    (realgud-command "reverse-step"))

  (defun ab/realgud:cmd-reverse-continue (arg)
    "Reverse continue"
    (interactive "p")
    (realgud-command "reverse-continue"))

  (defun ab/realgud:cmd-reverse-finish (arg)
    "Reverse finish"
    (interactive "p")
    (realgud-command "reverse-finish"))

  ;; Define a hydra binding
  (defhydra realgud-hydra (:color pink :hint nil :foreign-keys run)
    "
 Stepping  | _n_: next      | _i_: step    | _o_: finish  | _c_: continue  | _R_: restart  | _u_: until-here
 Revese    | _rn_: next     | _ri_: step   | _ro_: finish | _rc_: continue |
 Breakpts  | _ba_: break    | _bD_: delete | _bt_: tbreak | _bd_: disable  | _be_: enable  | _tr_: backtrace
 Eval      | _ee_: at-point | _er_: region | _eE_: eval   |
           | _!_: shell     | _Qk_: kill   | _Qq_: quit   | _Sg_: gdb      | _Ss_: start
"
    ("n"  realgud:cmd-next)
    ("i"  realgud:cmd-step)
    ("o"  realgud:cmd-finish)
    ("c"  realgud:cmd-continue)
    ("R"  realgud:cmd-restart)
    ("u"  realgud:cmd-until-here)
    ("rn" ab/realgud:cmd-reverse-next)
    ("ri" ab/realgud:cmd-reverse-step)
    ("ro" ab/realgud:cmd-reverse-finish)
    ("rc" ab/realgud:cmd-reverse-continue)
    ("ba" realgud:cmd-break)
    ("bt" realgud:cmd-tbreak)
    ("bD" realgud:cmd-delete)
    ("be" realgud:cmd-enable)
    ("bd" realgud:cmd-disable)
    ("ee" realgud:cmd-eval-at-point)
    ("er" realgud:cmd-eval-region)
    ("tr" realgud:cmd-backtrace)
    ("eE" realgud:cmd-eval)
    ("!"  realgud:cmd-shell)
    ("Qk" realgud:cmd-kill)
    ("Sg" realgud:gdb)
    ("Ss" ab/realgud:cmd-start)
    ("q"  nil "quit" :color blue) ;; :exit
    ("Qq" realgud:cmd-quit :color blue)) ;; :exit

  (defun +debugger/realgud:gdb-hydra ()
    "Run `realgud-hydra'."
    (interactive)
    (realgud-hydra/body))
```

```
69
70    (map! :leader :prefix ("l" . "custom")
71          (:when (featurep! :tools debugger)
72           :prefix-map ("d" . "debugger")
73           :desc "RealGUD hydra" "h" #'+debugger/realgud:gdb-hydra)))
```

RealGUD `.dir-locals.el` support (only for GDB) I do a lot of development on C/C++ apps that gets data from command line arguments, which means I have to type my arguments manually after calling `realgud:gdb`, which is very annoying.

For DAP mode, there is a support for either `dap-debug-edit-template`, or `launch.json`. For RealGUD though, I didn't find any ready-to-use feature like this. So I define a parameter list named `ab/realgud:launch-plist`, which supports `:program` and `:args`. The first is a string of the program path, and the second is a list of string arguments to pass to the program.

```
1  ;; A variable which to be used in .dir-locals.el, formatted as a property list;
2  ;; '(:program "..." :args ("args1" "arg2" ...))
3  ;; "${workspaceFolder}" => gets replaced with project workspace (from projectile)
4  ;; "${workspaceFolderBasename}" => gets replaced with project workspace's basename
5  (defvar ab/realgud:launch-plist nil)
```

This variable is set in a per-project basis thanks to `.dir-locals.el`, some thing like this:

```
1  ;; Example entry in .dir-locals.el
2  ((nil . ((ab/realgud:launch-plist . '(:program "${workspaceFolder}/build/bin/my_prog"
3                                        :args ("--in_file=${workspaceFolder}/some/file.csv"
4                                               "--out_file=/tmp/some_randome_file"
5                                               "-a"))))))
```

The special variables `${workspaceFolder}` and `${workspaceFolderBasename}` are defined as in VS Code, the actual values are filled from `projectile-project-root`.

```
1  (cl-defun ab/realgud:get-launch-debugger-args (&key program args)
2    (let ((debugger--args ""))
3      (when program
4        (setq debugger--args program)
5        (when args
6          (setq debugger--args (concat debugger--args " " (s-join " " args)))))
7      ;; Replace special variables
8      (let* ((ws--root (expand-file-name (or (projectile-project-root) ".")))
9             (ws--basename (file-name-nondirectory
10                           (if (s-ends-with-p "/" ws--root)
11                               (substring ws--root 0 -1)
12                             ws--root))))
13       (s-replace-all
14        (list (cons "${workspaceFolder}" ws--root)
15              (cons "${workspaceFolderBasename}" ws--basename))
16        debugger--args))))
17
18 (defun +debugger/realgud:gdb-launch ()
19   "Launch RealGUD with parameters from `ab/realgud:launch-plist'"
20   (interactive)
21   (require 'realgud)
22   (if ab/realgud:launch-plist
23       (realgud:gdb
24        (concat realgud:gdb-command-name
25                " --args "
26                (apply 'ab/realgud:get-launch-debugger-args ab/realgud:launch-plist)))
27     (progn
28       (message "Variable `ab/realgud:launch-plist' is `nil'")
29       (realgud:gdb))))
30
31 (map! :leader :prefix ("l" . "custom")
32       (:when (featurep! :tools debugger)
```

```
      :prefix-map ("d" . "debugger")
      :desc "RealGUD launch" "d" #'+debugger/realgud:gdb-launch))
```

Record and replay `rr` We then add some shortcuts to run `rr` from Emacs, the `rr record` takes the program name and arguments from my local `ab/realgud:gdb-launch-plist`, when `rr replay` respects the arguments configured in RealGUD's GDB command name. Some useful hints could be found here, here, here and here.

```
(after! realgud
  (require 's)

  (defun +debugger/rr-replay ()
    "Launch `rr replay'"
    (interactive)
    (realgud:gdb (s-replace "gdb" "rr replay" realgud:gdb-command-name)))

  (defun +debugger/rr-record ()
    "Launch `rr record' with parameters from `ab/realgud:launch-plist'"
    (interactive)
    (let ((debugger--args (apply 'ab/realgud:get-launch-debugger-args ab/realgud:launch-plist)))
      (unless (make-process :name "*rr record*"
                            :buffer "*rr record*"
                            :command (append '("rr" "record") (s-split " " debugger--args)))
        (message "Cannot make process 'rr record'"))))

  (map! :leader :prefix ("l" . "custom")
        (:when (featurep! :tools debugger)
         :prefix-map ("d" . "debugger")
         :desc "rr record"  "r" #'+debugger/rr-record
         :desc "rr replay"  "R" #'+debugger/rr-replay)))
```

8.7.3 GDB

Emacs GDB DAP mode is great, however, it is not mature for C/C++ debugging, it does not support some basic features like *Run until cursor*, *Show disassembled code*… etc. Emacs have builtin `gdb` support through `gdb-mi` and `gud`.

The `emacs-gdb` package overwrites the builtin `gdb-mi`, it is much faster (thanks to it's C module), and it defines some easy to use UI, with Visual Studio like keybindings.

```
(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 ;; TODO: Customize this
        gdb-ignore-gdbinit nil) ;; I use gdbinit to define some useful stuff
  ;; History
  (defvar +gdb-history-file "~/.gdb_history")
  (defun +gud-gdb-mode-hook-setup ()
    "GDB setup."

    ;; Suposes "~/.gdbinit" contains:
    ;; set history save on
    ;; set history filename ~/.gdb_history
    ;; set history remove-duplicates 2048
    (when (and (ring-empty-p comint-input-ring)
```

```
                (file-exists-p +gdb-history-file))
      (setq comint-input-ring-file-name +gdb-history-file)
      (comint-read-input-ring t)))

  (add-hook 'gud-gdb-mode-hook '+gud-gdb-mode-hook-setup))
```

Custom layout for `gdb-many-windows` Stolen from `https://stackoverflow.com/a/41326527/3058915`. I used it to change the builtin `gdb-many-windows` layout.

```
(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 c-editin
g)
  (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))
```

```
(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)
```

Highlight current line

8.8 Completion & IDE

8.8.1 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
  (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")))
```

8.8.2 LSP mode

Enable some useful UI stuff LSP mode provides a set of configurable UI stuff, Doom Emacs disables some of the UI components; 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 t ; not working properly with ccls!
        lsp-diagnostics-provider :auto
        lsp-enable-symbol-highlighting t
        lsp-headerline-breadcrumb-enable nil
        lsp-headerline-breadcrumb-segments '(symbols)))
```

```
(after! lsp-clangd
  (setq lsp-clients-clangd-args
        '("-j=4"
          "--background-index"
          "--clang-tidy"
          "--completion-style=detailed"
          "--header-insertion=never"
          "--header-insertion-decorators=0"))
  (set-lsp-priority! 'clangd 2))
```

LSP mode with `clangd`

```
;; NOTE: Not tangled, using the default 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
```

LSP mode with `ccls`

Enable `lsp` over `tramp`

```
(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 supported" error
        lsp-enable-symbol-highlighting nil)

  (lsp-register-client
   (make-lsp-client
    :new-connection (lsp-tramp-connection "pyls")
    :major-modes '(python-mode)
    :remote? t
    :server-id 'pyls-remote)))
```

Python

```
;; 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'
             ccls-args)))
    :major-modes '(c-mode c++-mode objc-mode cuda-mode)
    :remote? t
    :server-id 'ccls-remote))

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

C/C++ with `ccls`

```
(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 supported" error
        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 'ccls'
              lsp-clients-clangd-args)))
     :major-modes '(c-mode c++-mode objc-mode cuda-mode)
     :remote? t
     :server-id 'clangd-remote)))
```

C/C++ with `clangd`

VHDL By default, LSP uses the proprietary VHDL-Tool to provide LSP features; however, there is free and open source alternatives: `ghdl-ls` and `rust_hdl`. I have some issues running `ghdl-ls` installed form `pip` through the `pyghdl` package, so lets use `rust_hdl` instead.

```
(use-package! vhdl-mode
  ;; Required unless vhdl_ls is on the $PATH
  :config
  (setq lsp-vhdl-server-path "~/Projects/foss_projects/rust_hdl/target/release/vhdl_ls"
        lsp-vhdl-server 'vhdl-ls
        lsp-vhdl--params nil)
  (require 'lsp-vhdl)

  :hook (vhdl-mode . (lambda ()
                       (lsp t)
                       (flycheck-mode t))))
```

```
(package! lsp-sonarlint)
```

SonarLint

```
;; TODO: configure it, for the moment, it seems that it doesn't support C/C++
```

8.8.3 Cppcheck

Check for everything!

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

8.8.4 Project CMake

A good new package to facilitate using CMake projects with Emacs, it glues together `project`, `eglot`, `cmake` and `clangd`.

```
(package! project-cmake
  :disable (not (featurep! :tools lsp +eglot)) ; Enable only if (lsp +eglot) is used
  :recipe (:host github
           :repo "juanjosegarciaripoll/project-cmake"))
```

```
(use-package! project-cmake
    :config
    (require 'eglot)
    (project-cmake-scan-kits)
    (project-cmake-eglot-integration))
```

8.8.5 Unibeautify

The next-gen beautifier? Add initial support for Unibeautify.

`npm install -g @unibeautify/cli`

```
(package! unibeautify
  :recipe (:host github
           :repo "Unibeautify/emacs"))
```

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

8.8.6 FZF

```
(package! fzf)
```

```
(after! evil
  (evil-define-key 'insert fzf-mode-map (kbd "ESC") #'term-kill-subjob))

(define-minor-mode fzf-mode
  "Minor mode for the FZF buffer"
  :init-value nil
  :lighter " FZF"
  :keymap '(("C-c" . term-kill-subjob)))

(defadvice! doom-fzf--override-start-args-a (original-fn &rest args)
  "Set the FZF minor mode with the fzf buffer."
  :around #'fzf/start
  (message "called with args %S" args)
  (apply original-fn args)

  ;; set the FZF buffer to fzf-mode so we can hook ctrl+c
  (set-buffer "*fzf*")
  (fzf-mode))

(defvar fzf/args
  "-x --print-query -m --tiebreak=index --expect=ctrl-v,ctrl-x,ctrl-t")

(use-package! fzf
  :commands (fzf fzf-projectile fzf-hg fzf-git fzf-git-files fzf-directory fzf-git-grep))
```

8.8.7 Clang-format

```
(package! clang-format)
```

```
(use-package! clang-format
  :when +clang-format-ok-p
  :commands (clang-format-region))
```

8.9 Git & VC

8.9.1 Repo

Make sure the repo tool is installed, if not, `pacman -S repo` on Arch-based distributions, 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
  :when +repo-ok-p
  :commands repo-status)
```

8.9.2 Magit :heart: Delta

Integrate `git-delta` with `magit`. Currently disabled because it is too slow, specially with big change chunks.

```
(package! magit-delta
  :disable t) ;; Disabled, too slow on big chunks
```

```
(use-package! magit-delta
  :when +delta-ok-p
  :commands magit-status
  :hook (magit-mode . magit-delta-mode))
```

8.9.3 Blamer

Display Git information (author, date, message…) for current line

```
(package! blamer
  :recipe (:host github
           :repo "artawower/blamer.el"))
```

```
(use-package! blamer
  :custom
  (blamer-idle-time 0.3)
  (blamer-min-offset 60)
  (blamer-prettify-time-p t)
  (blamer-entire-formatter "    %s")
  (blamer-author-formatter " %s ")
```

```
  (blamer-datetime-formatter "[%s], ")
  (blamer-commit-formatter ""%s"")

  :custom-face
  (blamer-face ((t :foreground "#7a88cf"
                   :background nil
                   :height 125
                   :italic t)))

  :hook ((prog-mode . blamer-mode)
         (text-mode . blamer-mode))

  :config
  (when (featurep! :ui zen) ;; Disable in zen (writeroom) mode
    (add-hook! 'writeroom-mode-enable-hook (blamer-mode -1))
    (add-hook! 'writeroom-mode-disable-hook (blamer-mode 1))))
```

8.10 Assembly

Add some packages for better assembly coding.

```
(package! nasm-mode)
(package! haxor-mode)
(package! mips-mode)
(package! riscv-mode)
(package! x86-lookup)
```

```
(use-package! nasm-mode
  :mode "\\.[n]*\\(asm\\|s\\)\\'")

;; Get Haxor VM from https://github.com/krzysztof-magosa/haxor
(use-package! haxor-mode
  :mode "\\.hax\\'")

(use-package! mips-mode
  :mode "\\.mips$")

(use-package! riscv-mode
  :commands (riscv-mode)
  :mode "\\.riscv$")

(use-package! x86-lookup
  :commands (x86-lookup)
  :config
  (when (featurep! :tools pdf)
     (setq x86-lookup-browse-pdf-function 'x86-lookup-browse-pdf-pdf-tools))
  ;; Get manual from https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
  (setq x86-lookup-pdf "assets/325383-sdm-vol-2abcd.pdf"))
```

8.11 Disaster

```
(package! disaster)
```

```
;; TODO: Configure to take into account "compile_commands.json"
(use-package! disaster
  :commands (disaster))
```

8.12 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)))
```

8.13 Systemd

For editing systemd unit files.

```
(package! systemd)
```

8.14 Franca IDL

Add support for *Franca Interface Definition Language*.

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

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

8.15 LaTeX

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

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

8.16 Flycheck :heart: Projectile

WIP: Not working atm!

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

```
(use-package! flycheck-projectile
  :commands flycheck-projectile-list-errors)
```

8.17 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))
```

8.18 Inspector

```
(package! inspector
  :recipe (:host github
           :repo "mmontone/emacs-inspector"))
```

```
(use-package! inspector
  :commands (inspect-expression inspect-last-sexp))
```

9 Office

9.1 Org mode additional packages

To avoid problems in the `(after! org)` section.

```
(package! org-super-agenda)

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

(package! doct)

(package! org-appear)

;; To avoid problems with org-roam-ui
(unpin! org-roam)
(package! websocket)
(package! org-roam-ui)

(package! org-wild-notifier)

(package! org-modern
  :recipe (:host github
           :repo "minad/org-modern"))

(package! org-fragtog)

(package! org-bib
  :recipe (:host github
           :repo "rougier/org-bib-mode"))

(package! org-ref)

(package! academic-phrases
  :recipe (:host github
           :repo "nashamri/academic-phrases"))
```

9.2 Org mode

9.2.1 Intro

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

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

9.2.2 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-tags-column 0
      org-catch-invisible-edits 'smart      ;; try not to accidently do weird stuff in invisible regions
      org-export-with-sub-superscripts '{}  ;; don't treat lone _ / ^ as sub/superscripts, require _{} / ^{}
      org-auto-align-tags nil
      org-special-ctrl-a/e t
      org-startup-indented t ;; Enable 'org-indent-mode' by default, override with '+#startup: noindent' for big
↪  files
      org-insert-heading-respect-content t)
```

Org basics

Babel 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")))
```

Babel is really annoying when it comes to working with Scheme (via Geiser), it keeps asking about which Scheme implementation to use, I tried to set this in file local variables (with the) and dir-locals but it didn't work. This should work now!

```
(after! geiser
  (setq geiser-default-implementation 'guile))

;; stolen from https://github.com/yohan-pereira/.emacs#babel-config
(defun +org-confirm-babel-evaluate (lang body)
  (not (string= lang "scheme"))) ;; don't ask for scheme

(setq org-confirm-babel-evaluate #'+org-confirm-babel-evaluate)
```

Visual line / auto fill 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 plain text 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)
```

EVIL 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)
```

```
(setq org-todo-keywords
      '((sequence "TODO(t)" "NEXT(n)" "PROJ(p)" "STRT(s)" "WAIT(w)" "HOLD(h)" "IDEA(i)" "|" "DONE(d)" "KILL(k)")
        (sequence "[ ](T)" "[-](S)" "|" "[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)
```

TODOs

Agenda Set files for `org-agenda`

```
(setq org-agenda-files (list (expand-file-name "inbox.org" org-directory)
                             (expand-file-name "agenda.org" org-directory)
                             (expand-file-name "gcal-agenda.org" org-directory)
                             (expand-file-name "notes.org" org-directory)
                             (expand-file-name "projects.org" org-directory)))
```

Apply some styling on the standard agenda:

```
;; Agenda styling
(setq org-agenda-block-separator ?
      org-agenda-time-grid
      '((daily today require-timed)
        (800 1000 1200 1400 1600 1800 2000)
        "     " "        ")
      org-agenda-current-time-string
      "  now                          ")
```

Super agenda Configure `org-super-agenda`

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

Calendar

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

```
(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")))
```

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

TODO CalDAV Need to be configured, see the github repo.

```
(use-package! caldav
  :commands (org-caldav-sync))
```

Capture Set capture files

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

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) template))
                                                 (spec (plist-get (plist-get props :doct) :icon)))
                                       (setf (nth 1 template) (concat (+doct-icon-declaration-to-icon spec)
                                                                      "\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
↪  %^{recipiant}"
                              "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)
↪  "\tAbort")))))))
(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 'bold)
↪  (propertize ">" 'face 'font-lock-comment-face) "  " desc "…" "\n")))
                      ;; 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 'bold) "
↪    " desc "\n")
                         (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)) description))
                    (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 changing
↪  breaks stuff
(setq +org-capture-fn
      (lambda ()
        (interactive)
        (set-window-parameter nil 'mode-line-format 'none)
        (org-capture)))
```

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

```
(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))
```

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

Basic settings

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)
```

Mode line 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)))
```

```
(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: ${title}\n#+created: %U\n\n${body}\n")
           :unnarrowed t))))
```

Org Roam Capture template

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

```
(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))))
```

Org notifier Add support for `org-wild-notifier`.

```
(use-package! org-wild-notifier
  :hook (org-load . org-wild-notifier-mode)
  :config
  (setq org-wild-notifier-alert-time '(60 30)))
```

9.2.3 Custom links

Sub-figures 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
```

9.2.4 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

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)
        (1.000 . org-warning)
        (0.500 . org-upcoming-deadline)
        (0.000 . 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))
```

```
(setq org-inline-src-prettify-results '(" " . " ")
      doom-themes-org-fontify-special-tags nil)
```

Inline blocks

```
(use-package! org-modern
  :init
  (setq org-modern-table-vertical 1
        org-modern-table-horizontal 1
        org-modern-keyword nil) ;; I like to fontify them using ligatures

  (global-org-modern-mode))
```

Org Modern

```
;; From https://www.reddit.com/r/orgmode/comments/i6hl8b/comment/g1vsef2/
;; Scale image previews to 60% of the window width.
(setq org-image-actual-width (truncate (* (window-pixel-width) 0.6)))
```

Image previews

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

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

```
;; Org styling, hide markup etc.
(setq org-hide-emphasis-markers t
      org-pretty-entities t
      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)))
```

Symbols

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                " "
            :language                " "
            :author                  " "
            :email                   "@"
            :date                    " "
            :property                " "
            :options                 " "
            :startup                 " "
            :hugo_base_dir           " "
            :macro                   " "
            :html_head               " "
            :html                    " "
            :latex_class             " "
            :latex_class_options     " "
            :latex_header            " "
            :latex_header_extra      " "
            :latex_compiler          " "
            :beamer_header           " "
```

```
            :latex                  " "
            :attr_latex             " "
            :attr_html              " "
            :attr_org               " "
            :begin_quote            " "
            :end_quote              " "
            :begin_signature        " "
            :end_signature          " "
            :caption                " "
            :name                   " "
            :header                 "›"
            :results                " "
            :results_small          " "
            :begin_export           " "
            :end_export             " "
            :filetags               "#"
            :created                " "
            :include                " "
            :setupfile              " "
            :export_file_name       " "
            :export_select_tags     " "
            :export_exclude_tags    " "
            :properties             " "
            :end                    " "
            :properties_small       " "
            :end_small              " "
            :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:"
  :language               "#+language:"
  :author                 "#+author:"
  :email                  "#+email:"
  :date                   "#+date:"
  :property               "#+property:"
  :options                "#+options:"
  :startup                "#+startup:"
  :hugo_base_dir          "#+hugo_base_dir:"
  :macro                  "#+macro:"
  :html_head              "#+html_head:"
  :html                   "#+html:"
  :latex_class            "#+latex_class:"
  :latex_class_options    "#+latex_class_options:"
  :latex_header           "#+latex_header:"
  :latex_header_extra     "#+latex_header_extra:"
  :latex_compiler         "#+latex_compiler:"
  :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"
  :begin_signature        "#+begin_signature"
  :end_signature          "#+end_signature"
  :caption                "#+caption:"
```

```
  :header                 "#+header:"
  :begin_export           "#+begin_export"
  :end_export             "#+end_export"
  :filetags               "#+filetags:"
  :created                "#+created:"
  :include                "#+include:"
  :setupfile              "#+setupfile:"
  :export_file_name       "#+export_file_name:"
  :export_select_tags     "#+export_select_tags:"
  :export_exclude_tags    "#+export_exclude_tags:"
  :results                "#+RESULTS:"
  :results_small          "#+results:"
  :property               ":PROPERTIES:"
  :end                    ":END:"
  :property_small         ":properties:"
  :end_small              ":end:"
  :priority_a             "[#A]"
  :priority_b             "[#B]"
  :priority_c             "[#C]"
  :priority_d             "[#D]"
  :priority_e             "[#E]")
```

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.

```
;; (setq org-format-latex-header "\\documentclass{article}
;; \\usepackage[usenames]{xcolor}
;; \\usepackage[T1]{fontenc}
;; \\usepackage{booktabs}

;; \\pagestyle{empty} % do not remove

;; \\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}
```

```
;; \\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"))

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

;; Define a function to set the format latex scale (to be reused in hooks)
(defun ab/set-org-latex-scale (scale)
  (setq org-format-latex-options
        (plist-put org-format-latex-options :scale scale)))

;; Set the default scale
(ab/set-org-latex-scale 1.4)

;; Increase scale in Zen mode
(when (featurep! :ui zen)
  (add-hook! 'writeroom-mode-enable-hook (ab/set-org-latex-scale 2.0))
  (add-hook! 'writeroom-mode-disable-hook (ab/set-org-latex-scale 1.4)))
```

Better equation numbering Numbered equations all have (1) as the number for fragments with vanilla org-mode. This code (from scimax) injects the correct numbers into the previews so they look good. **Not working right now!**

```
(defun scimax-org-renumber-environment (orig-func &rest args)
  "A function to inject numbers in LaTeX fragment previews."
  (let ((results '())
        (counter -1)
        (numberp))
    (setq results (cl-loop for (begin .  env) in
                           (org-element-map (org-element-parse-buffer) 'latex-environment
                             (lambda (env)
                               (cons
                                (org-element-property :begin env)
                                (org-element-property :value env))))
                           collect
                           (cond
                            ((and (string-match "\\\\begin{equation}" env)
                                  (not (string-match "\\\\tag{" env)))
                             (cl-incf counter)
                             (cons begin counter))
                            ((string-match "\\\\begin{align}" env)
                             (prog2
                                 (cl-incf counter)
                                 (cons begin counter)
                               (with-temp-buffer
                                 (insert env)
                                 (goto-char (point-min))
                                 ;; \\ is used for a new line. Each one leads to a number
                                 (cl-incf counter (count-matches "\\\\$"))
                                 ;; unless there are nonumbers.
                                 (goto-char (point-min))
                                 (cl-decf counter (count-matches "\\nonumber")))))
                            (t
                             (cons begin nil)))))

    (when (setq numberp (cdr (assoc (point) results)))
      (setf (car args)
            (concat
             (format "\\setcounter{equation}{%s}\n" numberp)
             (car args))))))
```

```
  (apply orig-func args))


(defun scimax-toggle-latex-equation-numbering ()
  "Toggle whether LaTeX fragments are numbered."
  (interactive)
  (if (not (get 'scimax-org-renumber-environment 'enabled))
      (progn
        (advice-add 'org-create-formula-image :around #'scimax-org-renumber-environment)
        (put 'scimax-org-renumber-environment 'enabled t)
        (message "Latex numbering enabled"))
    (advice-remove 'org-create-formula-image #'scimax-org-renumber-environment)
    (put 'scimax-org-renumber-environment 'enabled nil)
    (message "Latex numbering disabled.")))
```

Fragtog Hook `org-fragtog-mode` to `org-mode`.

```
(use-package! org-fragtog
  :hook (org-mode . org-fragtog-mode))
```

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))
```

9.2.5 Bibliography

```
(setq bibtex-completion-bibliography '("~/Zotero/library.bib")
      bibtex-completion-library-path '("~/Zotero/storage/")
      bibtex-completion-notes-path "~/PhD/bibliography/notes/"
      bibtex-completion-notes-template-multiple-files "* ${author-or-editor}, ${title}, ${journal}, (${year})
↪  :${=type=}: \n\nSee [[cite:&${=key=}]]\n"
      bibtex-completion-additional-search-fields '(keywords)
      bibtex-completion-display-formats
      '((article       . "${=has-pdf=:1}${=has-note=:1} ${year:4} ${author:36} ${title:*} ${journal:40}")
        (inbook        . "${=has-pdf=:1}${=has-note=:1} ${year:4} ${author:36} ${title:*} Chapter
↪  ${chapter:32}")
        (incollection  . "${=has-pdf=:1}${=has-note=:1} ${year:4} ${author:36} ${title:*} ${booktitle:40}")
        (inproceedings . "${=has-pdf=:1}${=has-note=:1} ${year:4} ${author:36} ${title:*} ${booktitle:40}")
        (t             . "${=has-pdf=:1}${=has-note=:1} ${year:4} ${author:36} ${title:*}"))
      bibtex-completion-pdf-open-function
      (lambda (fpath)
        (call-process "open" nil 0 nil fpath)))
```

BibTeX

Org-bib A mode to work with annotated bibliography in Org-Mode. See the repo for an example.

```
(use-package! org-bib
  :commands (org-bib-mode))
```

Org-ref Use Org as LaTeX!

```
(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 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 key)))
      (with-current-buffer (find-file-noselect (cdr results))
        (save-excursion
          (bibtex-search-entry (car results))
          (org-ref-open-bibtex-pdf))))))
```

```
(setq citar-library-paths '("~/Zotero/storage")
      citar-notes-paths '("~/PhD/bibliography/notes/")
      citar-bibliography '("~/Zotero/library.bib"))
```

Citar

9.2.6 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.

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

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.

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

```elisp
;; Add 'svg' package to display SVG pictures (uses inkscape, imagemagik and ghostscript)
;; (when +inkscape-ok-p
;;  (add-to-list 'org-latex-packages-alist '("" "svg")))
;; (add-to-list 'org-latex-packages-alist '("" "fontspec")) ;; for xelatex
;; (add-to-list 'org-latex-packages-alist '("utf8" "inputenc"))
```

Org LaTeX packages

```elisp
;; this is for code syntax highlighting in export. you need to use
;; -shell-escape with latex, and install pygments.
;; (add-to-list 'org-latex-packages-alist '("svgnames" "xcolor"))
;; (add-to-list 'org-latex-packages-alist '("" "minted"))

;; (setq org-latex-listings 'minted) ;; Per document, in local variables
(setq org-latex-minted-options '(("frame" "lines")
                                 ("fontsize" "\\footnotesize")
                                 ("tabsize" "2")
                                 ("breaklines" "")
                                 ("breakanywhere" "") ;; break anywhere, no just on spaces
                                 ("style" "default")
                                 ("bgcolor" "GhostWhite")
                                 ("linenos" "")))

(dolist (pair '((ipython    "python")
                (jupyter    "python")
                (scheme     "scheme")
                (lisp-data  "lisp")
                (conf       "ini")
                (conf-unix  "unixconfig")
                (conf-space "unixconfig")
                (conf-toml  "yaml")
                (gitconfig  "ini")
                (systemd    "ini")
                (gdb-script "text")))
  (unless (member pair org-latex-minted-langs)
    (add-to-list 'org-latex-minted-langs pair)))
```

Export PDFs with syntax highlighting

```
(after! ox-latex
  (add-to-list 'org-latex-classes
               '("scr-article"
                 "\\documentclass{scrartcl}"
                 ("\\section{%s}" . "\\section*{%s}")
                 ("\\subsection{%s}" . "\\subsection*{%s}")
                 ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                 ("\\paragraph{%s}" . "\\paragraph*{%s}")
                 ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
  (add-to-list 'org-latex-classes
               '("blank"
                 "[NO-DEFAULT-PACKAGES]\n[NO-PACKAGES]\n[EXTRA]"
                 ("\\section{%s}" . "\\section*{%s}")
                 ("\\subsection{%s}" . "\\subsection*{%s}")
                 ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                 ("\\paragraph{%s}" . "\\paragraph*{%s}")
                 ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
  (add-to-list 'org-latex-classes
               '("bmc-article"
                 "\\documentclass[article,code,maths]{bmc}\n[NO-DEFAULT-PACKAGES]\n[NO-PACKAGES]\n[EXTRA]"
                 ("\\section{%s}" . "\\section*{%s}")
                 ("\\subsection{%s}" . "\\subsection*{%s}")
                 ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                 ("\\paragraph{%s}" . "\\paragraph*{%s}")
                 ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
  (add-to-list 'org-latex-classes
               '("bmc"
                 "\\documentclass[code,maths]{bmc}\n[NO-DEFAULT-PACKAGES]\n[NO-PACKAGES]\n[EXTRA]"
                 ("\\chapter{%s}" . "\\chapter*{%s}")
                 ("\\section{%s}" . "\\section*{%s}")
                 ("\\subsection{%s}" . "\\subsection*{%s}")
                 ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                 ("\\paragraph{%s}" . "\\paragraph*{%s}")
                 ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
  (add-to-list 'org-latex-classes
               '("IEEEtran"
                 "\\documentclass{IEEEtran}"
                 ("\\section{%s}" . "\\section*{%s}")
                 ("\\subsection{%s}" . "\\subsection*{%s}")
                 ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                 ("\\paragraph{%s}" . "\\paragraph*{%s}")
                 ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
  (add-to-list 'org-latex-classes
               '("thesis"
                 "\\documentclass[11pt]{book}"
                 ("\\chapter{%s}" . "\\chapter*{%s}")
                 ("\\section{%s}" . "\\section*{%s}")
                 ("\\subsection{%s}" . "\\subsection*{%s}")
                 ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                 ("\\paragraph{%s}" . "\\paragraph*{%s}")))
  (add-to-list 'org-latex-classes
               '("thesis-fr"
                 "\\documentclass[french,12pt,a4paper]{book}"
                 ("\\chapter{%s}" . "\\chapter*{%s}")
                 ("\\section{%s}" . "\\section*{%s}")
                 ("\\subsection{%s}" . "\\subsection*{%s}")
                 ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                 ("\\paragraph{%s}" . "\\paragraph*{%s}"))))

(setq org-latex-default-class "article")
;; org-latex-tables-booktabs t
;; org-latex-reference-command "\\cref{%s}")
```

Class templates

Hugo Update files with last modified date, when `#+lastmod:` is available

```
(setq time-stamp-active t
      time-stamp-start "#\\+lastmod:[ \t]*"
      time-stamp-end "$"
      time-stamp-format "%04Y-%02m-%02d")

(add-hook 'before-save-hook 'time-stamp nil)
```

9.3 Text editing

9.3.1 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))))
```

9.3.2 Academic phrases

When writing your academic paper, you might get stuck trying to find the right phrase that captures your intention. This package tries to alleviate that problem by presenting you with a list of phrases organized by the topic or by the paper section that you are writing. This package has around 600 phrases so far.

This is based on the book titled "English for Writing Research - Papers Useful Phrases".

```
(use-package! academic-phrases
  :commands (academic-phrases
             academic-phrases-by-section))
```

9.3.3 Quarto

Integration of Quarto in Emacs.

```
(package! quarto-mode)
```

```
(use-package! quarto-mode
  :when +quarto-ok-p)
```

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.

```
[Desktop Entry]
Name=Emacs Org-Protocol
Exec=emacsclient %u
Icon=/home/hacko/.doom.d/assets/org-mode.svg
Type=Application
Terminal=false
MimeType=x-scheme-handler/org-protocol
```

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.

```
read -p "Do you want to set Chrome/Brave to show the 'Always open ...' checkbox, to be used with the
↪  'org-protocol://' registration? [Y | N]: " INSTALL_CONFIRM

if [[ $INSTALL_CONFIRM == "Y" ]]
then
  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
fi
```

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.

```
*.tex                                   diff=tex
*.bib                                   diff=bibtex
*.{c,h,c++,h++,cc,hh,cpp,hpp}           diff=cpp
*.m                                     diff=matlab
*.py                                    diff=python
*.rb                                    diff=ruby
*.php                                   diff=php
*.pl                                    diff=perl
*.{html,xhtml}                          diff=html
*.f                                     diff=fortran
*.{el,lisp,scm}                         diff=lisp
*.r                                     diff=rstats
*.texi*                                 diff=texinfo
*.org                                   diff=org
*.rs                                    diff=rust

*.odt                                   diff=odt
*.odp                                   diff=libreoffice
*.ods                                   diff=libreoffice
*.doc                                   diff=doc
*.xls                                   diff=xls
*.ppt                                   diff=ppt
*.docx                                  diff=docx
*.xlsx                                  diff=xlsx
*.pptx                                  diff=pptx
*.rtf                                   diff=rtf

*.{png,jpg,jpeg,gif}                    diff=exif

*.pdf                                   diff=pdf
*.djvu                                  diff=djvu
*.epub                                  diff=pandoc
*.chm                                   diff=tika
*.mhtml?                                diff=tika

*.{class,jar}                           diff=tika
*.{rar,7z,zip,apk}                      diff=tika
```

Then adding some regular expressions for it to `~/.config/git/config`, with some tools to view diffs on binary files.

```
# ====== TEXT FORMATS ======
[diff "org"]
  xfuncname = "^(\\*+ +.*)$"

[diff "lisp"]
  xfuncname = "^(\\(.*)$"

[diff "rstats"]
  xfuncname = "^([a-zA-z.]+ <- function.*)$"

[diff "texinfo"]
# from http://git.savannah.gnu.org/gitweb/?p=coreutils.git;a=blob;f=.gitattributes;h=c3b2926c78c939d94358cc63d05
↪  1a70d38cfea5d;hb=HEAD
  xfuncname = "^@node[ \t][ \t]*\\([^,][^,]*\\)"

[diff "orgmode"]
  xfuncname = "^(\\*+.*)$"

[diff "rust"]
  xfuncname = "^[ \t]*(pub|)[ \t]*((fn|struct|enum|impl|trait|mod)[^;]*)$"

```

```
# ===== BINARY FORMATS =====
[diff "pdf"]
  binary = true
# textconv = pdfinfo
# textconv = sh -c 'pdftotext "$@" -' # sudo apt install pdftotext
  textconv = sh -c 'pdftotext -layout "$0" -enc UTF-8 -nopgbrk -q -'
  cachetextconv = true

[diff "djvu"]
  binary = true
# textconv = pdfinfo
  textconv = djvutxt # yay -S djvulibre
  cachetextconv = true

[diff "odt"]
  textconv = odt2txt
# textconv = pandoc --standalone --from=odt --to=plain
  binary = true
  cachetextconv = true

[diff "doc"]
# textconv = wvText
  textconv = catdoc # yay -S catdoc
  binary = true
  cachetextconv = true

[diff "xls"]
# textconv = in2csv
# textconv = xlscat -a UTF-8
# textconv = soffice --headless --convert-to csv
  textconv = xls2csv # yay -S catdoc
  binary = true
  cachetextconv = true

[diff "ppt"]
  textconv = catppt # yay -S catdoc
  binary = true
  cachetextconv = true

[diff "docx"]
  textconv = pandoc --standalone --from=docx --to=plain
# textconv = sh -c 'docx2txt.pl "$0" -'
  binary = true
  cachetextconv = true

[diff "xlsx"]
  textconv = xlsx2csv # pip install xlsx2csv
# textconv = in2csv
# textconv = soffice --headless --convert-to csv
  binary = true
  cachetextconv = true

[diff "pptx"]
# pip install --user pptx2md (currently not wotking with Python 3.10)
# textconv = sh -c 'pptx2md --disable_image --disable_wmf -i "$0" -o ~/.cache/git/presentation.md >/dev/null &&
↪  cat ~/.cache/git/presentation.md'
# Alternative hack, convert PPTX to PPT, then use the catppt tool
  textconv = sh -c 'soffice --headless --convert-to ppt --outdir /tmp "$0" && TMP_FILENAME=$(basename -- "$0")
↪  && catppt "/tmp/${TMP_FILENAME%.*}.ppt"'
  binary = true
  cachetextconv = true

[diff "rtf"]
  textconv = unrtf --text # yay -S unrtf
  binary = true
  cachetextconv = true

[diff "epub"]
  textconv = pandoc --standalone --from=epub --to=plain
  binary = true
```

```
  cachetextconv = true

[diff "tika"]
  textconv = tika --config=~/.local/share/tika/tika-conf.xml --text
  binary = true
  cachetextconv = true

[diff "libreoffice"]
  textconv = soffice --cat
  binary = true
  cachetextconv = true

[diff "exif"]
  binary = true
  textconv = exiftool # sudo apt install perl-image-exiftool
```

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 (pattern:
    ↪  X.X.X/)
    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 successfully"
    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.

```
[Unit]
Description=Emacs server daemon
Documentation=info:emacs man:emacs(1) https://gnu.org/software/emacs/

[Service]
Type=forking
ExecStart=sh -c 'emacs --daemon && emacsclient -c --eval "(delete-frame)"'
ExecStop=/usr/bin/emacsclient --no-wait --eval "(progn (setq kill-emacs-hook nil) (kill-emacs))"
Restart=on-failure

[Install]
WantedBy=default.target
```

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.

```
[Desktop Entry]
Name=Emacs (Client)
GenericName=Text Editor
Comment=A flexible platform for end-user applications
MimeType=text/english;text/plain;text/org;text/x-makefile;text/x-c++hdr;text/x-c++src;text/x-chdr;text/x-csrc;t
↪  ext/x-java;text/x-moc;text/x-pascal;text/x-tcl;text/x-tex;application/x-shellscript;text/x-c;text/x-c++;
Exec=emacsclient -create-frame --frame-parameters="'(fullscreen . maximized)"
↪  --alternate-editor="/usr/bin/emacs" --no-wait %F
Icon=emacs
Type=Application
Terminal=false
Categories=TextEditor;Utility;
StartupWMClass=Emacs
Keywords=Text;Editor;
X-KDE-StartupNotify=false
```

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-contents
  ↪  \"$TMP\") (delete-file \"$TMP\")${stdin_mode})")
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="/usr/bin/emacs" "${args[@]}"
else
  if ! $force_wait
  then
    args+=(--no-wait)
  fi

  emacsclient -create-frame --alternate-editor="/usr/bin/emacs" "${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.

```
# 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))"'
```

And this to launch Emacs in terminal mode `et`, I use this as a default `$EDITOR`

```
#!/usr/bin/env bash
e -t "$@"
```

And `ev` for use with `$VISUAL`:

```
#!/usr/bin/env bash
e -w "$@"
```

```
export EDITOR=$HOME/.local/bin/et
# export VISUAL=$HOME/.local/bin/ev
```

10.5 AppImage

Install/update the `appimageupdatetool.AppImage` tool:

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

  if [ -f ${APPIMAGE_UPDATE_TOOL_PATH} ] && $APPIMAGE_UPDATE_TOOL_PATH -j ${APPIMAGE_UPDATE_TOOL_PATH} 2&>/dev/null
  then
    echo "${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_TOOL_PATH}"
    fi
    wget -O ${APPIMAGE_UPDATE_TOOL_PATH} ${APPIMAGE_UPDATE_TOOL_URL} 2&>/dev/null &&
        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;
```

10.6 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/Softwares/src/esp-idf/ ]
then
  alias esp-prepare-env='source $HOME/Softwares/src/esp-idf/export.sh'
  alias esp-update='echo "Updating ESP-IDF framework..." && cd $HOME/src/esp-idf && git pull --all && echo
  ↪  "Updated successfully"'
else
  alias esp-prepare-env='echo "esp-idf repo not found. You can clone the esp-idf repo using git clone
  ↪  https://github.com/espressif/esp-idf.git"'
  alias esp-update=esp-prepare-env
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=$HOME/Softwares/src/install
export PATH=$PATH:$HOME/.cargo/bin:$HOME/Softwares/src/install/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"

# Tell Node about these packages
export NODE_PATH="$NPM_PACKAGES/lib/node_modules:$NODE_PATH"
```

Tell NPM to use this directory for its global package installs by adding this in `~/.npmrc`:

```
prefix = ~/.npm-packages
```

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/null

# Add ~/.config/emacs.doom/bin to path (for DOOM Emacs stuff)
export PATH=$PATH:$HOME/.config/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 not running inside Emacs (via vterm/eshell...)
if [ -z $INSIDE_EMACS ]
then
  if command -v tmux &> /dev/null && [ -z "$TMUX" ]
  then
    tmux attach -t default || tmux new -s default
  fi
fi
```

Define some environment variables.

```
export DS_DIR=~/PhD/datasets-no/experiment_images/
```

Load my `bitwarden-cli` session, exported to `BW_SESSION`.

```
source ~/.bitwarden-session
```

10.7 System dark theme 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`.

```
[Desktop Entry]
Type=Application
Name=Zotero
GenericName=A free, easy-to-use tool to help you collect, organize, cite, and share your research sources.
Icon=zotero
Exec=GTK_THEME=Default /usr/bin/zotero --url %u
Categories=Office
Terminal=false
MimeType=x-scheme-handler/zotero
```

Same thing for Scilab

```
[Desktop Entry]
Comment=Scientific software package for numerical computations
Exec=GTK_THEME=Default scilab -f %f
GenericName=Scientific Software Package
Icon=scilab
Name=Scilab
StartupNotify=false
Terminal=false
Type=Application
Categories=Science;Math;
Keywords=Science;Math;Numerical;Simulation
MimeType=application/x-scilab-sci;application/x-scilab-sce;application/x-scilab-tst;application/x-scilab-dem;ap
↪ plication/x-scilab-sod;application/x-scilab-xcos;application/x-scilab-zcos;application/x-scilab-bin;applica
↪ tion/x-scilab-cosf;application/x-scilab-cos;
```

```
[Desktop Entry]
Comment=Hybrid simulator
Exec=GTK_THEME=Default xcos
GenericName=Scientific Software Package
Icon=xcos
Name=Xcos
StartupNotify=false
Terminal=false
Type=Application
Categories=Science;Physics;
Keywords=Science;Physics;Simulation
MimeType=application/x-scilab-xcos;application/x-scilab-zcos;application/x-scilab-cosf;application/x-scilab-cos;
```

10.8 Rust format

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

```
# Rust edition 2018
edition = "2018"

# Use Unix style newlines, with 2 spaces tabulation.
newline_style = "Unix"
tab_spaces = 2
hard_tabs = false

# Make one line functions in a single line
fn_single_line = true

# Format strings
format_strings = true

# Increase the max line width
max_width = 120

# Merge nested imports
merge_imports = true

# Enum and Struct alignement
enum_discrim_align_threshold = 20
struct_field_align_threshold = 20

# Reorder impl items: type > const > macros > methods.
reorder_impl_items = true

# Comments and documentation formating
wrap_comments = true
normalize_comments = true
normalize_doc_attributes = true
format_code_in_doc_comments = true
report_fixme = "Always"
todo = "Always"
```

10.9 Ecryptfs

10.9.1 Unlock and mount script

```
#!/bin/sh -e
# This script mounts a user's confidential private folder
#
# Original by Michael Halcrow, IBM
# Extracted to a stand-alone script by Dustin Kirkland <kirkland@ubuntu.com>
# Modified by: Abdelhak Bougouffa <abougouffa@fedoraproject.org>
#
# This script:
#  * interactively prompts for a user's wrapping passphrase (defaults to their
#    login passphrase)
#  * checks it for validity
#  * unwraps a users mount passphrase with their supplied wrapping passphrase
#  * inserts the mount passphrase into the keyring
#  * and mounts a user's encrypted private folder

PRIVATE_DIR="Private"
PW_ATTEMPTS=3
MESSAGE=`gettext "Enter your login passphrase:"`

if [ -f $HOME/.ecryptfs/wrapping-independent ]
then
  # use a wrapping passphrase different from the login passphrase
  MESSAGE=`gettext "Enter your wrapping passphrase:"`
```

```
fi

WRAPPED_PASSPHRASE_FILE="$HOME/.ecryptfs/wrapped-passphrase"
MOUNT_PASSPHRASE_SIG_FILE="$HOME/.ecryptfs/$PRIVATE_DIR.sig"

# First, silently try to perform the mount, which would succeed if the appropriate
# key is available in the keyring
if /sbin/mount.ecryptfs_private >/dev/null 2>&1
then
  exit 0
fi

# Otherwise, interactively prompt for the user's password
if [ -f "$WRAPPED_PASSPHRASE_FILE" -a -f "$MOUNT_PASSPHRASE_SIG_FILE" ]
then
  tries=0

  while [ $tries -lt $PW_ATTEMPTS ]
  do
    LOGINPASS=`zenity --password --title "eCryptFS: $MESSAGE"`
    if [ $(wc -l < "$MOUNT_PASSPHRASE_SIG_FILE") = "1" ]
    then
      # No filename encryption; only insert fek
      if printf "%s\0" "$LOGINPASS" | ecryptfs-unwrap-passphrase "$WRAPPED_PASSPHRASE_FILE" - |
      ↪  ecryptfs-add-passphrase -
      then
        break
      else
        zenity --error --title "eCryptfs" --text "Error: Your passphrase is incorrect"
        tries=$(($tries + 1))
        continue
      fi
    else
      if printf "%s\0" "$LOGINPASS" | ecryptfs-insert-wrapped-passphrase-into-keyring
      ↪  "$WRAPPED_PASSPHRASE_FILE" -
      then
        break
      else
        zenity --error --title "eCryptfs" --text "Error: Your passphrase is incorrect"
        tries=$(($tries + 1))
        continue
      fi
    fi
  done

  if [ $tries -ge $PW_ATTEMPTS ]
  then
    zenity --error --title "eCryptfs" --text "Too many incorrect password attempts, exiting"
    exit 1
  fi

  /sbin/mount.ecryptfs_private
else
  zenity --error --title "eCryptfs" --text "Encrypted private directory is not setup properly"
  exit 1
fi

if grep -qs "$HOME/.Private $PWD ecryptfs " /proc/mounts 2>/dev/null; then
  zenity --info --title "eCryptfs" --text "Your private directory has been mounted."
fi

dolphin "$HOME/Private"
exit 0
```

10.9.2 Desktop integration

```
[Desktop Entry]
Type=Application
Version=1.0
Name=eCryptfs Unlock Private Directory
Icon=unlock
Exec=/home/hacko/.ecryptfs/ecryptfs-mount-private-gui
Terminal=False
```

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`.

```
# GDB early init file
# Abdelhak Bougouffa (c) 2022

# Disable showing the initial message
set startup-quietly
```

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.

```
# GDB init file
# Abdelhak Bougouffa (c) 2022

# Save history
set history save on
set history filename ~/.gdb_history
set history remove-duplicates 2048

# Set pretty print
set print pretty on

# This fixes the annoying ncurses TUI gliches and saves typing C-l each time to refresh the screen
define cc
  continue
  refresh
end

define nn
  next
  refresh
end

guile
<<guile-check-for-script>>
end
```

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.

```
(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 GnuPG

I add this to my `~/.gnupg/gpg-agent.conf`, to set the time-to-live to one day.

```
# Do not ask me about entered passwords for 24h (during the same session)
default-cache-ttl 86400
max-cache-ttl 86400
```

10.12 Packages

I like to use the BMC class, however, I do not like to manually install stuff in system directories, so I made an Arch Linux AUR package `bmc-git` for it.

I do use the `metropolis` theme for Beamer presentations, so I'm maintaining a package of it in the AUR too.

```
check_and_install_pkg () {
  PKG_NAME="$1"
  if ! pacman -Qiq ${PKG_NAME} &> /dev/null
```

```
  then
    echo "Package ${PKG_NAME} is missing, installing it using yay"
    yay -S ${PKG_NAME}
  fi
}

check_and_install_pkg bmc-git
check_and_install_pkg beamer-theme-metropolis
```

10.13 KDE Plasma

On KDE, there is a good support for HiDPI displays, however, an annoying thing is the small icons in some contexts (for example, a right click on desktop). This can be fixed by setting `PLASMA_USE_QT_SCALING=1`. KDE sources the files with `.sh` extension found on `~/.config/plasma-workspace/env`, so lets create our's.

```
# export PLASMA_USE_QT_SCALING=1
```