Doom Emacs Configuration

Emacs configuration for work and life!

Abdelhak Bougouffa*

September 16, 2022

Contents

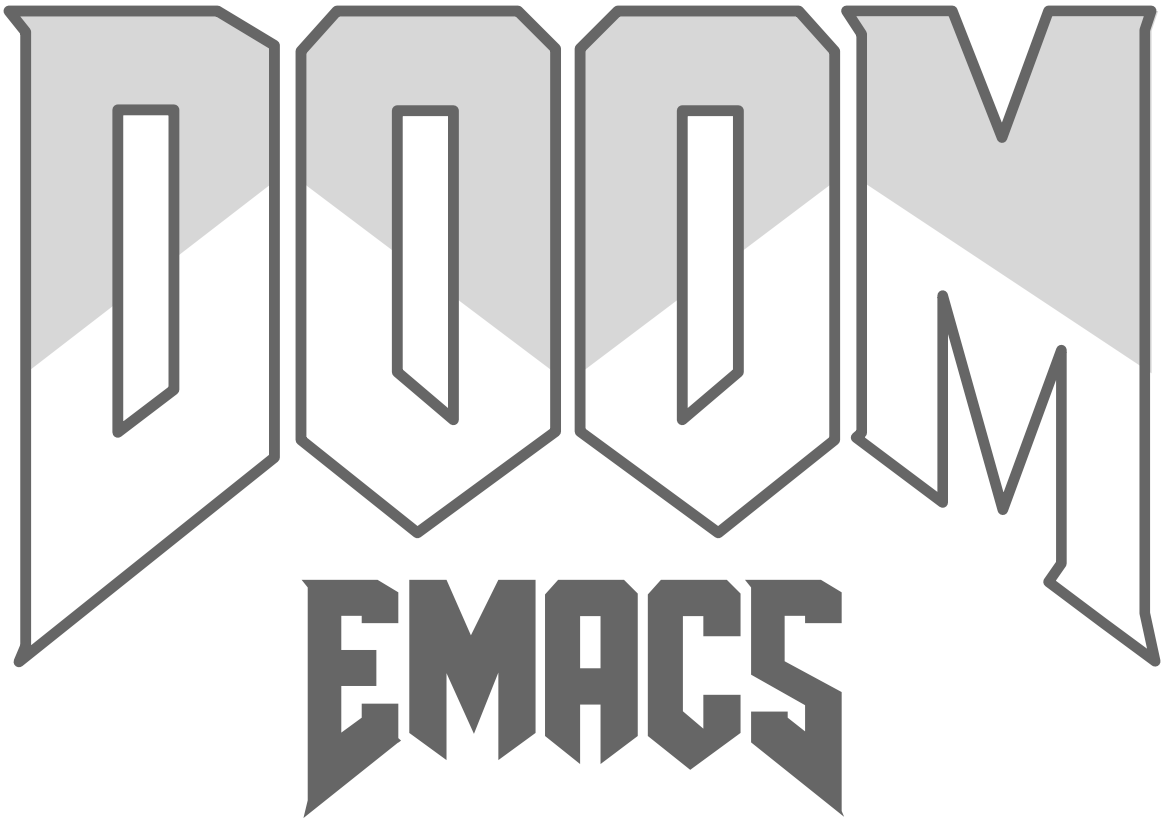
*abougouffa@fedoraproject.org

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 the following command; however, I don't recommend installing all of my dotfiles, try instead to adapt them or to copy some interesting chunks.

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

1.2 Emacs stuff

To use my Doom Emacs configuration, you need first to install Doom Emacs to `~/.config/emacs` or `.emacs.d`:

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

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

Until 12b3d20e, I was using Chemacs2 to manage multiple Emacs profiles. Since I'm using only Doom Emacs and Doom recently introduced a new feature to bootstrap other Emacs configs, so I switched to a plain Doom Emacs config.

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 My private Doom modules

I'm moving the big reusable configuration parts to separate modules. See the `.doom.d/modules/private` for the currently implemented modules.

2.2 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 -*- coding: utf-8-unix; 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 an initial comment to the `~/.zshrc` file.

```
# -*- mode: sh; -*-

# This file is automatically generated from my Org literate configuration.
# 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 -*- coding: utf-8-unix; lexical-binding: t; -*-

```

3.1.1 Useful functions

Here we define some useful functions, some of them are available via other packages like `cl-lib`, `dash.el` or `s.el`, but I don't like to load too much third party libraries, particulary in early stage, so let's define here.

```
;;; === Primitives ===

;; (+bool "someval") ;; ==> t
(defun +bool (val) (not (null val)))

;;; === Higher order functions ===

;; (+foldr (lambda (a b) (message "(%d + %d)" a b) (+ a b)) 0 '(1 2 3 4 5)) ;; ==> 15
;; (5 + 0) -> (4 + 5) -> (3 + 9) -> (2 + 12) --> (1 + 14)
(defun +foldr (fun acc seq)
  (if (null seq) acc
    (funcall fun (car seq) (+foldr fun acc (cdr seq)))))

;; (+foldl (lambda (a b) (message "(%d + %d)" a b) (+ a b)) 0 '(1 2 3 4 5)) ;; ==> 15
;; (0 + 1) -> (1 + 2) -> (3 + 3) -> (6 + 4) -> (10 + 5)
(defun +foldl (fun acc seq)
  (if (null seq) acc
    (+foldl fun (funcall fun acc (car seq)) (cdr seq))))

;; (+all '(83 88 t "txt")) ;; ==> t
(defun +all (seq)
  (+foldr (lambda (r l) (and r l)) t seq))

;; (+some '(nil nil "text" nil 2)) ;; ==> t
(defun +some (seq)
  (+bool (+foldr (lambda (r l) (or r l)) nil seq)))

;; (+filter 'stringp '("A" 2 "C" nil 3)) ;; ==> ("A" "C")
(defun +filter (fun seq)
  (when seq
    (let ((head (car seq))
          (tail (cdr seq)))
      (if (funcall fun head)
          (cons head (+filter fun tail))
        (+filter fun tail)))))

;; (+zip '(1 2 3 4) '(a b c d) '("A" "B" "C" "D")) ;; ==> ((1 a "A") (2 b "B") (3 c "C") (4 d "D"))
(defun +zip (&rest seqs)
  (if (null (car seqs)) nil
    (cons (mapcar #'car seqs)
          (apply #'+zip (mapcar #'cdr seqs)))))

;;; === Strings ===

;; (+str-join ", " '("foo" "10" "bar")) ;; ==> "foo, 10, bar"
(defun +str-join (sep seq)
  (+foldl (lambda (l r) (concat l sep r))
          (car seq) (cdr seq)))

;; (+str-split "foo, 10, bar" ", ") ;; ==> ("foo" "10" "bar")
(defun +str-split (str sep)
  (let ((s (string-search sep str)))
    (if s (cons (substring str 0 s)
                (+str-split (substring str (+ s (length sep))) sep))
      (list str))))

(defun +str-replace (old new s)
  "Replaces OLD with NEW in S."
  (replace-regexp-in-string (regexp-quote old) new s t t))

(defun +str-replace-all (replacements s)
  "REPLACEMENTS is a list of cons-cells. Each `car` is replaced with `cdr` in S."
  (replace-regexp-in-string (regexp-opt (mapcar 'car replacements))
                            (lambda (it) (cdr (assoc-string it replacements)))
                            s t t))
```

```
66
67  ;;; === Files, IO ===
68
69  (defun +file-mime-type (file)
70    "Get MIME type for FILE based on magic codes provided by the 'file' command.
71  Return a symbol of the MIME type, ex: `text/x-lisp', `text/plain',
72  `application/x-object', `application/octet-stream', etc."
73    (let ((mime-type (shell-command-to-string (format "file --brief --mime-type %s" file))))
74      (intern (string-trim-right mime-type))))
75
76  (defun +file-name-incremental (filename)
77    "Return an unique file name for FILENAME.
78  If \"file.ext\" exists, returns \"file-0.ext\"."
79    (let* ((ext (file-name-extension filename))
80           (dir (file-name-directory filename))
81           (file (file-name-base filename))
82           (filename-regex (concat "^" file "\\(?:-\\(?1:[[:digit:]]+\\)\\)?" (if ext (concat "\\." ext) "")))
83           (last-file (car (last (directory-files dir nil filename-regex))))
84           (last-file-num (when (and last-file (string-match filename-regex last-file)) (match-string 1
    ↪  last-file))))
85           (num (1+ (string-to-number (or last-file-num "-1"))))
86           (filename (file-name-concat dir (format "%s%s%s" file (if last-file (format "-%d" num) "") (if ext
    ↪  (concat "." ext) "")))))
87      filename))
88
89  (defun +file-read-to-string (filename)
90    "Return a string with the contents of FILENAME."
91    (when (and (file-exists-p filename) (not (file-directory-p filename)))
92      (with-temp-buffer
93        (insert-file-contents filename)
94        (buffer-string))))
95
96  ;;; === Systemd ===
97
98  (defun +systemd-running-p (service)
99    "Check if the systemd SERVICE is running."
100   (zerop (call-process "systemctl" nil nil nil "--user" "is-active" "--quiet" service ".service")))
101
102 (defun +systemd-command (service command &optional pre-fn post-fn)
103   "Call systemd with COMMAND and SERVICE."
104   (interactive)
105   (when pre-fn (funcall pre-fn))
106   (let ((success (zerop (call-process "systemctl" nil nil nil "--user" command service ".service"))))
107     (unless success
108       (message "[systemd]: Failed on calling '%s' on service %s.service." command service))
109     (when post-fn (funcall post-fn success))
110     success))
111
112 (defun +systemd-start (service &optional pre-fn post-fn)
113   "Start systemd SERVICE."
114   (interactive)
115   (+systemd-command service "start" pre-fn post-fn))
116
117 (defun +systemd-stop (service &optional pre-fn post-fn)
118   "Stops the systemd SERVICE."
119   (interactive)
120   (+systemd-command service "stop" pre-fn post-fn))
```

3.1.2 Fixes

```
1  ;; Fixes to apply early
2
3  (when (daemonp)
4    ;; When starting Emacs in daemon mode,
5    ;; I need to have a valid passphrase in the gpg-agent.
6    (let ((try-again 3)
7          unlocked)
```

```
    (while (not (or unlocked (zerop try-again)))
      (setq unlocked (zerop (shell-command "gpg -q --no-tty --logger-file /dev/null --batch -d ~/.authinfo.gpg
↪  > /dev/null" nil nil))
            try-again (1- try-again))
      (unless unlocked
        (message "GPG: failed to unlock, please try again (%d)" try-again)))
    (unless unlocked ;; Exit Emacs, systemd will restart it
      (kill-emacs 1))))
```

3.1.3 Check for external tools

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

```
(defconst EAF-DIR (expand-file-name "eaf/eaf-repo" doom-data-dir))
(defconst IS-LUCID (string-search "LUCID" system-configuration-features))
(defconst FRICAS-DIR "/usr/lib/fricas/emacs")

(defconst AG-P (executable-find "ag"))
(defconst EAF-P (and (not IS-LUCID) (file-directory-p EAF-DIR)))
(defconst MPD-P (+all (mapcar #'executable-find '("mpc" "mpd"))))
(defconst MPV-P (executable-find "mpv"))
(defconst REPO-P (executable-find "repo"))
(defconst FRICAS-P (and (executable-find "fricas") (file-directory-p FRICAS-DIR)))
(defconst MAXIMA-P (executable-find "maxima"))
(defconst TUNTOX-P (executable-find "tuntox"))
(defconst ROSBAG-P (executable-find "rosbag"))
(defconst ZOTERO-P (executable-find "zotero"))
(defconst CHEZMOI-P (executable-find "chezmoi"))
(defconst STUNNEL-P (executable-find "stunnel"))
(defconst ECRYPTFS-P (+all (mapcar #'executable-find '("ecryptfs-add-passphrase"
↪  "/sbin/mount.ecryptfs_private"))))
(defconst BITWARDEN-P (executable-find "bw"))
(defconst YOUTUBE-DL-P (+some (mapcar #'executable-find '("yt-dlp" "youtube-dl"))))
(defconst NETEXTENDER-P (and (executable-find "netExtender") (+all (mapcar #'file-exists-p
↪  '("~/.local/bin/netextender" "~/.ssh/sslvpn.gpg")))))
(defconst CLANG-FORMAT-P (executable-find "clang-format"))
(defconst LANGUAGETOOL-P (executable-find "languagetool"))
```

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 -*- coding: utf-8-unix; 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>>

  :private
  <<doom-private>>
)
```

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 +childframe)
```

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.

```
zen
deft
doom
hydra
hl-todo
ophints
modeline
nav-flash
workspaces
indent-guides
doom-dashboard
(treemacs +lsp)
(ligatures +extra)
(popup +all +defaults)
(emoji +ascii +github)
(window-select +numbers)
(vc-gutter +diff-hl +pretty)
```

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
```

3.2.7 Emacs builtin stuff (:emacs)

Beautify Emacs builtin packages.

```
vc
undo
(ibuffer +icons)
```

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).

```
term
vterm
shell
eshell
```

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)
```

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.

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

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.

```
qt
data
plantuml
emacs-lisp
common-lisp
(ess +lsp)
(yaml +lsp)
(markdown +grip)
(csharp +dotnet)
(racket +lsp +xp)
(lua +lsp +fennel)
(web +tree-sitter)
(latex +lsp +latexmk)
(cc +lsp +tree-sitter)
```

```
(sh +lsp +tree-sitter)
(json +lsp +tree-sitter)
(rust +lsp +tree-sitter)
(julia +lsp +tree-sitter)
(python +lsp +pyenv +pyright +tree-sitter)
(scheme +chez +mit +chicken +gauche +guile +chibi)
(org +dragndrop +gnuplot +jupyter +pandoc +noter +journal +hugo +present +pomodoro +roam2)
```

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.

```
(:if (executable-find "mu") (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`.

```
irc
rss
emms
calendar
everywhere
```

3.2.15 Private

```
(grammar +lsp)
(binary +disasm)
(dired-ng +icons +bindings)
```

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.

```
;; -*- coding: utf-8-unix; 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 Common variables

```
(defvar +my/lang-main          "en")
(defvar +my/lang-secondary     "fr")
(defvar +my/lang-mother-tongue "ar")

(defvar +my/biblio-libraries-list (list (expand-file-name "~/Zotero/library.bib")))
(defvar +my/biblio-storage-list   (list (expand-file-name "~/Zotero/storage/")))
(defvar +my/biblio-notes-path     (expand-file-name "~/PhD/bibliography/notes/"))
(defvar +my/biblio-styles-path    (expand-file-name "~/Zotero/styles/"))

;; Set it early, to avoid creating "~/org" at startup
(setq org-directory "~/Dropbox/Org")
```

4.3 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)

(after! epa
  (setq-default epa-file-encrypt-to '("F808A020A3E1AC37")))
```

4.4 Better defaults

4.4.1 File deletion

Delete files by moving them to trash.

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

4.4.2 Window

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

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

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))
```

4.4.3 Messages buffer

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

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

(defun +messages--auto-tail-a (&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-auto-tail-toggle ()
  "Auto tail the '*Messages*' buffer."
  (interactive)
  (if +messages--auto-tail-enabled
      (progn
        (advice-remove 'message '+messages--auto-tail-a)
        (setq +messages--auto-tail-enabled nil)
        (message "+messages-auto-tail: Disabled."))
    (advice-add 'message :after '+messages--auto-tail-a)
    (setq +messages--auto-tail-enabled t)
    (message "+messages-auto-tail: Enabled.")))
```

4.4.4 Undo and auto-save

```
(package! super-save
  :disable t
  :pin "d95d25615e69e7cc847641800c1886366336c97e")
```

Auto-save

```
(use-package! super-save
  :ensure t
  :config
  (setq auto-save-default t ;; nil to switch off the built-in `auto-save-mode', maybe leave it t to have a
↪  backup!
        super-save-exclude '(".gpg")
        super-save-remote-files nil
        super-save-auto-save-when-idle t)
  (super-save-mode +1))
```

```
(setq auto-save-default t) ;; enable built-in `auto-save-mode'
```

Undo Tweak `undo-fu` and other stuff from Doom's `:emacs undo`.

```
;; Increase undo history limits even more
(after! undo-fu
```

```
  (setq undo-limit        10000000     ;; 1MB   (default is 160kB, Doom's default is 400kB)
        undo-strong-limit 100000000    ;; 100MB (default is 240kB, Doom's default is 3MB)
        undo-outer-limit  1000000000)) ;; 1GB   (default is 24MB,  Doom's default is 48MB)

(after! evil
  (setq evil-want-fine-undo t)) ;; By default while in insert all changes are one big blob
```

```
(package! vundo
  :recipe (:host github
           :repo "casouri/vundo")
  :pin "16a09774ddfbd120d625cdd35fcf480e76e278bb")
```

Visual undo (vundo)

```
(use-package! vundo
  :defer t
  :init
  (defconst +vundo-unicode-symbols
   '((selected-node   . ?◉)
     (node            . ?○)
     (vertical-stem   . ?│)
     (branch          . ?├)
     (last-branch     . ?╰)
     (horizontal-stem . ?─)))

  (map! :leader
        (:prefix ("o")
         :desc "vundo" "v" #'vundo))

  :config
  (setq vundo-glyph-alist +vundo-unicode-symbols
        vundo-compact-display t
        vundo-window-max-height 6))
```

4.4.5 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)
```

4.4.6 Emacs sources

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

4.4.7 Frame

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.

4.4.8 Browsers

```
(setq browse-url-chrome-program "brave")
```

5 Emacs daemon

5.1 Initialization

```
(defun +daemon-startup ()
  ;; mu4e
  (when (require 'mu4e nil t)
    ;; Automatically start `mu4e' in background.
    (when (load! "mu-lock.el" (expand-file-name "email/mu4e/autoload" doom-modules-dir) t)
      (setq +mu4e-lock-greedy t
            +mu4e-lock-relaxed t)
      (when (+mu4e-lock-available t)
        ;; Check each 5m, if `mu4e' if closed, start it in background.
        (run-at-time nil ;; Launch now
                     (* 60 5) ;; Check each 5 minutes
                     (lambda ()
                       (when (and (not (mu4e-running-p)) (+mu4e-lock-available))
                         (mu4e--start)
                         (message "Started `mu4e' in background."))))))

  ;; RSS
  (when (require 'elfeed nil t)
    (run-at-time nil (* 2 60 60) #'elfeed-update))) ;; Check every 2h

(when (daemonp)
  ;; At daemon startup
  (add-hook 'emacs-startup-hook #'+daemon-startup)

  ;; After creating a new frame (via emacsclient)
  ;; Reload Doom's theme
  (add-hook 'server-after-make-frame-hook #'doom/reload-theme))
```

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)
    (let ((inhibit-message t))
      (recentf-save-list)
      (savehist-save))))
```

6 Package configuration

6.1 User interface

6.1.1 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 "Iosevka Fixed Curly Slab" :size 20)
      doom-big-font (font-spec :family "Iosevka Fixed Curly Slab" :size 30 :weight 'light)
      doom-variable-pitch-font (font-spec :family "Iosevka Fixed Curly Slab")
      doom-unicode-font (font-spec :family "JuliaMono")
      doom-serif-font (font-spec :family "Iosevka Fixed Curly Slab" :weight 'light))
```

6.1.2 Theme

Doom Set Doom's theme, some good choices:

- `doom-one` (Atom like)
- `doom-vibrant` (More vibrant version of `doom-one`)
- `doom-one-light` (Atom like)
- `doom-dark+` (VS Code like)
- `doom-xcode` (XCode like)
- `doom-material`
- `doom-material-dark`

- doom-palenight
- doom-ayu-mirage
- doom-monokai-pro
- doom-tomorrow-day
- doom-tomorrow-night

```
(setq doom-theme 'doom-one-light)
;; (setq doom-theme 'doom-tomorrow-day)
(remove-hook 'window-setup-hook #'doom-init-theme-h)
(add-hook 'after-init-hook #'doom-init-theme-h 'append)
```

```
(package! modus-themes :pin "e4ee971f53d5c687de7bfbcf9fd1522bb42be902")
```

Modus

```
(use-package! modus-themes
  :init
  (setq modus-themes-hl-line '(accented intense)
        modus-themes-subtle-line-numbers t
        modus-themes-region '(bg-only no-extend) ;; accented
        modus-themes-variable-pitch-ui nil
        modus-themes-fringes 'subtle
        modus-themes-diffs nil
        modus-themes-italic-constructs t
        modus-themes-bold-constructs t
        modus-themes-intense-mouseovers t
        modus-themes-paren-match '(bold intense)
        modus-themes-syntax '(green-strings)
        modus-themes-links '(neutral-underline background)
        modus-themes-mode-line '(borderless padded)
        modus-themes-tabs-accented nil ;; default
        modus-themes-completions
        '((matches . (extrabold intense accented))
          (selection . (semibold accented intense))
          (popup . (accented)))
        modus-themes-headings '((1 . (rainbow 1.4))
                                (2 . (rainbow 1.3))
                                (3 . (rainbow 1.2))
                                (4 . (rainbow bold 1.1))
                                (t . (rainbow bold)))
        modus-themes-org-blocks 'gray-background
        modus-themes-org-agenda
        '((header-block . (semibold 1.4))
          (header-date . (workaholic bold-today 1.2))
          (event . (accented italic varied))
          (scheduled . rainbow)
          (habit . traffic-light))
        modus-themes-markup '(intense background)
        modus-themes-mail-citations 'intense
        modus-themes-lang-checkers '(background))

  (defun +modus-themes-tweak-packages ()
    (modus-themes-with-colors
      (set-face-attribute 'cursor nil :background (modus-themes-color 'blue))
      (set-face-attribute 'font-lock-type-face nil :foreground (modus-themes-color 'magenta-alt))
      (custom-set-faces
       ;; Tweak `evil-mc-mode'
       `(evil-mc-cursor-default-face ((,class :background ,magenta-intense-bg)))
```

```
        ;; Tweak `git-gutter-mode'
        `(git-gutter-fr:added ((,class :foreground ,green-fringe-bg)))
        `(git-gutter-fr:deleted ((,class :foreground ,red-fringe-bg)))
        `(git-gutter-fr:modified ((,class :foreground ,yellow-fringe-bg)))
        ;; Tweak `doom-modeline'
        `(doom-modeline-evil-normal-state ((,class :foreground ,green-alt-other)))
        `(doom-modeline-evil-insert-state ((,class :foreground ,red-alt-other)))
        `(doom-modeline-evil-visual-state ((,class :foreground ,magenta-alt)))
        `(doom-modeline-evil-operator-state ((,class :foreground ,blue-alt)))
        `(doom-modeline-evil-motion-state ((,class :foreground ,blue-alt-other)))
        `(doom-modeline-evil-replace-state ((,class :foreground ,yellow-alt)))
        ;; Tweak `diff-hl-mode'
        `(diff-hl-insert ((,class :foreground ,green-fringe-bg)))
        `(diff-hl-delete ((,class :foreground ,red-fringe-bg)))
        `(diff-hl-change ((,class :foreground ,yellow-fringe-bg)))
        ;; Tweak `solaire-mode'
        `(solaire-default-face ((,class :inherit default :background ,bg-alt :foreground ,fg-dim)))
        `(solaire-line-number-face ((,class :inherit solaire-default-face :foreground ,fg-unfocused)))
        `(solaire-hl-line-face ((,class :background ,bg-active)))
        `(solaire-org-hide-face ((,class :background ,bg-alt :foreground ,bg-alt)))
        ;; Tweak `display-fill-column-indicator-mode'
        `(fill-column-indicator ((,class :height 0.3 :background ,bg-inactive :foreground ,bg-inactive)))
        ;; Tweak `mmm-mode'
        `(mmm-cleanup-submode-face ((,class :background ,yellow-refine-bg)))
        `(mmm-code-submode-face ((,class :background ,bg-active)))
        `(mmm-comment-submode-face ((,class :background ,blue-refine-bg)))
        `(mmm-declaration-submode-face ((,class :background ,cyan-refine-bg)))
        `(mmm-default-submode-face ((,class :background ,bg-alt)))
        `(mmm-init-submode-face ((,class :background ,magenta-refine-bg)))
        `(mmm-output-submode-face ((,class :background ,red-refine-bg)))
        `(mmm-special-submode-face ((,class :background ,green-refine-bg))))))

  (add-hook 'modus-themes-after-load-theme-hook #'+modus-themes-tweak-packages)

  :config
  (modus-themes-load-operandi)
  (map! :leader
        :prefix "t" ;; toggle
        :desc "Toggle Modus theme" "m" #'modus-themes-toggle))
```

```
(package! lambda-themes
  :disable t
  :recipe (:host github
           :repo "Lambda-Emacs/lambda-themes")
  :pin "d95d25615e69e7cc847641800c1886366336c97e")

(package! lambda-line
  :disable t
  :recipe (:host github
           :repo "Lambda-Emacs/lambda-line"))
```

Lambda themes

```
(use-package! lambda-themes
  :init
  (setq lambda-themes-set-italic-comments t
        lambda-themes-set-italic-keywords t
        lambda-themes-set-variable-pitch t
        lambda-themes-set-evil-cursors t)
  :config
  ;; load preferred theme
  (load-theme 'lambda-light-faded t))

```

```
(use-package! lambda-line
  :custom
  (lambda-line-position 'top) ;; Set position of status-line
  (lambda-line-abbrev t) ;; abbreviate major modes
  (lambda-line-hspace "  ")  ;; add some cushion
  (lambda-line-prefix t) ;; use a prefix symbol
  (lambda-line-prefix-padding nil) ;; no extra space for prefix
  (lambda-line-status-invert nil)  ;; no invert colors
  (lambda-line-gui-ro-symbol  " ") ;; symbols
  (lambda-line-gui-mod-symbol " ")
  (lambda-line-gui-rw-symbol  " ")
  (lambda-line-space-top +.50)  ;; padding on top and bottom of line
  (lambda-line-space-bottom -.50)
  (lambda-line-symbol-position 0.1) ;; adjust the vertical placement of symbol
  :config
  ;; activate lambda-line
  (lambda-line-mode)
  ;; set divider line in footer
  (when (eq lambda-line-position 'top)
    (setq-default mode-line-format (list "%_"))
    (setq mode-line-format (list "%_"))))
```

```
(package! spacemacs-theme :pin "e04d1f21107a1565861625209bb9c46a7aa43cc5")
```

Spacemacs theme

6.1.3 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

  ;; Add padding to the right
  (doom-modeline-def-modeline 'main
    '(bar workspace-name window-number modals matches follow buffer-info remote-host buffer-position word-count
    ↪  parrot selection-info)
    '(objed-state misc-info persp-name battery grip irc mu4e gnus github debug repl lsp minor-modes input-method
    ↪  indent-info buffer-encoding major-mode process vcs checker "   ")))
```

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))))
```

```
(after! doom-modeline
  (setq doom-modeline-bar-width 4
        doom-modeline-mu4e t
        doom-modeline-major-mode-icon t
```

```
        doom-modeline-major-mode-color-icon t
        doom-modeline-buffer-file-name-style 'truncate-upto-project))
```

Mode line customization

6.1.4 Set transparency

```
;; NOTE: Not tangled
(set-frame-parameter (selected-frame) 'alpha '(85 100))
(add-to-list 'default-frame-alist '(alpha 97 100))
```

6.1.5 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
doom-emacs-gray.svg

```
(setq fancy-splash-image (expand-file-name "assets/doom-emacs-gray.svg" doom-user-dir))
```

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

Dashboard

6.1.6 Which key

Make `which-key` popup faster.

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

I've stolen this chunk (like many others) from tecosaur's config, it helps to replace the `evil-` prefix with a unicode symbol, making `which-key`'s candidate list less verbose.

```
(setq which-key-allow-multiple-replacements t)

(after! which-key
  (pushnew! which-key-replacement-alist
            '(("" . "\\`+?evil[-:]?\\(?:a-\\)?\\(.*\\)") . (nil . " ·\\1"))
            '(("\\`g s" . "\\`evilem--?motion-\\(.*\\)") . (nil . " ·\\1"))))
```

6.1.7 Window title

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

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

6.1.8 SVG tag and `svg-lib`

```
(package! svg-tag-mode :pin "efd22edf650fb25e665269ba9fed7ccad0771a2f")
```

```
(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 10
               :height 0.8
               :padding 0
               :margin 0))))
          ("\\(:[A-Za-z0-9]+:\\)$" .
           ((lambda (tag)
              (svg-tag-make
               tag
               :beg 1
               :end -1
               :font-family "Roboto Mono"
               :font-size 10
               :height 0.8
               :padding 0
               :margin 0)))))))
```

```
(after! svg-lib
  ;; Set `svg-lib' cache directory
  (setq svg-lib-icons-dir (expand-file-name "svg-lib" doom-data-dir)))
```

6.1.9 Focus

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

```
(package! focus :pin "9dd85fc474bbc1ebf22c287752c960394fcd465a")
```

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

6.1.10 Scrolling

```
(package! good-scroll
  :disable EMACS29+
  :pin "a7ffd5c0e5935cebd545a0570f64949077f71ee3")
```

```
(use-package! good-scroll
  :unless EMACS29+
  :config (good-scroll-mode 1))

(when EMACS29+
  (pixel-scroll-precision-mode 1))

(setq hscroll-step 1
      hscroll-margin 0
      scroll-step 1
      scroll-margin 0
      scroll-conservatively 101
      scroll-up-aggressively 0.01
      scroll-down-aggressively 0.01
      scroll-preserve-screen-position 'always
      auto-window-vscroll nil
      fast-but-imprecise-scrolling nil)
```

6.1.11 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))))
```

6.1.12 Tabs

```
(after! centaur-tabs
  ;; For some reason, setting `centaur-tabs-set-bar' this to `right'
  ;; instead of Doom's default `left', fixes this issue with Emacs daemon:
  ;; https://github.com/doomemacs/doomemacs/issues/6647#issuecomment-1229365473
  (setq centaur-tabs-set-bar 'right
        centaur-tabs-gray-out-icons 'buffer
        centaur-tabs-set-modified-marker t
        centaur-tabs-close-button " "
        centaur-tabs-modified-marker " "))
```

6.1.13 Zen (writeroom) mode

```
(after! writeroom-mode
  ;; Show mode line
  (setq writeroom-mode-line t)

  ;; Disable line numbers
  (add-hook! 'writeroom-mode-enable-hook
```

```
    (when (bound-and-true-p display-line-numbers-mode)
      (setq-local +line-num--was-activate-p display-line-numbers-type)
      (display-line-numbers-mode -1)))

  (add-hook! 'writeroom-mode-disable-hook
    (when (bound-and-true-p +line-num--was-activate-p)
      (display-line-numbers-mode +line-num--was-activate-p)))

  (after! org
    ;; Increase latex previews scale in Zen mode
    (add-hook! 'writeroom-mode-enable-hook (+org-format-latex-set-scale 2.0))
    (add-hook! 'writeroom-mode-disable-hook (+org-format-latex-set-scale 1.4)))

  (after! blamer
    ;; Disable blamer in zen (writeroom) mode
    (add-hook! 'writeroom-mode-enable-hook
      (when (bound-and-true-p blamer-mode)
        (setq +blamer-mode--was-active-p t)
        (blamer-mode -1)))
    (add-hook! 'writeroom-mode-disable-hook
      (when (bound-and-true-p +blamer-mode--was-active-p)
        (blamer-mode 1)))))
```

6.2 Highlight indent guides

```
(after! highlight-indent-guides
  (setq highlight-indent-guides-character ?|
        highlight-indent-guides-responsive 'top))
```

6.3 Editing

6.3.1 File templates

For some file types, we can overwrite the defaults in the snippets' directory.

```
(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)
```

6.3.2 Scratch buffer

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

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

6.3.3 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)

;; Enable horizontal scrolling with the second mouse wheel or the touchpad
(setq mouse-wheel-tilt-scroll t
      mouse-wheel-progressive-speed nil)
```

6.3.4 Very large files

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

```
(package! vlf :pin "cc02f2533782d6b9b628cec7e2dcf25b2d05a27c")
```

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)
```

6.3.5 Evil

```
(after! evil
  ;; This fixes https://github.com/doomemacs/doomemacs/issues/6478
  ;; Ref: https://github.com/emacs-evil/evil/issues/1630
  (evil-select-search-module 'evil-search-module 'isearch)

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

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

6.3.6 Aggressive indent

```
(package! aggressive-indent :pin "70b3f0add29faff41e480e82930a231d88ee9ca7")
```

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

6.3.7 YASnippet

Nested snippets are good, enable that.

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

6.4 Completion & IDE

6.4.1 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))
```

```
(after! company-box
  (when (daemonp)
    (defun +company-box--reload-icons-h ()
      (setq company-box-icons-all-the-icons
            (let ((all-the-icons-scale-factor 0.8))
              `((Unknown       . ,(all-the-icons-faicon   "code"                :face 'all-the-icons-purple))
                (Text          . ,(all-the-icons-material "text_fields"         :face 'all-the-icons-green))
                (Method        . ,(all-the-icons-faicon   "cube"                :face 'all-the-icons-red))
                (Function      . ,(all-the-icons-faicon   "cube"                :face 'all-the-icons-red))
                (Constructor   . ,(all-the-icons-faicon   "cube"                :face 'all-the-icons-red))
                (Field         . ,(all-the-icons-faicon   "tag"                 :face 'all-the-icons-red))
                (Variable      . ,(all-the-icons-material "adjust"              :face 'all-the-icons-blue))
                (Class         . ,(all-the-icons-material "class"               :face 'all-the-icons-red))
                (Interface     . ,(all-the-icons-material "tune"                :face 'all-the-icons-red))
                (Module        . ,(all-the-icons-faicon   "cubes"               :face 'all-the-icons-red))
                (Property      . ,(all-the-icons-faicon   "wrench"              :face 'all-the-icons-red))
                (Unit          . ,(all-the-icons-material "straighten"          :face 'all-the-icons-red))
                (Value         . ,(all-the-icons-material "filter_1"            :face 'all-the-icons-red))
                (Enum          . ,(all-the-icons-material "plus_one"            :face 'all-the-icons-red))
                (Keyword       . ,(all-the-icons-material "filter_center_focus" :face 'all-the-icons-red-alt))
                (Snippet       . ,(all-the-icons-faicon   "expand"              :face 'all-the-icons-red))
                (Color         . ,(all-the-icons-material "colorize"            :face 'all-the-icons-red))
                (File          . ,(all-the-icons-material "insert_drive_file"   :face 'all-the-icons-red))
                (Reference     . ,(all-the-icons-material "collections_bookmark" :face 'all-the-icons-red))
                (Folder        . ,(all-the-icons-material "folder"              :face 'all-the-icons-red-alt))
                (EnumMember    . ,(all-the-icons-material "people"              :face 'all-the-icons-red))
                (Constant      . ,(all-the-icons-material "pause_circle_filled" :face 'all-the-icons-red))
                (Struct        . ,(all-the-icons-material "list"                :face 'all-the-icons-red))
                (Event         . ,(all-the-icons-material "event"               :face 'all-the-icons-red))
                (Operator      . ,(all-the-icons-material "control_point"       :face 'all-the-icons-red))
                (TypeParameter . ,(all-the-icons-material "class"               :face 'all-the-icons-red))
                (Template      . ,(all-the-icons-material "settings_ethernet"   :face 'all-the-icons-green))
                (ElispFunction . ,(all-the-icons-faicon   "cube"                :face 'all-the-icons-red))
                (ElispVariable . ,(all-the-icons-material "adjust"              :face 'all-the-icons-blue))
                (ElispFeature  . ,(all-the-icons-material "stars"               :face 'all-the-icons-orange))
                (ElispFace     . ,(all-the-icons-material "format_paint"        :face 'all-the-icons-pink))))))

    ;; Replace Doom defined icons with mine
    (when (memq #'+company-box--load-all-the-icons server-after-make-frame-hook)
      (remove-hook 'server-after-make-frame-hook #'+company-box--load-all-the-icons))
    (add-hook 'server-after-make-frame-hook #'+company-box--reload-icons-h)))
```

Tweak `company-box`

6.4.2 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))
```

6.4.3 Projectile

Doom Emacs defined a function `(doom-project-ignored-p path)` and uses it with `projectile-ignored-project-function`. So we will create a wrapper function which calls Doom's one, with an extra check.

```
;; Run `M-x projectile-discover-projects-in-search-path' to reload paths from this variable
(setq projectile-project-search-path
      '("~/PhD/papers"
        "~/PhD/workspace"
        "~/PhD/workspace-no"
        "~/PhD/workspace-no/ez-wheel/swd-starter-kit-repo"
        ("~/Projects/foss" . 2))) ;; ("dir" . depth)

(setq projectile-ignored-projects
      '("/tmp"
        "~/"
        "~/.cache"
        "~/.doom.d"
```

```
        "~/.emacs.d/.local/straight/repos/"))

(setq +projectile-ignored-roots
      '("~/.cache"
        ;; No need for this one, as `doom-project-ignored-p' checks for files in `doom-local-dir'
        "~/.emacs.d/.local/straight/"))

(defun +projectile-ignored-project-function (filepath)
  "Return t if FILEPATH is within any of `+projectile-ignored-roots'"
  (require 'cl-lib)
  (or (doom-project-ignored-p filepath) ;; Used by default by doom with `projectile-ignored-project-function'
      (cl-some (lambda (root) (file-in-directory-p (expand-file-name filepath) (expand-file-name root)))
          +projectile-ignored-roots)))

(setq projectile-ignored-project-function #'+projectile-ignored-project-function)
```

6.4.4 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 +
```

6.4.5 Eros-eval

This makes the result of evals slightly prettier.

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

6.4.6 `dir-locals.el`

Reload `dir-locals.el` variables after modification. Taken from this answer.

```
(defun +dir-locals-reload-for-current-buffer ()
  "reload dir locals for the current buffer"
  (interactive)
  (let ((enable-local-variables :all))
    (hack-dir-local-variables-non-file-buffer)))

(defun +dir-locals-reload-for-all-buffers-in-this-directory ()
  "For every buffer with the same `default-directory` as the
current buffer's, reload dir-locals."
  (interactive)
  (let ((dir default-directory))
    (dolist (buffer (buffer-list))
      (with-current-buffer buffer
        (when (equal default-directory dir)
          (+dir-locals-reload-for-current-buffer))))))

(defun +dir-locals-enable-autoreload ()
  (when (and (buffer-file-name)
             (equal dir-locals-file (file-name-nondirectory (buffer-file-name))))
    (message "Dir-locals will be reloaded after saving.")
    (add-hook 'after-save-hook '+dir-locals-reload-for-all-buffers-in-this-directory nil t)))

(add-hook! '(emacs-lisp-mode-hook lisp-data-mode-hook) #'+dir-locals-enable-autoreload)
```

6.4.7 Language Server Protocol

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")))
```

LSP mode

```
;; (unpin! lsp-mode)
```

Unpin package

Tweaks

1. Performance Use `plist` instead of hash table, LSP mode needs to be reinstalled after setting this environment variable (see).

```
export LSP_USE_PLISTS=true
```

```
(after! lsp-mode
  (setq lsp-idle-delay 1.0
        lsp-log-io nil
        gc-cons-threshold (* 1024 1024 100))) ;; 100MiB
```

2. Features & UI LSP mode provides a set of configurable UI stuff. By default, Doom Emacs disables some UI components; however, I like to enable some less intrusive, more useful UI stuff.

```
(after! lsp-mode
  (setq lsp-lens-enable t
        lsp-semantic-tokens-enable t ;; hide unreachable ifdefs
        lsp-enable-symbol-highlighting t
        lsp-headerline-breadcrumb-enable nil
        ;; LSP UI related tweaks
        lsp-ui-sideline-enable nil
        lsp-ui-sideline-show-hover nil
        lsp-ui-sideline-show-symbol nil
        lsp-ui-sideline-show-diagnostics nil
        lsp-ui-sideline-show-code-actions nil))
```

```
(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 1))
```

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`

1. Python

```
(after! tramp
  (when (require 'lsp-mode nil t)
    ;; (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))))
```

2. C/C++ with `ccls`

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

3. C/C++ with `clangd`

```
(after! tramp
  (when (require 'lsp-mode nil t)

    (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))))
```

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 let's use `rust_hdl` instead.

```
(use-package! vhdl-mode
  :when (and (modulep! :tools lsp) (not (modulep! :tools lsp +eglot)))
  :hook (vhdl-mode . #'+lsp-vhdl-ls-load)
  :init
  (defun +lsp-vhdl-ls-load ()
    (interactive)
    (lsp t)
    (flycheck-mode t))

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

```
(package! lsp-sonarlint
  :disable t :pin "d95d25615e69e7cc847641800c1886366336c97e")
```

SonarLint

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

6.4.8 Cppcheck

Check for everything!

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

6.4.9 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 t ; (not (modulep! :tools lsp +eglot)) ; Enable only if (lsp +eglot) is used
  :pin "d95d25615e69e7cc847641800c1886366336c97e"
  :recipe (:host github
           :repo "juanjosegarciaripoll/project-cmake"))
```

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

6.4.10 Clang-format

```
(package! clang-format :pin "e48ff8ae18dc7ab6118c1f6752deb48cb1fc83ac")
```

```
(use-package! clang-format
  :when CLANG-FORMAT-P
  :commands (clang-format-region))
```

6.4.11 Auto-include C++ headers

```
(package! cpp-auto-include
  :recipe (:host github
           :repo "emacsorphanage/cpp-auto-include")
  :pin "0ce829f27d466c083e78b9fe210dcfa61fb417f4")
```

```
(use-package! cpp-auto-include
  :commands cpp-auto-include)
```

6.4.12 C/C++ preprocessor conditions

In LSP mode, I configure `lsp-semantic-tokens-enable` to enable fading unreachable `#ifdef` blocks, in case LSP is disabled, there is a similar built-in mode for Emacs to do so. However, for a fully satisfying experience, it needs more work to take into account macros defined at compile time (using `compile_commands.json` for example).

```
(unless (modulep! :lang cc +lsp) ;; Disable if LSP for C/C++ is enabled
  (use-package! hideif
    :hook (c-mode . hide-ifdef-mode)
    :hook (c++-mode . hide-ifdef-mode)
    :init
    (setq hide-ifdef-shadow t
          hide-ifdef-initially t)))
```

6.4.13 Erefactor

```
(package! erefactor
  :recipe (:host github
           :repo "mhayashi1120/Emacs-erefactor")
  :pin "bfe27a1b8c7cac0fe054e76113e941efa3775fe8")
```

```
(use-package! erefactor
  :defer t)
```

6.4.14 Lorem ipsum

```
(package! emacs-lorem-ipsum
  :recipe (:host github
           :repo "jschaf/emacs-lorem-ipsum")
  :pin "da75c155da327c7a7aedb80f5cfe409984787049")
```

```
(use-package! lorem-ipsum
  :commands (lorem-ipsum-insert-sentences
             lorem-ipsum-insert-paragraphs
             lorem-ipsum-insert-list))
```

6.4.15 Coverage test

```
(package! cov :pin "cd3e1995c596cc227124db9537792d8329ffb696")
```

6.5 Debugging

6.5.1 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
  ;; Set latest versions
  (setq dap-cpptools-extension-version "1.11.5")
  (require 'dap-cpptools)

  (setq dap-codelldb-extension-version "1.7.4")
  (require 'dap-codelldb)

  (setq dap-gdb-lldb-extension-version "0.26.0")
  (require 'dap-gdb-lldb)

  ;; More minimal UI
  (setq dap-auto-configure-features '(breakpoints locals expressions 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)))

  ;; A workaround to correctly show breakpoints
  ;; from: https://github.com/emacs-lsp/dap-mode/issues/374#issuecomment-1140399819
  (add-hook! +dap-running-session-mode
    (set-window-buffer nil (current-buffer))))
```

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 (modulep! :tools debugger +lsp)
       :prefix ("d" . "debugger")
       :desc "Clear last DAP session" "c" #'+debugger/clear-last-session))
```

6.5.2 RealGUD

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 better than using pure GDB, it makes debugging extremely flexible. Let's 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 a debug session).

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

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

```

```
  (defun +realgud:cmd-reverse-next (arg)
    "Reverse next"
    (interactive "p")
    (realgud-command "reverse-next"))

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

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

  (defun +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" +realgud:cmd-reverse-next)
    ("ri" +realgud:cmd-reverse-step)
    ("ro" +realgud:cmd-reverse-finish)
    ("rc" +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" +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))

  (map! :leader :prefix ("l" . "custom")
        (:when (modulep! :tools debugger)
         :prefix ("d" . "debugger")
         :desc "RealGUD hydra" "h" #'+debugger/realgud:gdb-hydra)))
```

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 `+realgud-debug-config`, 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
  (defun +debugger/rr-replay ()
    "Launch `rr replay'."
    (interactive)
    (realgud:gdb (+str-replace "gdb" "rr replay" realgud:gdb-command-name)))

  (defun +debugger/rr-record ()
    "Launch `rr record' with parameters from launch.json or `+launch-json-debug-config'."
    (interactive)
    (let* ((conf (launch-json--config-choice))
           (args (launch-json--substite-special-vars (plist-get conf :program) (plist-get conf :args))))
      (unless (make-process :name "rr-record"
                            :buffer "*rr record*"
                            :command (append '("rr" "record") args))
        (message "Cannot start the 'rr record' process"))))

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

```
(package! realgud-lldb :pin "19a2c0a8b228af543338f3a8e51141a9e23484a5")
(package! realgud-ipdb :pin "f18f907aa4ddd3e59dc19ca296d4ee2dc5e436b0")

(package! realgud-trepan-xpy
  :recipe (:host github
           :repo "realgud/trepan-xpy")
  :pin "f53fea61a86226dcf5222b2814a549c8f8b8d5a9")

(package! realgud-maxima
  :recipe (:host github
           :repo "realgud/realgud-maxima")
  :pin "74d1615be9105d7f8a8d6d0b9f6d7a91638def11")
```

Additional debuggers for RealGUD

6.5.3 GDB

Emacs GDB *a.k.a.* `gdb-mi` 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-gdb`.

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
  :disable t
  :recipe (:host github
           :repo "weirdNox/emacs-gdb"
           :files ("*.el" "*.c" "*.h" "Makefile"))
  :pin "d95d25615e69e7cc847641800c1886366336c97e")
```

```
(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

6.5.4 WIP `launch.json` support for GUD and RealGUD

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 GUD/RealGUD though, I didn't find any ready-to-use feature like this. So let's code it!

I like to define a variable named `+launch-json-debug-config` to use as a fallback, if no `launch.json` file is present.

```
;; A variable which to be used in .dir-locals.el, formatted as a list of plists;
;; '((:program "..." :args ("args1" "arg2" ...)))
(defvar +launch-json-debug-config nil)
```

This variable should have the same structure of a `launch.json` file, in Elisp, it should be a *list of plists*, each `plist` represents a configuration, this variable can be set in `.dir-locals.el` for example.

The configuration plists supports some of `launch.json` parameters, including:

- `:name` a description of the debug configuration;
- `:type` which debugger to use;
- `:program` the path of the debuggee program;
- `:args` a list of string arguments to pass to the debuggee program.

The variable `+launch-json-debug-config` can be set in a per-project basis thanks to `.dir-locals.el`, something like this:

```
;; Example entry in .dir-locals.el
((nil . ((+launch-json-debug-config . '((:name "Debug my_prog with csv data"
                                         :type "realgud:gdb" ;; One config with `realgud:gdb'
                                         :program "${workspaceFolder}/build/bin/my_prog"
                                         :args ("--in_file=${workspaceFolder}/some/file.csv"
                                                "--some-parameter" "-a" "-b"
                                                "--out_file=/tmp/some_randome_file"
```

```
                                                "-a"))
                                          (:name "Debug my_prog with options '-a' and '-b'"
                                           :type "gdb-mi" ;; Another config with `gdb-mi'
                                           :program "${workspaceFolder}/build/bin/my_prog"
                                           :args ("-a" "-b")))))))
```

The list of implemented special variables are listed in the table below, they have been defined as specified in VS Code.

Variable	Example
`userHome`	`/home/username`
`workspaceFolder`	`/home/username/your-project`
`workspaceFolderBasename`	`your-project`
`file`	`/home/username/your-project/folder/file.cc`
`fileWorkspaceFolder`	`/home/username/your-project`
`relativeFile`	`folder/file.cc`
`relativeFileDirname`	`folder`
`fileBasename`	`file.cc`
`fileBasenameNoExtension`	`file`
`fileDirname`	`/home/username/your-project/folder`
`fileExtname`	`.cc`
`lineNumber`	Line number of the cursor
`selectedText`	Text selected in your code editor
`pathSeparator`	Returns `/` on *nix, and `\` on Windows

If a `launch.json` file is detected in the project directory, it gets read and searches for a configuration for the `realgud:gdb` debugger. So you need to have a section with type `realgud:gdb`. This is an example of a valid `launch.json` file.

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Emacs::RealGUD:GDB (view_trajectory)",
      "type": "realgud:gdb",
      "request": "launch",
      "dap-compilation": "cmake --build build/debug -- -j 8",
      "dap-compilation-dir": "${workspaceFolder}",
      "program": "${workspaceFolder}/build/debug/bin/view_trajectory",
      "args": [
        "htraj=${workspaceFolder}/data/seq1/h_poses.csv",
        "traj=${workspaceFolder}/data/seq1/poses.csv"
      ],
      "stopAtEntry": false,
      "cwd": "${workspaceFolder}",
      "environment": [],
      "externalConsole": false
    }
  ]
}
```

The example above defines several parameters, however, only `type`, `program` and `args` are used at the moment.

```
(defvar launch-json--gud-debugger-regex
  (rx (seq bol (group-n 1 (or "gdb" "gud-gdb" "perldb" "pdb" "jdb" "guiler" "dbx" "sdb" "xdb") eol))))

(defvar launch-json--realgud-debugger-regex
  (rx (seq bol (or (seq "realgud:" (group-n 1 (or "gdb" "pdb"
                                                  "bashdb"  "kshdb" "zshd"
                                                  "perldb" "rdebug" "remake"
                                                  "trepan" "trepan2" "trepan3k" "trepanjs" "trepan.pl")))
```

```
                    (seq "realgud-" (group-n 1 (or "gub")))
                    ;; Additional debuggers
                    (seq "realgud:" (group-n 1 (or "xdebug" "pry" "jdb" "ipdb" "trepan-xpy" "trepan-ni"
↪  "node-inspect")))
                    ;; `realgud-lldb' defines the debug command as `realgud--lldb',
                    ;; We accept both `realgud:lldb' and `realgud--lldb' in the config
                    (seq "realgud" (or ":" "--") (group-n 1 (or "lldb")))) eol)))

;; Define aliases for realgud-lldb
(with-eval-after-load 'realgud-lldb
  (defalias 'realgud:lldb 'realgud--lldb)
  (defalias 'realgud:lldb-command-name 'realgud--lldb-command-name))

;; Define aliases for realgud-ipdb
(with-eval-after-load 'realgud-ipdb
  (defalias 'realgud:ipdb-command-name 'realgud--ipdb-command-name))

(defvar launch-json--last-config nil)

(defun launch-json-last-config-clear ()
  (interactive)
  (setq-local launch-json--last-config nil))

(defun launch-json--substite-special-vars (program &optional args)
  "Substitue variables in PROGRAM and ARGS.
Return a list, in which processed PROGRAM is the first element, followed by ARGS."
  (let* ((curr-file (ignore-errors (expand-file-name (buffer-file-name))))
         (ws-root (string-trim-right
                   (expand-file-name
                    (or (projectile-project-root)
                        (ignore-errors (file-name-directory curr-file))
                        "."))
                   "/"))
         (ws-basename (file-name-nondirectory ws-root)))
    ;; Replace special variables
    (mapcar
     (lambda (str)
       (+str-replace-all
        (append
         (list
          (cons "${workspaceFolder}" ws-root)
          (cons "${workspaceFolderBasename}" ws-basename)
          (cons "${userHome}" (or (getenv "HOME") (expand-file-name "~")))
          (cons "${pathSeparator}" (if (memq system-type
                                             '(windows-nt ms-dos cygwin))
                                       "\\" "/"))
          (cons "${selectedText}" (if (use-region-p)
                                      (buffer-substring-no-properties
                                       (region-beginning) (region-end)) "")))
         ;; To avoid problems if launched from a non-file buffer
         (when curr-file
           (list
            (cons "${file}" curr-file)
            (cons "${relativeFile}" (file-relative-name curr-file ws-root))
            (cons "${relativeFileDirname}" (file-relative-name
                                            (file-name-directory curr-file) ws-root))
            (cons "${fileBasename}" (file-name-nondirectory curr-file))
            (cons "${fileBasenameNoExtension}" (file-name-base curr-file))
            (cons "${fileDirname}" (file-name-directory curr-file))
            (cons "${fileExtname}" (file-name-extension curr-file))
            (cons "${lineNumber}" (line-number-at-pos (point) t)))))
        str))
     (cons program args))))

(defun launch-json--debugger-params (type)
  (let* ((front/backend
          (cond ((string-match launch-json--realgud-debugger-regex type)
                 (cons 'realgud (intern (match-string 1 type))))
                ((string-match launch-json--gud-debugger-regex type)
                 (cons 'gud (intern (match-string 1 type))))
```

```
                (t
                 (cons 'unknown 'unknown))))
         (frontend (car front/backend))
         (backend (cdr front/backend))
         (cmd-sym (unless (eq frontend 'unknown)
                    (intern (format (cond ((eq frontend 'gud) "gud-%s-%s")
                                          ((eq frontend 'realgud) "%s-%s")
                                          (t "%s-%s"))
                                    type
                                    "command-name")))))
    (message "[launch-json:params]: Found type: %s -> { frontend: %s | backend: %s }"
             type (symbol-name frontend) (symbol-name backend))
    (cond ((memq backend '(gud-gdb gdb))
           ;; Special case for '(gud . gdb), uses `gdb-mi'
           (let ((use-gdb-mi (equal front/backend '(gud . gdb))))
             `(:type ,type
               :debug-cmd ,(if use-gdb-mi 'gdb (intern type))
               :args-format " --args %s %s"
               :cmd ,cmd-sym
               :require ,(if use-gdb-mi 'gdb-mi frontend))))
          ((eq backend 'lldb)
           `(:type ,type
             :debug-cmd ,(intern type)
             :args-format " -- %s %s"
             :cmd ,cmd-sym
             :require ,(intern (if (eq frontend 'realgud)
                                   (+str-replace-all '(("--" . "-") (":" . "-")) type)
                                 type))))
          (t ;; TODO: to be expanded for each debugger
           `(:type ,type
             :debug-cmd ,(intern type)
             :args-format " %s %s"
             :cmd ,(if (equal front/backend '(realgud . ipdb)) 'realgud--ipdb-command-name cmd-sym)
             :require ,(cond ((equal front/backend '(realgud . trepan-ni)) 'realgud-trepan-ni)
                             (t frontend)))))))

(defun launch-json--debug-command (params debuggee-args)
  "Return the debug command for PARAMS with DEBUGGEE-ARGS."
  (when-let* ((prog (car debuggee-args))
              (cmd (plist-get params :cmd))
              (pkg (plist-get params :require)))
    (if (or (not pkg) (eq pkg 'unknown))
        (progn (message "[launch-json:command]: Unknown debugger")
               nil)
      (if (require (plist-get params :require) nil t)
          (let ((args (+str-join " " (cdr debuggee-args))))
            (when args (setq args (format (plist-get params :args-format) prog args)))
            (if (bound-and-true-p cmd)
                (concat (eval cmd) (if args args ""))
              (message "[launch-json:command]: Invalid command for type %s" (plist-get params :type))
              nil))
        (message "[launch-json:command]: Cannot add package %s" (symbol-name pkg))
        nil))))

(defun launch-json-read (&optional file)
  "Return the configurations section from a launch.json FILE.
If FILE is nil, launch.json will be searched in the current project,
if it is set to a launch.json file, it will be used instead."
  (let ((launch-json (expand-file-name (or file "launch.json") (or (projectile-project-root) "."))))
    (when (file-exists-p launch-json)
      (message "[launch-json]: Found \"launch.json\" at %s" launch-json)
      (let* ((launch (with-temp-buffer
                       (insert-file-contents launch-json)
                       (json-parse-buffer :object-type 'plist :array-type 'list :null-object nil :false-object
↪  nil)))
             (configs (plist-get launch :configurations)))
        (+filter (lambda (conf)
                   (or (string-match-p launch-json--gud-debugger-regex (plist-get conf :type))
                       (string-match-p launch-json--realgud-debugger-regex (plist-get conf :type))))
                 configs)))))
```

```
147
148 (defun launch-json--config-choice (&optional file)
149   (let* ((confs (or (launch-json-read file)
150                     +launch-json-debug-config))
151          (candidates (mapcar (lambda (conf)
152                                (cons (format "%s [%s]" (plist-get conf :name) (plist-get conf :type))
153                                      conf))
154                              confs)))
155     (cond ((eq (length confs) 1)
156            (car confs))
157           ((> (length confs) 1)
158            (cdr (assoc (completing-read "Configuration: " candidates) candidates))))))
159
160 (defun launch-json-debug (&optional file)
161   "Launch RealGUD or GDB with parameters from `+launch-json-debug-config' or launch.json file."
162   (interactive)
163   (let* ((conf (or launch-json--last-config
164                    (launch-json--config-choice file)))
165          (args (launch-json--substite-special-vars (plist-get conf :program) (plist-get conf :args)))
166          (type (plist-get conf :type))
167          (params (launch-json--debugger-params type)))
168     (when params
169       (let ((debug-cmd (plist-get params :debug-cmd)))
170         (when (fboundp debug-cmd)
171           (setq-local launch-json--last-config conf)
172           (funcall debug-cmd
173                    (launch-json--debug-command params args)))))))
174
175 (map! :leader :prefix ("l" . "custom")
176       (:when (modulep! :tools debugger)
177        :prefix ("d" . "debugger")
178        :desc "GUD/RealGUD launch.json" "d" #'launch-json-debug))
```

6.5.5 Valgrind

```
1 (package! valgrind
2   :recipe `(:local-repo ,(expand-file-name "lisp/valgrind" doom-user-dir)))
```

```
1 (use-package! valgrind
2   :commands valgrind)
```

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 .

```
1 (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

```
1 (defvar emojify-disabled-emojis
2   '(;; Org
3     "" "" "" "" "" "" "" "" "" "" "" "" "®" "" "" "" "" ""
4     ;; Terminal powerline
5     "" ""
```

```
    ;; 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! Let's disable them in C/C++, Rust and Python modes. In addition to that, Lisps do replace lambdas with the greek symbol , however, this cause miss formatting and sometimes messes up with the parenthesis, so let's disable ligatures on Lisps.

```
(defun +appened-to-negation-list (head tail)
  (if (sequencep head)
      (delete-dups
       (if (eq (car tail) 'not)
           (append head tail)
         (append tail head)))
    tail))

(when (modulep! :ui ligatures)
  (setq +ligatures-extras-in-modes
        (+appened-to-negation-list
         +ligatures-extras-in-modes
         '(not c-mode c++-mode emacs-lisp-mode python-mode scheme-mode racket-mode rust-mode)))

  (setq +ligatures-in-modes
        (+appened-to-negation-list
         +ligatures-in-modes
         '(not emacs-lisp-mode scheme-mode racket-mode))))
```

6.7 Checkers (spell & grammar)

6.7.1 Spell-Fu

Install the `aspell` back-end and the dictionaries to use with `spell-fu`

Now, `spell-fu` supports multiple languages! Let's 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-user-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 +my/lang-main)
              (+spell-fu-register-dictionary +my/lang-secondary))))
```

6.7.2 Proselint

A good and funny linter for English prose!, install via `pip install proselint`.

```
(after! flycheck
  (flycheck-define-checker proselint
    "A linter for prose."
    :command ("proselint" source-inplace)
    :error-patterns
    ((warning line-start (file-name) ":" line ":" column ": "
              (id (one-or-more (not (any " "))))
              (message) line-end))
    :modes (text-mode markdown-mode gfm-mode org-mode))

  ;; TODO: Can be enabled automatically for English documents using `guess-language'
  (defun +flycheck-proselint-toggle ()
    "Toggle Proselint checker for the current buffer."
    (interactive)
    (if (and (fboundp 'guess-language-buffer) (string= "en" (guess-language-buffer)))
        (if (memq 'proselint flycheck-checkers)
            (setq-local flycheck-checkers (delete 'proselint flycheck-checkers))
          (setq-local flycheck-checkers (append flycheck-checkers '(proselint))))
      (message "Proselint understands only English!"))))
```

6.7.3 Grammarly

Use either eglot-grammarly or lsp-grammarly.

```
(package! grammarly
  :recipe (:host github
           :repo "emacs-grammarly/grammarly")
  :pin "e47b370faace9ca081db0b87ae3bcfd73212c56d")
```

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

```
(package! eglot-grammarly
  :disable (not (modulep! :tools lsp +eglot))
  :recipe (:host github
           :repo "emacs-grammarly/eglot-grammarly")
  :pin "d95d25615e69e7cc847641800c1886366336c97e")
```

Eglot

```
(use-package! eglot-grammarly
  :commands (+lsp-grammarly-load)
  :init
  (defun +lsp-grammarly-load ()
    "Load Grammarly LSP server for Eglot."
    (interactive)
    (require 'eglot-grammarly)
    (call-interactively #'eglot)))
```

```
(package! lsp-grammarly
  :disable (or (not (modulep! :tools lsp)) (modulep! :tools lsp +eglot))
  :recipe (:host github
           :repo "emacs-grammarly/lsp-grammarly")
  :pin "eab5292037478c32e7d658fb5cba8b8fb6d72a7c")
```

LSP Mode

```
(use-package! lsp-grammarly
  :commands (+lsp-grammarly-load +lsp-grammarly-toggle)
  :init
  (defun +lsp-grammarly-load ()
    "Load Grammarly LSP server for LSP Mode."
    (interactive)
    (require 'lsp-grammarly)
    (lsp-deferred)) ;; or (lsp)

  (defun +lsp-grammarly-enabled-p ()
    (not (member 'grammarly-ls lsp-disabled-clients)))

  (defun +lsp-grammarly-enable ()
    "Enable Grammarly LSP."
    (interactive)
    (when (not (+lsp-grammarly-enabled-p))
      (setq lsp-disabled-clients (remove 'grammarly-ls lsp-disabled-clients))
      (message "Enabled grammarly-ls"))
    (+lsp-grammarly-load))

  (defun +lsp-grammarly-disable ()
    "Disable Grammarly LSP."
    (interactive)
    (when (+lsp-grammarly-enabled-p)
      (add-to-list 'lsp-disabled-clients 'grammarly-ls)
      (lsp-disconnect)
      (message "Disabled grammarly-ls")))

  (defun +lsp-grammarly-toggle ()
    "Enable/disable Grammarly LSP."
    (interactive)
    (if (+lsp-grammarly-enabled-p)
        (+lsp-grammarly-disable)
      (+lsp-grammarly-enable)))

```

```
  (after! lsp-mode
    ;; Disable by default
    (add-to-list 'lsp-disabled-clients 'grammarly-ls))

  :config
  (set-lsp-priority! 'grammarly-ls 1))
```

6.7.4 Grammalecte

```
(package! flycheck-grammalecte
  :recipe (:host github
           :repo "milouse/flycheck-grammalecte")
  :pin "314de13247710410f11d060a214ac4f400c02a71")
```

```
(use-package! flycheck-grammalecte
  :when nil ;; BUG: Disabled, there is a Python error
  :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-data-dir)
        grammalecte-python-package-directory (expand-file-name "grammalecte/grammalecte" doom-data-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{\\(?1:\\(?:equation.\\|align.\\)\\)}.*?\\\\end{\\1}"))

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

6.7.5 LTeX/LanguageTool

Originally, LTeX LS stands for *LaTeX Language Server*, it acts as a Language Server for LaTeX, but not only. It can check the grammar and the spelling of several markup languages such as BibTeX, ConTeXt, LaTeX, Markdown, Org, reStructuredText… and others. Alongside, it provides interfacing with LanguageTool to implement natural language checking.

```
(after! lsp-ltex
  (setq lsp-ltex-language "auto"
        lsp-ltex-mother-tongue +my/lang-mother-tongue
        flycheck-checker-error-threshold 1000)
```

```

  (advice-add
   '+lsp-ltex-setup :after
   (lambda ()
     (setq-local lsp-idle-delay 5.0
                 lsp-progress-function #'lsp-on-progress-legacy
                 lsp-progress-spinner-type 'half-circle
                 lsp-ui-sideline-show-code-actions nil
                 lsp-ui-sideline-show-diagnostics nil
                 lsp-ui-sideline-enable nil)))

  ;; FIXME
  (defun +lsp-ltex-check-document ()
    (interactive)
    (when-let ((file (buffer-file-name)))
      (let* ((uri (lsp--path-to-uri file))
             (beg (region-beginning))
             (end (region-end))
             (req (if (region-active-p)
                      `(:uri ,uri
                        :range ,(lsp--region-to-range beg end))
                    `(:uri ,uri))))
        (lsp-send-execute-command "_ltex.checkDocument" req)))))
```

6.7.6 Go Translate (Google, Bing and DeepL)

```
(package! go-translate
  :recipe (:host github
           :repo "lorniu/go-translate")
  :pin "8bbcbce42a7139f079df3e9b9bda0def2cbb690f")
```

```
(use-package! go-translate
  :commands (gts-do-translate
             +gts-yank-translated-region
             +gts-translate-with)
  :init
  ;; Your languages pairs
  (setq gts-translate-list (list (list +my/lang-main +my/lang-secondary)
                                 (list +my/lang-main +my/lang-mother-tongue)
                                 (list +my/lang-secondary +my/lang-mother-tongue)
                                 (list +my/lang-secondary +my/lang-main)))

  (map! :localleader
        :map (org-mode-map markdown-mode-map latex-mode-map text-mode-map)
        :desc "Yank translated region" "R" #'+gts-yank-translated-region)

  (map! :leader :prefix "l"
        (:prefix ("G" . "go-translate")
         :desc "Bing"                  "b" (lambda () (interactive) (+gts-translate-with 'bing))
         :desc "DeepL"                 "d" (lambda () (interactive) (+gts-translate-with 'deepl))
         :desc "Google"                "g" (lambda () (interactive) (+gts-translate-with))
         :desc "Yank translated region" "R" #'+gts-yank-translated-region
         :desc "gts-do-translate"       "t" #'gts-do-translate))

  :config
  ;; Config the default translator, which will be used by the command `gts-do-translate'
  (setq gts-default-translator
        (gts-translator
         ;; Used to pick source text, from, to. choose one.
         :picker (gts-prompt-picker)
         ;; One or more engines, provide a parser to give different output.
         :engines (gts-google-engine :parser (gts-google-summary-parser))
         ;; Render, only one, used to consumer the output result.
         :render (gts-buffer-render)))

```

```elisp
;; Custom texter which remove newlines in the same paragraph
(defclass +gts-translate-paragraph (gts-texter) ())

(cl-defmethod gts-text ((_ +gts-translate-paragraph))
  (when (use-region-p)
    (let ((text (buffer-substring-no-properties (region-beginning) (region-end))))
      (with-temp-buffer
        (insert text)
        (goto-char (point-min))
        (let ((case-fold-search nil))
          (while (re-search-forward "\n[^\n]" nil t)
            (replace-region-contents
             (- (point) 2) (- (point) 1)
             (lambda (&optional a b) " ")))
          (buffer-string))))))

;; Custom picker to use the paragraph texter
(defclass +gts-paragraph-picker (gts-picker)
  ((texter :initarg :texter :initform (+gts-translate-paragraph))))

(cl-defmethod gts-pick ((o +gts-paragraph-picker))
  (let ((text (gts-text (oref o texter))))
    (when (or (null text) (zerop (length text)))
      (user-error "Make sure there is any word at point, or selection exists"))
    (let ((path (gts-path o text)))
      (setq gts-picker-current-path path)
      (cl-values text path))))

(defun +gts-yank-translated-region ()
  (interactive)
  (gts-translate
   (gts-translator
    :picker (+gts-paragraph-picker)
    :engines (gts-google-engine)
    :render (gts-kill-ring-render))))

(defun +gts-translate-with (&optional engine)
  (interactive)
  (gts-translate
   (gts-translator
    :picker (+gts-paragraph-picker)
    :engines
    (cond ((eq engine 'deepl)
           (gts-deepl-engine
            :auth-key ;; Get API key from ~/.authinfo.gpg (machine api-free.deepl.com)
            (funcall
             (plist-get (car (auth-source-search :host "api-free.deepl.com" :max 1))
                        :secret))
            :pro nil))
          ((eq engine 'bing) (gts-bing-engine))
          (t (gts-google-engine)))
    :render (gts-buffer-render))))
```

6.8 System tools

6.8.1 Disk usage

```elisp
(package! disk-usage :pin "311542e1b5cf74aecf3df871160c5ad5d30a2579")
```

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

6.8.2 Chezmoi

```
(package! chezmoi :pin "781783c483bc8fcdba3a230bb774c3a8a5ebe396")
```

```
(use-package! chezmoi
  :when CHEZMOI-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 (modulep! :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 (modulep! :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)))

(map! :leader :prefix ("l" . "custom")
      (:prefix ("t" . "tools")
       (:when CHEZMOI-P
        :prefix ("c" . "chezmoi")
        :desc "Magit status" "g" #'chezmoi-magit-status
        :desc "Write"        "w" #'chezmoi-write
        :desc "Write files"  "W" #'chezmoi-write-files
        :desc "Find source"  "f" #'chezmoi-find
        :desc "Sync files"   "s" #'chezmoi-sync-files
        :desc "Diff"         "d" #'chezmoi-diff
        :desc "EDiff"        "e" #'chezmoi-ediff
        :desc "Open other"   "o" #'chezmoi-open-other)))
```

6.8.3 Aweshell

```
(package! aweshell
  :recipe (:host github
           :repo "manateelazycat/aweshell")
  :pin "d246df619573ca3f46070cc0ac82d024271ed243")
```

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

6.8.4 Lemon

```
(package! lemon
  :recipe (:host nil
           :repo "https://codeberg.org/emacs-weirdware/lemon.git")
  :pin "37a6e6d6ef0900ca19c820a2dbc122c7fe6d86cf")
```

```
(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)))))
```

6.8.5 eCryptfs

```
(when ECRYPTFS-P
  (defvar +ecryptfs-private-dir "Private")
  (defvar +ecryptfs-buffer-name "*emacs-ecryptfs*")
  (defvar +ecryptfs-config-dir (expand-file-name "~/.ecryptfs"))
  (defvar +ecryptfs-passphrase-gpg (expand-file-name "~/.ecryptfs/my-pass.gpg"))
  (defvar +ecryptfs--wrapping-independent-p (not (null (expand-file-name "wrapping-independent"
↪  +ecryptfs-config-dir))))
  (defvar +ecryptfs--wrapped-passphrase-file (expand-file-name "wrapped-passphrase" +ecryptfs-config-dir))
  (defvar +ecryptfs--mount-passphrase-sig-file (concat (expand-file-name +ecryptfs-private-dir
↪  +ecryptfs-config-dir) ".sig"))
  (defvar +ecryptfs--mount-private-cmd "/sbin/mount.ecryptfs_private")
  (defvar +ecryptfs--umount-private-cmd "/sbin/umount.ecryptfs_private")
  (defvar +ecryptfs--passphrase
    (lambda ()
      (s-trim-right ;; To remove the new line
       (epg-decrypt-file (epg-make-context)
                         +ecryptfs-passphrase-gpg
                         nil))))
  (defvar +ecryptfs--encrypt-filenames-p
    (not (eq 1
             (with-temp-buffer
               (insert-file-contents +ecryptfs--mount-passphrase-sig-file)
               (count-lines (point-min) (point-max))))))
  (defvar +ecryptfs--command-format
```

```
    (if +ecryptfs--encrypt-filenames-p
        "ecryptfs-insert-wrapped-passphrase-into-keyring %s '%s'"
      "ecryptfs-unwrap-passphrase %s '%s' | ecryptfs-add-passphrase -"))

  (defun +ecryptfs-mount-private ()
    (interactive)
    (unless (and (file-exists-p +ecryptfs--wrapped-passphrase-file)
                 (file-exists-p +ecryptfs--mount-passphrase-sig-file))
      (error "Encrypted private directory \"%s\" is not setup properly."
             +ecryptfs-private-dir)
      (return))

    (let ((try-again t))
      (while (and
              ;; In the first iteration, we try to silently mount the ecryptfs private directory,
              ;; this would succeed if the key is available in the keyring.
              (shell-command +ecryptfs--mount-private-cmd
                             +ecryptfs-buffer-name)
              try-again)
        (setq try-again nil)
        (message "Encrypted filenames mode [%s]." (if +ecryptfs--encrypt-filenames-p "ENABLED" "DISABLED"))
        (shell-command
         (format +ecryptfs--command-format
                 +ecryptfs--wrapped-passphrase-file
                 (funcall +ecryptfs--passphrase))
         +ecryptfs-buffer-name))
      (message "Ecryptfs mount private.")))

  (defun +ecryptfs-umount-private ()
    (interactive)
    (while (string-match-p "Sessions still open, not unmounting"
                           (shell-command-to-string +ecryptfs--umount-private-cmd)))
    (message "Unmounted private directory."))

(map! :leader :prefix ("l" . "custom")
      (:prefix ("t" . "tools")
       (:when ECRYPTFS-P
        :prefix ("e" . "ecryptfs")
        :desc "eCryptfs mount private"    "e" #'+ecryptfs-mount-private
        :desc "eCryptfs un-mount private" "E" #'+ecryptfs-umount-private)))
```

6.9 Features

6.9.1 Workspaces

```
(map! :leader
      (:when (modulep! :ui workspaces)
       :prefix ("TAB" . "workspace")
       :desc "Display tab bar"           "TAB" #'+workspace/display
       :desc "Switch workspace"          "."   #'+workspace/switch-to
       :desc "Switch to last workspace"  "$"   #'+workspace/other ;; Modified
       :desc "New workspace"             "n"   #'+workspace/new
       :desc "New named workspace"       "N"   #'+workspace/new-named
       :desc "Load workspace from file"  "l"   #'+workspace/load
       :desc "Save workspace to file"    "s"   #'+workspace/save
       :desc "Delete session"            "x"   #'+workspace/kill-session
       :desc "Delete this workspace"     "d"   #'+workspace/delete
       :desc "Rename workspace"          "r"   #'+workspace/rename
       :desc "Restore last session"      "R"   #'+workspace/restore-last-session
       :desc "Next workspace"            ">"   #'+workspace/switch-right ;; Modified
       :desc "Previous workspace"        "<"   #'+workspace/switch-left ;; Modified
       :desc "Switch to 1st workspace"   "1"   #'+workspace/switch-to-0
       :desc "Switch to 2nd workspace"   "2"   #'+workspace/switch-to-1
       :desc "Switch to 3rd workspace"   "3"   #'+workspace/switch-to-2
       :desc "Switch to 4th workspace"   "4"   #'+workspace/switch-to-3
       :desc "Switch to 5th workspace"   "5"   #'+workspace/switch-to-4
```

```
      :desc "Switch to 6th workspace"   "6"   #'+workspace/switch-to-5
      :desc "Switch to 7th workspace"   "7"   #'+workspace/switch-to-6
      :desc "Switch to 8th workspace"   "8"   #'+workspace/switch-to-7
      :desc "Switch to 9th workspace"   "9"   #'+workspace/switch-to-8
      :desc "Switch to final workspace" "0"   #'+workspace/switch-to-final))
```

6.9.2 Weather

```
(package! wttrin
  :recipe `(:local-repo ,(expand-file-name "lisp/wttrin" doom-user-dir))
  :pin "df5427ce2a5ad4dab652dbb1c4a1834d7ddc2abc")
```

```
;; https://raw.githubusercontent.com/tecosaur/emacs-config/master/lisp/wttrin/wttrin.el
(use-package! wttrin
  :commands wttrin)
```

6.9.3 OpenStreetMap

```
(package! osm :pin "808baabecd9882736b240e6ea9344047aeb669e2")
```

```
(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-data-dir))
  ;; Load Org link support
  (with-eval-after-load 'org
    (require 'osm-ol)))
```

6.9.4 Islamic prayer times

```
(package! awqat
  :recipe (:host github
           :repo "zkry/awqat")
  :pin "72b821aad0cb16165e27643c7d968e1528f00f8d")
```

```
(use-package! 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))
```

6.9.5 Info colors

Better colors for manual pages.

```
(package! info-colors :pin "47ee73cc19b1049eef32c9f3e264ea7ef2aaf8a5")
```

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

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

6.9.6 Zotero Zotxt

```
(package! zotxt :pin "96a132d6b39f6bc19a58913b761d42efc198f8a4")
```

```
(use-package! zotxt
  :when ZOTERO-P
  :commands org-zotxt-mode)
```

6.9.7 CRDT

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

```
(package! crdt :pin "ec0b9cc652c0e980d5865abbba7cbffefea6e8cc")
```

```
(use-package! crdt
  :commands (crdt-share-buffer
             crdt-connect)
  :init
  (cond
   (TUNTOX-P  (setq crdt-use-tuntox t
                    crdt-tuntox-password-in-url t))
   (STUNNEL-P (setq crdt-use-stunnel t))))
```

6.9.8 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 :pin "ed7e32064f92f1315cecbfc43f120bbc7508672c")
```

```
(use-package! ag
  :when AG-P
  :commands (ag
             ag-files
             ag-regexp
             ag-project
             ag-project-files
             ag-project-regexp))
```

6.9.9 Page break lines

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

```
(package! page-break-lines :pin "79eca86e0634ac68af862e15c8a236c37f446dcd")
```

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

6.9.10 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
  :when EAF-P
  :load-path EAF-DIR
  :commands (eaf-open
             eaf-open-browser
             eaf-open-jupyter
             +eaf-open-mail-as-html)
  :init
  (defvar +eaf-enabled-apps
    '(org browser mindmap jupyter org-previewer markdown-previewer file-sender video-player))

  (defun +eaf-app-p (app-symbol)
    (memq app-symbol +eaf-enabled-apps))

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

    (map! :localleader
          :map (mu4e-headers-mode-map mu4e-view-mode-map)
          :desc "Open mail as HTML" "h" #'+eaf-open-mail-as-html
          :desc "Open URL (EAF)" "o" #'eaf-open-browser))

  (when (+eaf-app-p 'pdf-viewer)
    (after! org
      ;; Use EAF PDF Viewer in Org
      (defun +eaf--org-open-file-fn (file &optional link)
        "An wrapper function on `eaf-open'."
        (eaf-open file))

      ;; use `emacs-application-framework' to open PDF file: link
      (add-to-list 'org-file-apps '("\\.pdf\\'" . +eaf--org-open-file-fn)))

    (after! latex
      ;; Link EAF with the LaTeX compiler in emacs. When a .tex file is open,
      ;; the Command>Compile and view (C-c C-a) option will compile the .tex
      ;; file into a .pdf file and display it using EAF. Double clicking on the
      ;; PDF side jumps to editing the clicked section.
      (add-to-list 'TeX-command-list '("XeLaTeX" "%`xelatex --synctex=1%(mode)%' %t" TeX-run-TeX nil t))
      (add-to-list 'TeX-view-program-list '("eaf" eaf-pdf-synctex-forward-view))
      (add-to-list 'TeX-view-program-selection '(output-pdf "eaf"))))

  :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 (symbol-name (font-get doom-font :family))
      eaf-webengine-fixed-font-family (symbol-name (font-get doom-font :family))
      eaf-webengine-serif-font-family (symbol-name (font-get doom-serif-font :family))
      eaf-webengine-font-size 16
      eaf-webengine-fixed-font-size 16
      eaf-webengine-enable-scrollbar t
      eaf-webengine-scroll-step 200
      eaf-webengine-default-zoom 1.25
      eaf-webengine-show-hover-link t
      eaf-webengine-download-path "~/Downloads"
      eaf-webengine-enable-plugin t
      eaf-webengine-enable-javascript t
      eaf-webengine-enable-javascript-access-clipboard t)

(when (display-graphic-p)
  (require 'eaf-all-the-icons)
  (mapc (lambda (v) (eaf-all-the-icons-icon (car v)))
        eaf-all-the-icons-alist))

;; Browser settings
(when (+eaf-app-p 'browser)
  (setq eaf-browser-continue-where-left-off t
        eaf-browser-dark-mode nil ;; "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 +my/lang-main
        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 t
        eaf-browser-aria2-auto-file-renaming t)

  (require 'eaf-browser)

  (defun +eaf-open-mail-as-html ()
    "Open the html mail in EAF Browser."
    (interactive)
    (let ((msg (mu4e-message-at-point t))
          ;; Bind browse-url-browser-function locally, so it works
          ;; even if EAF Browser is not set as a default browser.
          (browse-url-browser-function #'eaf-open-browser))
      (if msg
          (mu4e-action-view-in-browser msg)
        (message "No message at point."))))

;; File manager settings
(when (+eaf-app-p 'file-manager)
  (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 (+eaf-app-p 'file-browser)
  (require 'eaf-file-browser))

;; PDF Viewer settings
(when (+eaf-app-p 'pdf-viewer)
  (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)
  (require 'eaf-pdf-viewer))

;; Org
(when (+eaf-app-p 'rss-reader)
  (setq eaf-rss-reader-split-horizontally nil
        eaf-rss-reader-web-page-other-window t)
  (require 'eaf-org))

;; Org
(when (+eaf-app-p 'org)
  (require 'eaf-org))

;; Mail
;; BUG The `eaf-open-mail-as-html' is not working,
;;     I use `+eaf-open-mail-as-html' instead
(when (+eaf-app-p 'mail)
  (require 'eaf-mail))

;; Org Previewer
(when (+eaf-app-p 'org-previewer)
  (setq eaf-org-dark-mode "follow")
  (require 'eaf-org-previewer))

;; Markdown Previewer
(when (+eaf-app-p 'markdown-previewer)
  (setq eaf-markdown-dark-mode "follow")
  (require 'eaf-markdown-previewer))

;; Jupyter
(when (+eaf-app-p 'jupyter)
  (setq eaf-jupyter-dark-mode "follow"
        eaf-jupyter-font-family (symbol-name (font-get doom-font :family))
        eaf-jupyter-font-size 13)
  (require 'eaf-jupyter))

;; Mindmap
(when (+eaf-app-p 'mindmap)
  (setq eaf-mindmap-dark-mode "follow"
        eaf-mindmap-save-path "~/Dropbox/Mindmap")
  (require 'eaf-mindmap))

;; File Sender
(when (+eaf-app-p 'file-sender)
  (require 'eaf-file-sender))

;; Music Player
(when (+eaf-app-p 'music-player)
  (require 'eaf-music-player))

;; Video Player
(when (+eaf-app-p 'video-player)
  (setq eaf-video-player-keybinding
        '(("p" . "toggle_play")
          ("q" . "close_buffer")
          ("h" . "play_backward")
          ("l" . "play_forward")
          ("j" . "decrease_volume")
          ("k" . "increase_volume")
          ("f" . "toggle_fullscreen")
          ("R" . "restart")))
  (require 'eaf-video-player))

;; Image Viewer
(when (+eaf-app-p 'image-viewer)
  (require 'eaf-image-viewer))

;; Git
```

```
  (when (+eaf-app-p 'git)
    (require 'eaf-git))

  ;; Fix EVIL keybindings
  (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))
              ("file-sender" (kbd eaf-evil-leader-key))
              ("mindmap" (kbd eaf-evil-leader-key))
              (_  (kbd "SPC")))
          (kbd "SPC"))))))
```

6.9.11 Bitwarden

```
(package! bitwarden
  :recipe (:host github
           :repo "seanfarley/emacs-bitwarden")
  :pin "02d6410003a42e7fbeb4aa109aba949eea553706")
```

```
(use-package! bitwarden
  ;;:config
  ;;(bitwarden-auth-source-enable)
  :when BITWARDEN-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))
            ""))))
```

6.9.12 PDF tools

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

```
(after! pdf-tools
  ;; Auto install
  (pdf-tools-install t t t)
  (setq-default pdf-view-image-relief 2
                pdf-view-display-size 'fit-width)

  (add-hook! 'pdf-view-mode-hook
    (when (memq doom-theme '(modus-vivendi doom-one doom-dark+ doom-vibrant))
      ;; TODO: find a more generic way to detect if we are in a dark theme
      (pdf-view-midnight-minor-mode 1)))

```

```
  ;; Color the background, so we can see the PDF page borders
  ;; https://protesilaos.com/emacs/modus-themes#h:ff69dfe1-29c0-447a-915c-b5ff7c5509cd
  (defun +pdf-tools-backdrop ()
    (face-remap-add-relative
     'default
     `(:background ,(if (memq doom-theme '(modus-vivendi modus-operandi))
                        (modus-themes-color 'bg-alt)
                      (doom-color 'bg-alt)))))

  (add-hook 'pdf-tools-enabled-hook #'+pdf-tools-backdrop))

(after! pdf-links
  ;; Tweak for Modus and `pdf-links'
  (when (memq doom-theme '(modus-vivendi modus-operandi))
    ;; https://protesilaos.com/emacs/modus-themes#h:2659d13e-b1a5-416c-9a89-7c3ce3a76574
    (let ((spec (apply #'append
                       (mapcar
                        (lambda (name)
                          (list name
                                (face-attribute 'pdf-links-read-link
                                                name nil 'default)))
                        '(:family :width :weight :slant)))))
      (setq pdf-links-read-link-convert-commands
            `("-density"    "96"
              "-family"     ,(plist-get spec :family)
              "-stretch"    ,(let* ((width (plist-get spec :width))
                                    (name (symbol-name width)))
                               (replace-regexp-in-string "-" ""
                                                         (capitalize name)))
              "-weight"     ,(pcase (plist-get spec :weight)
                               ('ultra-light "Thin")
                               ('extra-light "ExtraLight")
                               ('light       "Light")
                               ('semi-bold   "SemiBold")
                               ('bold        "Bold")
                               ('extra-bold  "ExtraBold")
                               ('ultra-bold  "Black")
                               (_weight      "Normal"))
              "-style"      ,(pcase (plist-get spec :slant)
                               ('italic  "Italic")
                               ('oblique "Oblique")
                               (_slant   "Normal"))
              "-pointsize"  "%P"
              "-undercolor" "%f"
              "-fill"       "%b"
              "-draw"       "text %X,%Y '%c'"))))
```

6.9.13 LTDR

Add the `tldr.el` client for TLDR pages.

```
(package! tldr :pin "d3fd2a809a266c005915026799121c78e8b358f0")
```

```
(use-package! tldr
  :commands (tldr-update-docs tldr)
  :init
  (setq tldr-enabled-categories '("common" "linux" "osx" "sunos")))
```

6.9.14 FZF

```
(package! fzf :pin "21912ebc7e1084aa88c9d8b7715e782a3978ed23")
```

```
(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))
```

6.10 Fun

6.10.1 Speed Type

A game to practice speed typing in Emacs.

```
(package! speed-type :pin "304cb8cd6c30d07577d7d864fd32858a29a73dba")
```

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

6.10.2 2048 Game

```
(package! 2048-game :pin "87936ac2ff37f92daf7a69ae86ba226b031349e4")
```

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

6.10.3 Snow

Let it snow in Emacs!

```
(package! snow :pin "4cd41a703b730a6b59827853f06b98d91405df5a")
```

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

6.10.4 xkcd

```
(package! xkcd
  :recipe (:host github
           :repo "vibhavp/emacs-xkcd")
  :pin "80011da2e7def8f65233d4e0d790ca60d287081d")
```

```
(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 e-Books (nov)

```
(package! nov :pin "cb5f45cbcfbcf263cdeb2d263eb15edefc8b07cb")
```

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))
```

7.3 News feed (elfeed)

Set RSS news feeds

```
(setq elfeed-feeds
      '("https://this-week-in-rust.org/rss.xml"
        "https://planet.emacslife.com/atom.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"))
```

7.4 VPN configuration

7.4.1 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 > sslvpn.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/sslvpn.gpg"

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

7.4.2 Emacs + NetExtender

```
(when NETEXTENDER-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"))))

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

  (map! :leader
        :prefix ("l")
        (:prefix ("t")
         (:prefix ("n" . "netExtender")
          :desc "Start" "s" #'+netextender-start
          :desc "Kill"  "k" #'+netextender-kill))))
```

7.5 Email (mu4e)

Configuring `mu4e` as email client needs three parts:

- Incoming mail configuration IMAP (using `mbsync`)
- Outgoing mail configuration SMTP (using `smtpmail` or `msmtp`)
- Email indexer and viewer (via `mu` and `mu4e`)

7.5.1 IMAP (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.DOMAIN.TLD login USER port 587 password PASSWD`
- 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.

In the configuration file, there is a parameter named `Pass` which should be set to the password in plain text. Most of the examples you can find online uses this parameter, but in real life, nobody uses it, it is extremely unsafe to put the password in plain text configuration file. Instead, `mbsync` configuration file provides the alternative `PassCmd` parameter, which can be set to an arbitrary shell command which gets the password for you. You can set it for example to call the `pass` password manager to output the account password, or to `bw` command (for Bitwarden users). For me, I'm using it with Emacs' `~/.authinfo.gpg`, the `PassCmd` in my configuration uses GPG and `awk` to decrypt and filter the file content to find the required account's password. I set `PassCmd` to something like this:

```
gpg -q --for-your-eyes-only --no-tty --logger-file /dev/null --batch -d ~/.authinfo.gpg | awk '/machine
↪  smtp\.googlemail\.com login username@gmail\.com/ {print $NF}'
```

Remember the line format in the `~/.authinfo.gpg` file:

```
machine smtp.googlemail.com login username@gmail.com port 587 password PASSWD
```

This `PassCmd` command above, decrypts the `~/.authinfo.gpg`, passes it to `awk` to search the line containing `"machine smtp.googlemail.com login username@gmail.com"` and prints the last field (the last field `$NF` in the `awk` command corresponds to the password, as you can see in the line format).

The whole `~/.mbsync` file should look like this:

```
# mbsync config file
# GLOBAL OPTIONS
BufferLimit 50mb              # Global option:   Default buffer size is 10M, too small for modern machines.
Sync All                      # Channels global: Sync everything "Pull Push New ReNew Delete Flags" (default
↪  option)
Create Both                   # Channels global: Automatically create missing mailboxes on both sides
Expunge Both                  # Channels global: Delete messages marked for deletion on both sides
CopyArrivalDate yes           # Channels global: Propagate arrival time with the messages

# SECTION (IMAP4 Accounts)
IMAPAccount work              # IMAP Account name
Host mail.host.ccc            # The host to connect to
User user@host.ccc            # Login user name
SSLVersions TLSv1.2 TLSv1.1  # Supported SSL versions
# Extract password from encrypted ~/.authinfo.gpg
# File format: "machine <SERVER> login <LOGIN> port <PORT> password <PASSWORD>"
# This uses sed to extract <PASSWORD> from line matching the account's <SERVER>
PassCmd "gpg2 -q --for-your-eyes-only --no-tty --logger-file /dev/null --batch -d ~/.authinfo.gpg | awk
↪  '/machine smtp.domain.tld/ {print $NF}'"
AuthMechs *                   # Authentication mechanisms
SSLType IMAPS                 # Protocol (STARTTLS/IMAPS)
CertificateFile /etc/ssl/certs/ca-certificates.crt
# END OF SECTION
```

```
# IMPORTANT NOTE: you need to keep the blank line after each section

# SECTION (IMAP Stores)
IMAPStore work-remote        # Remote storage name
Account work                 # Associated account
# END OF SECTION

# SECTION (Maildir Stores)
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 "gpg2 -q --for-your-eyes-only --no-tty --logger-file /dev/null --batch -d ~/.authinfo.gpg | awk
↪  '/machine smtp.domain.tld/ {print $NF}'"
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 *
```

```
91
92  Channel gmail-all
93  Far :gmail-remote:"[Gmail]/All Mail"
94  Near :gmail-local:"All Mail"
95  SyncState *
96
97  Channel gmail-starred
98  Far :gmail-remote:"[Gmail]/Starred"
99  Near :gmail-local:"Starred"
100 SyncState *
101
102 Channel gmail-spam
103 Far :gmail-remote:"[Gmail]/Spam"
104 Near :gmail-local:"Spam"
105 SyncState *
106
107 # GROUPS PUT TOGETHER CHANNELS, SO THAT WE CAN INVOKE
108 # MBSYNC ON A GROUP TO SYNC ALL CHANNELS
109 #
110 # FOR INSTANCE: "mbsync gmail" GETS MAIL FROM
111 # "gmail-inbox", "gmail-sent", and "gmail-trash"
112 #
113 # SECTION (Groups)
114 Group gmail
115 Channel gmail-inbox
116 Channel gmail-sent
117 Channel gmail-trash
118 Channel gmail-drafts
119 Channel gmail-all
120 Channel gmail-starred
121 Channel gmail-spam
122 # END OF SECTION
```

7.5.2 SMTP (msmtp)

I was using the standard `smtpmail` to send mails; but recently, I'm getting problems when sending mails. I passed a whole day trying to fix mail sending for one of my accounts, at the end of the day, I got a working setup; BUT, sending the first mail always ask me about password! I need to enter the password to be able to send the mail, Emacs asks me then if I want to save it to `~/.authifo.gpg`, when I confirm saving it, it got duplicated in the `.authinfo.gpg` file.

This seems to be a bug; I also found somewhere that `smtpmail` is buggy, and that `msmtp` seems to be a good alternative, so now I'm using a `msmtp`-based setup, and it works like a charm!

For this, we will need an additional configuration file, `~/.msmtprc`, I configure it the same way as `mbsync`, specifying this time SMTP servers instead of IMAP ones. I extract the passwords from `~/.authinfo.gpg` using GPG and `awk`, the same way we did in `mbsync`'s configuration.

The following is a sample file `~/.msmtprc`.

```
1  # Set default values for all following accounts.
2  defaults
3  auth                    on
4  tls                     on
5  tls_starttls            on
6  tls_trust_file          /etc/ssl/certs/ca-certificates.crt
7  logfile                 ~/.msmtp.log
8
9  # Gmail
10 account                 gmail
11 auth                    plain
12 host                    smtp.googlemail.com
13 port                    587
14 from                    username@gmail.com
15 user                    username
16 passwordeval            "gpg -q --for-your-eyes-only --no-tty --logger-file /dev/null --batch -d
   ↪  ~/.authinfo.gpg | awk '/machine smtp.googlemail.com login .*@gmail.com/ {print $NF}'"
17 add_missing_date_header on
```

```
18
19  ## Gmail - aliases
20  account                 alias-account : gmail
21  from                    alias@mail.com
22
23  account                 other-alias : gmail
24  from                    other.alias@address.org
25
26  # Work
27  account                 work
28  auth                    on
29  host                    smtp.domaine.tld
30  port                    587
31  from                    username@domaine.tld
32  user                    username
33  passwordeval            "gpg -q --for-your-eyes-only --no-tty --logger-file /dev/null --batch -d
    ↪  ~/.authinfo.gpg | awk '/machine smtp.domaine.tld/ {print $NF}'"
34  tls_nocertcheck # ignore TLS certificate errors
```

7.5.3 Mail client and indexer (mu and mu4e)

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

```
1  (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:

```
1  (after! mu4e
2    (require 'mu4e-contrib)
3    (require 'mu4e-icalendar)
4    (require 'org-agenda)
5
6    ;; Common parameters
7    (setq mu4e-update-interval (* 3 60) ;; Every 3 min
8          mu4e-index-update-error-warning nil ;; Do not show warning after update
9          mu4e-get-mail-command "mbsync -a" ;; Not needed, as +mu4e-backend is 'mbsync by default
10         mu4e-main-hide-personal-addresses t ;; No need to display a long list of my own addresses!
11         mu4e-attachment-dir (expand-file-name "~/Downloads/mu4e-attachements")
12         mu4e-sent-messages-behavior 'sent ;; Save sent messages
13         mu4e-context-policy 'pick-first   ;; Start with the first context
14         mu4e-compose-context-policy 'ask) ;; Always ask which context to use when composing a new mail
15
16   ;; Use msmtp instead of smtpmail
17   (setq sendmail-program (executable-find "msmtp")
18         send-mail-function #'smtpmail-send-it
19         message-sendmail-f-is-evil t
20         message-sendmail-extra-arguments '("--read-envelope-from")
21         message-send-mail-function #'message-send-mail-with-sendmail
22         message-sendmail-envelope-from 'obey-mail-envelope-from
23         mail-envelope-from 'header
24         mail-personal-alias-file (expand-file-name "mail-aliases.mailrc" doom-user-dir)
25         mail-specify-envelope-from t)
26
27   (setq mu4e-headers-fields '((:flags . 6) ;; 3 flags
28                               (:account-stripe . 2)
29                               (:from-or-to . 25)
30                               (:folder . 10)
31                               (:recipnum . 2)
32                               (:subject . 80)
33                               (:human-date . 8))
34         +mu4e-min-header-frame-width 142
35         mu4e-headers-date-format "%d/%m/%y"
36         mu4e-headers-time-format " %H:%M"
37         mu4e-search-results-limit 1000
38         mu4e-index-cleanup t)
```

```
39
40  (defvar +mu4e-header--folder-colors nil)
41  (appendq! mu4e-header-info-custom
42            '((:folder .
43               (:name "Folder" :shortname "Folder" :help "Lowest level folder" :function
44                (lambda (msg)
45                  (+mu4e-colorize-str
46                   (replace-regexp-in-string "\\`.*/" "" (mu4e-message-field msg :maildir))
47                   '+mu4e-header--folder-colors))))))
48
49  ;; Add a unified inbox shortcut
50  (add-to-list
51   'mu4e-bookmarks
52   '(:name "Unified inbox" :query "maildir:/.*inbox/" :key ?i) t)
53
54  ;; Add shortcut to view yesterday's messages
55  (add-to-list
56   'mu4e-bookmarks
57   '(:name "Yesterday's messages" :query "date:1d..today" :key ?y) t)
58
59  ;; Load a list of my email addresses '+my-addresses', defined as:
60  ;; (setq +my-addresses '("user@gmail.com" "user@hotmail.com"))
61  (load! "lisp/private/+my-addresses.el")
62
63  (when (bound-and-true-p +my-addresses)
64    ;; I like always to add myself in BCC, Lets add a bookmark to show all my BCC mails
65    (defun +mu-long-query (query oper arg-list)
66      (concat "(" (+str-join (concat " " oper " ") (mapcar (lambda (addr) (format "%s:%s" query addr))
    ↪  arg-list)) ")"))
67
68    ;; Build a query to match mails send from "me" with "me" in BCC
69    (let ((bcc-query (+mu-long-query "bcc" "or" +my-addresses))
70          (from-query (+mu-long-query "from" "or" +my-addresses)))
71      (add-to-list
72       'mu4e-bookmarks
73       (list :name "My black copies" :query (format "maildir:/.*inbox/ and %s and %s" from-query bcc-query)
    ↪  :key ?k) t)))
74
75  ;; `mu4e-alert' configuration
76  ;; Use a nicer icon in alerts
77  (setq mu4e-alert-icon "/usr/share/icons/Papirus/64x64/apps/mail-client.svg")
78
79  (defun +mu4e-alert-helper-name-or-email (msg)
80    (let* ((from (car (plist-get msg :from)))
81           (name (plist-get from :name)))
82      (if (or (null name) (eq name ""))
83          (plist-get from :email)
84        name)))
85
86  (defun +mu4e-alert-grouped-mail-notif-formatter (mail-group _all-mails)
87    (when +mu4e-alert-bell-cmd
88      (start-process "mu4e-alert-bell" nil (car +mu4e-alert-bell-cmd) (cdr +mu4e-alert-bell-cmd)))
89    (let* ((filtered-mails (+filter
90                            (lambda (msg)
91                              (not (string-match-p "\\(junk\\|spam\\|trash\\|deleted\\)"
92                                                   (downcase (plist-get msg :maildir)))))
93                            mail-group))
94           (mail-count (length filtered-mails)))
95      (list
96       :title (format "You have %d unread email%s"
97                      mail-count (if (> mail-count 1) "s" ""))
98       :body (concat
99              "• "
100             (+str-join
101              "\n• "
102              (mapcar
103               (lambda (msg)
104                 (format "<b>%s</b>: %s"
105                         (+mu4e-alert-helper-name-or-email msg)
106                         (plist-get msg :subject)))
```

```
                filtered-mails))))))

  ;; I use auto-hiding task manager, setting window
  ;; urgency shows the entier task bar (in KDE), which I find annoying.
  (setq mu4e-alert-set-window-urgency nil
        mu4e-alert-grouped-mail-notification-formatter #'+mu4e-alert-grouped-mail-notif-formatter)

  ;; Org-Msg stuff
  ;; org-msg-[signature|greeting-fmt] are separately set for each account
  (setq mail-user-agent 'mu4e-user-agent) ;; Needed by OrgMsg
  (require 'org-msg)
  (setq org-msg-convert-citation t
        org-msg-default-alternatives
        '((new           . (utf-8 html))
          (reply-to-html . (utf-8 html))
          (reply-to-text . (utf-8 html))))

  (map! :map org-msg-edit-mode-map
        :after org-msg
        :n "G" #'org-msg-goto-body)

  (map! :localleader
        :map (mu4e-headers-mode-map mu4e-view-mode-map)
        :desc "Open URL in Brave"   "b" #'browse-url-chrome ;; Brave
        :desc "Open URL in Firefox" "f" #'browse-url-firefox)

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

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

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

  ;; iCalendar / Org
  (mu4e-icalendar-setup)
  (setq mu4e-icalendar-trash-after-reply nil
        mu4e-icalendar-diary-file "~/Dropbox/Org/diary-invitations.org"
        gnus-icalendar-org-capture-file "~/Dropbox/Org/notes.org"
        gnus-icalendar-org-capture-headline '("Calendar"))

  ;; To enable optional iCalendar->Org sync functionality
  ;; NOTE: both the capture file and the headline(s) inside must already exist
  (gnus-icalendar-org-setup))
```

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, no need for these when using 'msmtp'
   (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")

   ;; No need for these when using 'msmtp'
   (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")))
```

7.5.4 Dashboard

```
(after! mu4e
  ;; Fix icons
  (defun +mu4e-initialise-icons ()
    (setq mu4e-use-fancy-chars t
          mu4e-headers-draft-mark      (cons "D" (+mu4e-normalised-icon "edit"           :set "material"))
          mu4e-headers-flagged-mark    (cons "F" (+mu4e-normalised-icon "flag"           :set "material"))
          mu4e-headers-new-mark        (cons "N" (+mu4e-normalised-icon "file_download"  :set "material" :color
↪  "dred"))
          mu4e-headers-passed-mark     (cons "P" (+mu4e-normalised-icon "forward"        :set "material"))
          mu4e-headers-replied-mark    (cons "R" (+mu4e-normalised-icon "reply"          :set "material"))
          mu4e-headers-seen-mark       (cons "S" "")
          mu4e-headers-trashed-mark    (cons "T" (+mu4e-normalised-icon "delete"         :set "material"))
          mu4e-headers-attach-mark     (cons "a" (+mu4e-normalised-icon "attach_file"    :set "material"))
          mu4e-headers-encrypted-mark  (cons "x" (+mu4e-normalised-icon "lock"           :set "material"))
          mu4e-headers-signed-mark     (cons "s" (+mu4e-normalised-icon "verified_user"  :set "material" :color
↪  "dpurple"))
          mu4e-headers-unread-mark     (cons "u" (+mu4e-normalised-icon "remove_red_eye" :set "material" :color
↪  "dred"))
          mu4e-headers-list-mark       (cons "l" (+mu4e-normalised-icon "list"           :set "material"))
          mu4e-headers-personal-mark   (cons "p" (+mu4e-normalised-icon "person"         :set "material"))
          mu4e-headers-calendar-mark   (cons "c" (+mu4e-normalised-icon "date_range"     :set "material"))))

  (+mu4e-initialise-icons))
```

7.5.5 Save all attachements

```
(after! mu4e
  ;; From https://github.com/sje30/emacs/blob/d7e21b94c79a5b6f244f33faff514036226e183c/mu4e-view-save-all-attachments.el

  (defun +cleanse-subject (sub)
    (replace-regexp-in-string "[^A-Z0-9]+" "-" (downcase sub)))

  (defun +mu4e-view-save-all-attachments (&optional arg)
    "Save all MIME parts from current mu4e gnus view buffer."
    ;; Copied from mu4e-view-save-attachments
    (interactive "P")
    (cl-assert (and (eq major-mode 'mu4e-view-mode)
                    (derived-mode-p 'gnus-article-mode)))
    (let* ((msg (mu4e-message-at-point))
           (id (+cleanse-subject (mu4e-message-field msg :subject)))
           (attachdir (expand-file-name id mu4e-attachment-dir))
           (parts (mu4e~view-gather-mime-parts))
           (handles '())
           (files '())
           dir)
      (mkdir attachdir t)
      (dolist (part parts)
        (let ((fname (or (cdr (assoc 'filename (assoc "attachment" (cdr part))))
                         (seq-find #'stringp
                                   (mapcar (lambda (item) (cdr (assoc 'name item)))
                                           (seq-filter 'listp (cdr part)))))))
          (when fname
            (push `(,fname . ,(cdr part)) handles)
            (push fname files))))
      (if files
          (progn
            (setq dir
                  (if arg (read-directory-name "Save to directory: ")
                    attachdir))
            (cl-loop for (f . h) in handles
                     when (member f files)
                     do (mm-save-part-to-file h
                                              (+file-name-incremental
                                               (expand-file-name f dir)))))
        (mu4e-message "No attached files found"))))

  (map! :map mu4e-view-mode-map
     :ne "P" #'+mu4e-view-save-all-attachments))
```

7.6 IRC

```
;; TODO: Not tangled
(defun +fetch-my-password (&rest params)
  (require 'auth-source)
  (let ((match (car (apply #'auth-source-search params))))
    (if match
        (let ((secret (plist-get match :secret)))
          (if (functionp secret)
              (funcall secret)
            secret))
      (error "Password not found for %S" params))))

(defun +my-nickserv-password (server)
  (+fetch-my-password :user "abougouffa" :host "irc.libera.chat"))

(set-irc-server! "irc.libera.chat"
  '(:tls t
    :port 6697
```

```
    :nick "abougouffa"
    :sasl-password +my-nickserver-password
    :channels ("#emacs")))
```

7.7 Multimedia

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

7.7.1 MPD and MPC

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

I like to launch the music daemon `mpd` using Systemd, let's 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 (+systemd-start "mpd")))
    (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)
  (+systemd-stop "mpd")
  (message "MPD stopped!"))

(defun +mpd-daemon-running-p ()
  "Check if the MPD service is running."
  (+systemd-running-p "mpd"))

(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-P
    (require 'emms-player-mpd))
```

```
7
8   (emms-all)
9   (emms-default-players)
10
11  (setq emms-source-file-default-directory "~/Music/"
12        ;; Load cover images
13        emms-browser-covers 'emms-browser-cache-thumbnail-async
14        emms-seek-seconds 5)
15
16  (if MPD-P
17      ;; If using MPD as backend
18      (setq emms-player-list '(emms-player-mpd)
19            emms-info-functions '(emms-info-mpd)
20            emms-player-mpd-server-name "localhost"
21            emms-player-mpd-server-port "6600"
22            emms-player-mpd-music-directory (expand-file-name "~/Music"))
23    ;; Use whatever backend EMMS is using by default (VLC in my machine)
24    (setq emms-info-functions '(emms-info-tinytag))) ;; use Tinytag, or '(emms-info-exiftool) for Exiftool
25
26  ;; Keyboard shortcuts
27  (global-set-key (kbd "<XF86AudioPrev>")  'emms-previous)
28  (global-set-key (kbd "<XF86AudioNext>")  'emms-next)
29  (global-set-key (kbd "<XF86AudioPlay>")  'emms-pause)
30  (global-set-key (kbd "<XF86AudioPause>") 'emms-pause)
31  (global-set-key (kbd "<XF86AudioStop>")  'emms-stop)
32
33  ;; Try to start MPD or connect to it if it is already started.
34  (when MPD-P
35    (emms-player-set emms-player-mpd 'regex
36                     (emms-player-simple-regexp
37                      "m3u" "ogg" "flac" "mp3" "wav" "mod" "au" "aiff"))
38    (add-hook 'emms-playlist-cleared-hook 'emms-player-mpd-clear)
39    (+mpd-daemon-start))
40
41  ;; Activate EMMS in mode line
42  (emms-mode-line 1)
43
44  ;; More descriptive track lines in playlists
45  ;; From: https://www.emacswiki.org/emacs/EMMS#h5o-15
46  (defun +better-emms-track-description (track)
47    "Return a somewhat nice track description."
48    (let ((artist (emms-track-get track 'info-artist))
49          (album (emms-track-get track 'info-album))
50          (tracknumber (emms-track-get track 'info-tracknumber))
51          (title (emms-track-get track 'info-title)))
52      (cond
53       ((or artist title)
54        (concat
55         (if (> (length artist) 0) artist "Unknown artist") ": "
56         (if (> (length album) 0) album "Unknown album") " - "
57         (if (> (length tracknumber) 0) (format "%02d. " (string-to-number tracknumber)) "")
58         (if (> (length title) 0) title "Unknown title")))
59       (t
60        (emms-track-simple-description track)))))
61
62  (setq emms-track-description-function '+better-emms-track-description)
63
64  ;; Manage notifications, inspired by:
65  ;; https://www.emacswiki.org/emacs/EMMS#h5o-9
66  ;; https://www.emacswiki.org/emacs/EMMS#h5o-11
67  (cond
68   ;; Choose D-Bus to disseminate messages, if available.
69   ((and (require 'dbus nil t) (dbus-ping :session "org.freedesktop.Notifications"))
70    (setq +emms-notifier-function '+notify-via-freedesktop-notifications)
71    (require 'notifications))
72   ;; Try to make use of KNotify if D-Bus isn't present.
73   ((and window-system (executable-find "kdialog"))
74    (setq +emms-notifier-function '+notify-via-kdialog))
75   ;; Use the message system otherwise
76   (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))))
```

7.7.3 EMPV

```
(package! empv
  :recipe (:host github
           :repo "isamert/empv.el")
  :pin "228fd324a11da661cb471a501d04340254cc682f")
```

```
(use-package! empv
  :when MPV-P
  :init
  (map! :leader :prefix ("l m")
        (:prefix ("v" . "empv")
         :desc "Play"                  "p" #'empv-play
         :desc "Seach Youtube"         "y" #'consult-empv-youtube
         :desc "Play radio"            "r" #'empv-play-radio
         :desc "Save current playlist" "s" #'+empv-save-playtlist-to-file))
  :config
  ;; See https://docs.invidious.io/instances/
  (setq empv-invidious-instance "https://invidious.projectsegfau.lt/api/v1"
        empv-audio-dir "~/Music"
        empv-video-dir "~/Videos"
        empv-max-directory-search-depth 6
        empv-radio-log-file (expand-file-name "logged-radio-songs.org" org-directory)
        empv-audio-file-extensions '("webm" "mp3" "ogg" "wav" "m4a" "flac" "aac" "opus")
        ;; Links from https://www.radio-browser.info
        empv-radio-channels
        '(("El-Bahdja FM" . "http://webradio.tda.dz:8001/ElBahdja_64K.mp3")
          ("El-Chaabia" . "https://radio-dzair.net/proxy/chaabia?mp=/stream")
          ("Quran Radio" . "http://stream.radiojar.com/0tpy1h0kxtzuv")
          ("Algeria International" . "https://webradio.tda.dz/Internationale_64K.mp3")
          ("JOW Radio" . "https://str0.creacast.com/jowradio")
          ("Europe1" . "http://ais-live.cloud-services.paris:8000/europe1.mp3")
          ("France Iter" . "http://direct.franceinter.fr/live/franceinter-hifi.aac")
          ("France Info" . "http://direct.franceinfo.fr/live/franceinfo-midfi.mp3")
          ("France Culture" . "http://icecast.radiofrance.fr/franceculture-hifi.aac")
          ("France Musique" . "http://icecast.radiofrance.fr/francemusique-hifi.aac")
```

```
        ("FIP" . "http://icecast.radiofrance.fr/fip-hifi.aac")
        ("Beur FM" . "http://broadcast.infomaniak.ch/beurfm-high.aac")
        ("Skyrock" . "http://icecast.skyrock.net/s/natio_mp3_128k")))

(empv-playlist-loop-on)

;; Hacky palylist management (only supports saving playlist,
;; loading a playlist can be achieved using `empv-play-file')

(defun +empv--dl-playlist (playlist &optional dist)
  (let ((default-directory
         (or dist
             (let ((d (expand-file-name "empv-downloads" empv-audio-dir)))
               (unless (file-directory-p d) (mkdir d t)) d)))
        (vids (+filter
               'identity ;; Filter nils
               (mapcar
                (lambda (item)
                  (when-let
                      ((vid (when (string-match
                                   (rx (seq "watch?v=" (group-n 1 (one-or-more (or alnum "_" "-")))))
                                   item)
                              (match-string 1 item))))
                    vid))
                playlist)))
        (proc-name "empv-yt-dlp"))
    (unless (zerop (length vids))
      (message "Downloading %d songs to %s" (length vids) default-directory)
      (when (get-process proc-name)
        (kill-process proc-name))
      (make-process :name proc-name
                    :buffer (format "*%s*" proc-name)
                    :command (append
                              (list
                               (executable-find "yt-dlp")
                               "--no-abort-on-error"
                               "--no-colors"
                               "--extract-audio"
                               "--no-progress"
                               "-f" "bestaudio")
                              vids)
                    :sentinel (lambda (prc event)
                                (when (string= event "finished\n")
                                  (message "Finished downloading playlist files!")))))))

(defun +empv-download-playtlist-files (&optional path)
  (interactive "DSave download playlist files to: ")
  (empv--playlist-apply #'+empv--dl-playlist path)))
```

7.7.4 Keybindings

Lastly, let's define the keybindings for these commands, under `<leader> l m`.

```
(map! :leader :prefix ("l" . "custom")
      (:when (modulep! :app emms)
       :prefix ("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 "Stop"                       "s" #'emms-stop))
```

Then we add MPD related keybindings if MPD is used.

```
(map! :leader :prefix ("l m")
      (:when (and (modulep! :app emms) MPD-P)
       :prefix ("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")
  :pin "7a269c9aef9ece7ecf997f6abb9cd3818403b0bb")
```

```
(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-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 cames 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 let's 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"))
  :pin "1334f44725bd80a265de858d652f3fde4ae401fa")
```

```
(use-package! maxima
  :when MAXIMA-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 let's 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 subdirectory `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/*"))
  :pin "519ea34095e749634d3a188733a3ad284b593e12")
```

```
(use-package! imaxima
  :when MAXIMA-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))
```

7.9 FriCAS

The FriCAS comes bundled with an Emacs mode, let's load it.

```
(use-package! fricas
  :when FRICAS-P
  :load-path FRICAS-DIR
  :commands (fricas-mode fricas-eval fricas))
```

7.10 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))
```

7.10.1 Basic settings

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

Let's disable `org-roam` if the directory doesn't exist.

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

7.10.2 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)))
```

7.10.3 Org Roam Capture template

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

7.10.4 View notes in Deft

```
;; From
↪  https://org-roam.discourse.group/t/configure-deft-title-stripping-to-hide-org-roam-template-headers/478/10
(use-package! deft
  :after org
  :ensure t
  :bind
  ("C-c n d" . deft)
  :init
```

```
  (setq deft-directory org-roam-directory
        ;; deft-recursive t
        deft-use-filter-string-for-filename t
        deft-default-extension "org")
  :config
  (defun +deft-parse-title (file contents)
    "Parse the given FILE and CONTENTS and determine the title.
     If `deft-use-filename-as-title' is nil, the title is taken to
     be the first non-empty line of the FILE.  Else the base name of the FILE is
     used as title."
    (let ((begin (string-match "^#\\+[tT][iI][tT][lL][eE]: .*$" contents)))
      (if begin
          (string-trim (substring contents begin (match-end 0)) "#\\+[tT][iI][tT][lL][eE]: *" "[\n\t ]+")
        (deft-base-filename file))))

  (advice-add 'deft-parse-title :override #'+deft-parse-title)

  (setq deft-strip-summary-regexp
        (concat "\\("
                "[\n\t]" ;; blank
                "\\|^#\\+[[:alpha:]_]+:.*$" ;; org-mode metadata
                "\\|^:PROPERTIES:\n\\(.+\n\\)+:END:\n" ;; org-roam ID
                "\\|\\[\\[\\(.*\\]\\)" ;; any link
                "\\)")))
```

8 Programming

8.1 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)))
```

8.2 Vimrc

```
(package! vimrc-mode
  :recipe (:host github
           :repo "mcandre/vimrc-mode")
  :pin "13bc150a870d5d4a95f1111e4740e2b22813c30e")
```

```
(use-package! vimrc-mode
  :mode "\\.vim\\(rc\\)?\\'")
```

8.3 ESS

View data frames better with

```
(package! ess-view :pin "925cafd876e2cc37bc756bb7fcf3f34534b457e2")
```

8.4 Python IDE

```
(package! elpy :pin "de31d30003c515c25ff7bfd3a361c70c298f78bb")
```

```
(use-package! elpy
  :hook ((elpy-mode . flycheck-mode)
         (elpy-mode . (lambda ()
                        (set (make-local-variable 'company-backends)
                             '((elpy-company-backend :with company-yasnippet))))))
  :config
  (elpy-enable))
```

8.5 Semgrep

Lightweight static analysis for many languages. Find bug variants with patterns that look like source code.

```
(package! semgrep
  :disable t
  :recipe (:host github
           :repo "Ruin0x11/semgrep.el")
  :pin "d95d25615e69e7cc847641800c1886366336c97e")
```

8.6 GNU Octave

Files with the `.m` extension gets recognized automatically as Objective-C files. I've never used Objective-C before, so let's change it to be recognized as Octave/Matlab files.

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

```
(defun +octave-eval-last-sexp ()
  "Evaluate Octave sexp before point and print value into current buffer."
  (interactive)
  (inferior-octave t)
  (let ((print-escape-newlines nil)
        (opoint (point)))
    (prin1
     (save-excursion
       (forward-sexp -1)
       (inferior-octave-send-list-and-digest
        (list (concat (buffer-substring-no-properties (point) opoint)
                      "\n")))
       (mapconcat 'identity inferior-octave-output-list "\n")))))

(defun +eros-octave-eval-last-sexp ()
```

```
  "Wrapper for `+octave-eval-last-sexp' that overlays results."
  (interactive)
  (eros--eval-overlay
   (octave-eval-last-sexp)
   (point)))

(map! :localleader
      :map (octave-mode-map)
      (:prefix ("e" . "eval")
       :desc "Eval and print last sexp" "e" #'+eros-octave-eval-last-sexp))
```

8.7 ROS

8.7.1 Extensions

Add ROS specific file formats:

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

;; Use gdb-script-mode for msg and srv files
(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))
```

8.7.2 ROS bags

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

```
(when ROSBAG-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)))
```

8.7.3 `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 :pin "3fd1ea892e44f7fe6f86df2b5c0a0a1e0f3913fa")
(package! ros
  :recipe (:host github
           :repo "DerBeutlin/ros.el")
  :pin "f66d2177b00b277a36c058549c477d854148623c")
```

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/")))))
```

8.8 Scheme

```
(after! geiser
  (setq geiser-default-implementation 'guile
        geiser-chez-binary "chez-scheme")) ;; default is "scheme"
```

8.9 Embedded systems

8.9.1 Embed.el

Some embedded systems development tools.

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

- probe-rs-debugger
- stm32-emacs
- cortex-debug with potential integration with DAP
- esp-debug-adapter

```
(package! embed
  :recipe (:host github
           :repo "sjsch/embed-el")
  :pin "8df65777450c6c70a418d1bd2ba87ad590377b47")
```

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

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

```
        (:when (modulep! :tools debugger +lsp)
         :prefix ("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)))
```

8.9.2 Arduino

```
(package! arduino-mode
  :recipe (:host github
           :repo "bookest/arduino-mode")
  :pin "3e2bad4569ad26e929e6db2cbcff0d6d36812698")
```

8.9.3 Bitbake (Yocto)

Add support for Yocto Project files.

```
(package! bitbake-modes
  :recipe (:host nil
           :repo "https://bitbucket.org/olanilsson/bitbake-modes")
    :pin "a042118fd2010ef203a11e1de14e7537f8184a78")
```

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

8.10 Git & VC

8.10.1 Magit

```
(after! code-review
  (setq code-review-auth-login-marker 'forge))
```

```

(after! magit
  ;; Disable if it causes performance issues
  (setq magit-diff-refine-hunk t))
```

Granular diff-highlights for *all* hunks

```

(after! magit
  ;; Show gravatars
  (setq magit-revision-show-gravatars '("^Author:     " . "^Commit:     ")))
```

Gravatars

```
(package! company-conventional-commits
  :recipe `(:local-repo ,(expand-file-name "lisp/company-conventional-commits" doom-user-dir)))
```

WIP Company for commit messages

```
(use-package! company-conventional-commits
  :after (magit company)
  :config
  (add-hook
   'git-commit-setup-hook
   (lambda ()
     (add-to-list 'company-backends 'company-conventional-commits))))
```

```
(package! magit-pretty-graph
  :recipe (:host github
           :repo "georgek/magit-pretty-graph")
  :pin "26dc5535a20efe781b172bac73f14a5ebe13efa9")
```

Pretty graph

```
(use-package! magit-pretty-graph
  :after magit
  :init
  (setq magit-pg-command
        (concat "git --no-pager log"
                " --topo-order --decorate=full"
                " --pretty=format:\"%H%x00%P%x00%an%x00%ar%x00%s%x00%d\""
                " -n 2000")) ;; Increase the default 100 limit

  (map! :localleader
        :map (magit-mode-map)
        :desc "Magit pretty graph" "p" (cmd! (magit-pg-repo (magit-toplevel)))))
```

8.10.2 Repo

This adds Emacs integration of `repo`, The Multiple Git Repository Tool. 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 :pin "e504aa831bfa38ddadce293face28b3c9d9ff9b7")
```

```
(use-package! repo
  :when REPO-P
  :commands repo-status)
```

8.10.3 Blamer

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

```
(package! blamer
  :recipe (:host github
           :repo "artawower/blamer.el")
  :pin "99b43779341af0d924bfe2a9103993a6b9e3d3b2")
```

```
(use-package! blamer
  :commands (blamer-mode)
  ;; :hook ((prog-mode . blamer-mode))
  :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))))
```

8.11 Assembly

Add some packages for better assembly coding.

```
(package! nasm-mode :pin "65ca6546fc395711fac5b3b4299e76c2303d43a8")
(package! haxor-mode :pin "6fa25a8e6b6a59481bc0354c2fe1e0ed53cbdc91")
(package! mips-mode :pin "98795cdc81979821ac35d9f94ce354cd99780c67")
(package! riscv-mode :pin "8e335b9c93de93ed8dd063d702b0f5ad48eef6d7")
(package! x86-lookup :pin "1573d61cc4457737b94624598a891c837fb52c16")
```

```
(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
  :mode "\\.riscv\\'")

(use-package! x86-lookup
  :commands (x86-lookup)
  :config
  (when (modulep! :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 (expand-file-name "x86-lookup/325383-sdm-vol-2abcd.pdf" doom-data-dir)))
```

8.12 Disaster

```
(package! disaster :pin "0c13bd244cc43773af81e52ce73a55f199d58a61")
```

```
(use-package! disaster
  :commands (disaster)
  :init
  (setq disaster-assembly-mode 'nasm-mode)

  (map! :localleader
        :map (c++-mode-map c-mode-map fortran-mode)
        :desc "Disaster" "d" #'disaster))
```

8.13 Devdocs

```
(package! devdocs
  :recipe (:host github
           :repo "astoff/devdocs.el"
           :files ("*.el"))
  :pin "61ce83b79dc64e2f99d7f016a09b97e14b331459")
```

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

8.14 Systemd

For editing systemd unit files.

```
(package! systemd :pin "b6ae63a236605b1c5e1069f7d3afe06ae32a7bae")
(package! journalctl-mode :pin "c5bca1a5f42d2fe2a00fdf52fe480137ace971d3")
```

```
(use-package! journalctl-mode
  :commands (journalctl
             journalctl-boot
             journalctl-unit
             journalctl-user-unit)
  :init
  (map! :map journalctl-mode-map
        :nv "J" #'journalctl-next-chunk
        :nv "K" #'journalctl-previous-chunk))
```

8.15 PKGBUILD

```
(package! pkgbuild-mode :pin "9525be8ecbd3a0d0bc7cc27e6d0f403e111aa067")
```

```
(use-package! pkgbuild-mode
  :commands (pkgbuild-mode)
  :mode "/PKGBUILD$")
```

8.16 Franca IDL

Add support for *Franca Interface Definition Language*.

```
(package! franca-idl
  :recipe (:host github
           :repo "zeph1e/franca-idl.el")
  :pin "12703ee42533bd851a1d911609020f71eb31204a")
```

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

8.17 LaTeX

```
(package! aas
  :recipe (:host github
           :repo "ymarco/auto-activating-snippets")
  :pin "566944e3b336c29d3ac11cd739a954c9d112f3fb")
```

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

8.18 Flycheck + Projectile

WIP: Not working atm!

```
(package! flycheck-projectile
  :recipe (:host github
           :repo "nbfalcon/flycheck-projectile")
  :pin "ce6e9e8793a55dace13d5fa13badab2dca3b5ddb")
```

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

8.19 Graphviz

Graphviz is a nice method of visualizing simple graphs, based on th DOT graph description language (`*.dot` / `*.gv` files).

```
(package! graphviz-dot-mode :pin "6e96a89762760935a7dff6b18393396f6498f976")
```

```
(use-package! graphviz-dot-mode
  :commands graphviz-dot-mode
  :mode ("\\.dot\\'" "\\.gv\\'")
  :init
  (after! org
    (setcdr (assoc "dot" org-src-lang-modes) 'graphviz-dot))

  :config
  (require 'company-graphviz-dot))
```

8.20 Modula-II

Gaius Mulley is doing a great job, bringing Modula-II support to GCC, he also created a new mode for Modula-II with extended features. The mode is included with the GNU Modula 2 source code, and can be downloaded separately from the Git repository, from here gm2-mode.el. I added `(provide 'gm2-mode)` to the `gm2-mode.el`.

```
(package! gm2-mode
  :recipe `(:local-repo ,(expand-file-name "lisp/gm2-mode" doom-user-dir)))
```

8.21 Mermaid

```
(package! mermaid-mode :pin "a98a9e733b1da1e6a19e68c1db4367bf46283479")

(package! ob-mermaid
  :recipe (:host github
           :repo "arnm/ob-mermaid")
  :pin "b4ce25699e3ebff054f523375d1cf5a17bd0dbaf")
```

```
(use-package! mermaid-mode
  :commands mermaid-mode
  :mode "\\.mmd\\'")

(use-package! ob-mermaid
  :after org
  :init
  (after! org
    (add-to-list 'org-babel-load-languages '(mermaid . t))))
```

8.22 The V Programming Language

```
(package! v-mode :pin "a701f4cedfff91cf4bcd17c9a2cd16a49f942743")
```

```
(use-package! v-mode
  :mode ("\\(\\.v?v\\|\\.vsh\\)$" . 'v-mode)
  :config
  (map! :localleader
        :map (v-mode-map)
        :desc "v-format-buffer" "f" #'v-format-buffer
        :desc "v-menu" "m" #'v-menu))
```

8.23 Inspector

```
(package! inspector
  :recipe (:host github
           :repo "mmontone/emacs-inspector")
  :pin "7e0a7afc425b8fd0f8a769849eecf11e44cb3c86")
```

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

9 Office

9.1 Org additional packages

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

```
(unpin! org-roam) ;; To avoid problems with org-roam-ui
(package! websocket :pin "82b370602fa0158670b1c6c769f223159affce9b")
(package! org-roam-ui :pin "16a8da9e5107833032893bc4c0680b368ac423ac")
(package! org-wild-notifier :pin "9392b06d20b2f88e45a41bea17bb2f10f24fd19c")
(package! org-fragtog :pin "c675563af3f9ab5558cfd5ea460e2a07477b0cfd")
(package! org-appear :pin "60ba267c5da336e75e603f8c7ab3f44e6f4e4dac")
(package! org-super-agenda :pin "f4f528985397c833c870967884b013cf91a1da4a")
(package! doct :pin "506c22f365b75f5423810c4933856802554df464")

(package! citar-org-roam
  :recipe (:host github
           :repo "emacs-citar/citar-org-roam")
  :pin "29688b89ac3bf78405fa0dce7e17965aa8fe0dff")

(package! org-menu
  :recipe (:host github
           :repo "sheijk/org-menu")
  :pin "9cd10161c2b50dfef581f3d0441683eeeae6be59")

(package! caldav
  :recipe (:host github
           :repo "dengste/org-caldav")
  :pin "8569941a0a5a9393ba51afc8923fd7b77b73fa7a")

(package! org-ol-tree
  :recipe (:host github
           :repo "Townk/org-ol-tree")
  :pin "207c748aa5fea8626be619e8c55bdb1c16118c25")

(package! org-modern
  :recipe (:host github
           :repo "minad/org-modern")
  :pin "828cf100c62fc9dfb50152c192ac3a968c1b54bc")

(package! org-bib
  :recipe (:host github
           :repo "rougier/org-bib-mode")
  :pin "fed9910186e5e579c2391fb356f55ae24093b55a")

(package! academic-phrases
  :recipe (:host github
           :repo "nashamri/academic-phrases")
  :pin "25d9cf67feac6359cb213f061735e2679c84187f")

(package! phscroll
  :recipe (:host github
           :repo "misohena/phscroll")
  :pin "65e00c89f078997e1a5665d069ad8b1e3b851d49")
```

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 nil   ; 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-pretty-entities-include-sub-superscripts nil
      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 as a local variable (using) and `.dir-locals.el`, but it didn't work. This hack should solve the problem now!

```
;; 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)
```

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 "IDEA(i)" "TODO(t)" "NEXT(n)" "PROJ(p)" "STRT(s)" "WAIT(w)" "HOLD(h)" "|" "DONE(d)" "KILL(k)")
        (sequence "[ ](T)" "[-](S)" "|" "[X](D)")
        (sequence "|" "OKAY(o)" "YES(y)" "NO(n)")))

(setq org-todo-keyword-faces
      '(("IDEA" . (:foreground "goldenrod" :weight bold))
        ("NEXT" . (:foreground "IndianRed1" :weight bold))
        ("STRT" . (:foreground "OrangeRed" :weight bold))
        ("WAIT" . (:foreground "coral" :weight bold))
        ("KILL" . (:foreground "DarkGreen" :weight bold))
        ("PROJ" . (:foreground "LimeGreen" :weight bold))
        ("HOLD" . (:foreground "orange" :weight bold))))

(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

```
(setq org-tag-persistent-alist
      '((:startgroup . nil)
        ("home"      . ?h)
        ("research"  . ?r)
        ("work"      . ?w)
        (:endgroup   . nil)
        (:startgroup . nil)
        ("tool"      . ?o)
        ("dev"       . ?d)
        ("report"    . ?p)
        (:endgroup   . nil)
        (:startgroup . nil)
        ("easy"      . ?e)
        ("medium"    . ?m)
        ("hard"      . ?a)
        (:endgroup   . nil)
        ("urgent"    . ?u)
        ("key"       . ?k)
        ("bonus"     . ?b)
        ("ignore"    . ?i)
        ("noexport"  . ?x)))

(setq org-tag-faces
      '(("home"      . (:foreground "goldenrod"  :weight bold))
        ("research"  . (:foreground "goldenrod"  :weight bold))
        ("work"      . (:foreground "goldenrod"  :weight bold))
        ("tool"      . (:foreground "IndianRed1" :weight bold))
        ("dev"       . (:foreground "IndianRed1" :weight bold))
        ("report"    . (:foreground "IndianRed1" :weight bold))
        ("urgent"    . (:foreground "red"        :weight bold))
        ("key"       . (:foreground "red"        :weight bold))
        ("easy"      . (:foreground "green4"     :weight bold))
        ("medium"    . (:foreground "orange"     :weight bold))
        ("hard"      . (:foreground "red"        :weight bold))
        ("bonus"     . (:foreground "goldenrod"  :weight bold))
        ("ignore"    . (:foreground "Gray"       :weight bold))
        ("noexport"  . (:foreground "LimeGreen"  :weight bold))))

```

Tags

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)
            (expand-file-name "archive.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
  :defer t
  :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))
```

Let's 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)))
```

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`

```elisp
(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 arguments.

```elisp
(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`).

```elisp
(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)))))
```

```
17
18 (defun +yas/org-most-common-no-property-lang ()
19   "Find the lang with the most source blocks that has no global header-args, else nil."
20   (let (src-langs header-langs)
21     (save-excursion
22       (goto-char (point-min))
23       (while (re-search-forward "^[ \t]*#\\+begin_src" nil t)
24         (push (+yas/org-src-lang) src-langs))
25       (goto-char (point-min))
26       (while (re-search-forward "^[ \t]*#\\+property: +header-args" nil t)
27         (push (+yas/org-src-lang) header-langs)))
28
29     (setq src-langs
30           (mapcar #'car
31                   ;; sort alist by frequency (desc.)
32                   (sort
33                    ;; generate alist with form (value . frequency)
34                    (cl-loop for (n . m) in (seq-group-by #'identity src-langs)
35                             collect (cons n (length m)))
36                    (lambda (a b) (> (cdr a) (cdr b))))))
37
38     (car (cl-set-difference src-langs header-langs :test #'string=))))
```

Translate capital keywords to lower case 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

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

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

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

```
1  (use-package! org-menu
2    :commands (org-menu)
3    :init
4    (map! :localleader
5          :map org-mode-map
6          :desc "Org menu" "M" #'org-menu))
```

Org menu

```
(when (and (modulep! :tools lsp) (not (modulep! :tools lsp +eglot)))
  (cl-defmacro +lsp-org-babel-enable (lang)
    "Support LANG in org source code block."
    ;; (setq centaur-lsp 'lsp-mode)
    (cl-check-type lang stringp)
    (let* ((edit-pre (intern (format "org-babel-edit-prep:%s" lang)))
           (intern-pre (intern (format "lsp--%s" (symbol-name edit-pre)))))
      `(progn
         (defun ,intern-pre (info)
           (let ((file-name (->> info caddr (alist-get :file))))
             (unless file-name
               (setq file-name (make-temp-file "babel-lsp-")))
             (setq buffer-file-name file-name)
             (lsp-deferred)))
         (put ',intern-pre 'function-documentation
              (format "Enable lsp-mode in the buffer of org source block (%s)."
                      (upcase ,lang)))
         (if (fboundp ',edit-pre)
             (advice-add ',edit-pre :after ',intern-pre)
           (progn
             (defun ,edit-pre (info)
               (,intern-pre info))
             (put ',edit-pre 'function-documentation
                  (format "Prepare local buffer environment for org source block (%s)."
                          (upcase ,lang))))))))

  (defvar +org-babel-lang-list
    '("go" "python" "ipython" "bash" "sh"))

  (dolist (lang +org-babel-lang-list)
    (eval `(+lsp-org-babel-enable ,lang))))
```

LSP in src blocks

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
```

LaTeX inline markup Needs to make a ?, with this hack you can write `[[latex:textsc][Some text]]`.

```
(org-add-link-type
 "latex" nil
 (lambda (path desc format)
   (cond
    ((eq format 'html)
     (format "<span class=\"%s\">%s</span>" path desc))
    ((eq format 'latex)
     (format "\\%s{%s}" path desc)))))
```

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
  :hook (org-mode . org-modern-mode)
  :config
  (setq org-modern-star '(" " " " " " " " " " " " " " " ")
        org-modern-table-vertical 5
        org-modern-table-horizontal 2
        org-modern-list '((43 . " ") (45 . "–") (42 . "•"))
        org-modern-footnote (cons nil (cadr org-script-display))
        org-modern-priority t
        org-modern-block t
        org-modern-block-fringe nil
        org-modern-horizontal-rule t
        org-modern-keyword
        '((t                      . t)
          ("title"                . " ")
          ("subtitle"             . " ")
          ("author"               . " ")
          ("email"                . "@")
          ("date"                 . " ")
          ("lastmod"              . " ")
          ("property"             . " ")
          ("options"              . " ")
          ("startup"              . " ")
          ("macro"                . " ")
          ("bind"                 . #(" " 0 1 (display (raise -0.1))))
          ("bibliography"         . " ")
          ("print_bibliography"   . #(" " 0 1 (display (raise -0.1))))
          ("cite_export"          . " ")
          ("print_glossary"       . #("  " 0 1 (display (raise -0.1))))
          ("glossary_sources"     . #(" " 0 1 (display (raise -0.14))))
          ("export_file_name"     . " ")
          ("include"              . " ")
          ("setupfile"            . " ")
          ("html_head"            . " ")
          ("html"                 . " ")
          ("latex_class"          . " ")
          ("latex_class_options"  . #(" " 1 2 (display (raise -0.14))))
          ("latex_header"         . " ")
          ("latex_header_extra"   . " ")
          ("latex"                . " ")
          ("beamer_theme"         . " ")
          ("beamer_color_theme"   . #(" " 1 2 (display (raise -0.12))))
          ("beamer_font_theme"    . " ")
          ("beamer_header"        . " ")
          ("beamer"               . " ")
          ("attr_latex"           . " ")
          ("attr_html"            . " ")
          ("attr_org"             . " ")
          ("name"                 . " ")
          ("header"               . ">")
          ("caption"              . " ")
          ("RESULTS"              . " ")
          ("language"             . " ")
          ("hugo_base_dir"        . " ")
          ("latex_compiler"       . " ")
          ("results"              . " ")
          ("filetags"             . "#")
          ("created"              . " ")
          ("export_select_tags"   . " ")
```

```
          ("export_exclude_tags" . " ")))

  ;; Change faces
  (custom-set-faces! '(org-modern-tag :inherit (region org-modern-label)))
  (custom-set-faces! '(org-modern-statistics :inherit org-checkbox-statistics-todo)))
```

Org Modern

Not let's remove the overlap between the substitutions we set here and those that Doom applies via `:ui ligatures` and `:lang org`.

```
(when (modulep! :ui ligatures)
  (defadvice! +org-init-appearance-h--no-ligatures-a ()
    :after #'+org-init-appearance-h
    (set-ligatures! 'org-mode
                    :name nil
                    :src_block nil
                    :src_block_end nil
                    :quote nil
                    :quote_end nil)))
```

We'll bind this to `O` on the `org-mode` localleader, and manually apply a PR recognising the pgtk window system.

```
(use-package! org-ol-tree
  :commands org-ol-tree
  :config
  (setq org-ol-tree-ui-icon-set
        (if (and (display-graphic-p)
                 (fboundp 'all-the-icons-material))
            'all-the-icons
          'unicode))
  (org-ol-tree-ui--update-icon-set))

(map! :localleader
      :map org-mode-map
      :desc "Outline" "O" #'org-ol-tree)
```

```
(defvar +org-responsive-image-percentage 0.4)
(defvar +org-responsive-image-width-limits '(400 . 700)) ;; '(min-width . max-width)

(defun +org--responsive-image-h ()
  (when (eq major-mode 'org-mode)
    (setq org-image-actual-width
          (max (car +org-responsive-image-width-limits)
               (min (cdr +org-responsive-image-width-limits)
                    (truncate (* (window-pixel-width) +org-responsive-image-percentage)))))))

(add-hook 'window-configuration-change-hook #'+org--responsive-image-h)
```

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

LaTeX fragments

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

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

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

Prettier rendering 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 +org-format-latex-set-scale (scale)
  (setq org-format-latex-options (plist-put org-format-latex-options :scale scale)))

;; Set the default scale
(+org-format-latex-set-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.

> This hack is not properly working right now!, it seems to work only with `align` blocks. **NEEDS INVESTIGATION.**

```
(defun +parse-the-fun (str)
  "Parse the LaTeX environment STR.
Return an AST with newlines counts in each level."
  (let (ast)
    (with-temp-buffer
      (insert str)
      (goto-char (point-min))
      (while (re-search-forward
              (rx "\\"
                  (group (or "\\" "begin" "end" "nonumber"))
                  (zero-or-one "{" (group (zero-or-more not-newline)) "}"))
```

```
              nil t)
        (let ((cmd (match-string 1))
              (env (match-string 2)))
          (cond ((string= cmd "begin")
                 (push (list :env (intern env)) ast))
                ((string= cmd "\\")
                 (let ((curr (pop ast)))
                   (push (plist-put curr :newline (1+ (or (plist-get curr :newline) 0))) ast)))
                ((string= cmd "nonumber")
                 (let ((curr (pop ast)))
                   (push (plist-put curr :nonumber (1+ (or (plist-get curr :nonumber) 0))) ast)))
                ((string= cmd "end")
                 (let ((child (pop ast))
                       (parent (pop ast)))
                   (push (plist-put parent :childs (cons child (plist-get parent :childs))) ast)))))))
    (plist-get (car ast) :childs)))

(defun +scimax-org-renumber-environment (orig-func &rest args)
  "A function to inject numbers in LaTeX fragment previews."
  (let ((results '())
        (counter -1))
    (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)
                     (cl-incf counter)
                     (let ((p (car (+parse-the-fun env))))
                       ;; Parse the `env', count new lines in the align env as equations, unless
                       (cl-incf counter (- (or (plist-get p :newline) 0)
                                           (or (plist-get p :nonumber) 0))))
                     (cons begin counter))
                    (t
                     (cons begin nil)))))
    (when-let ((number (cdr (assoc (point) results))))
      (setf (car args)
            (concat
             (format "\\setcounter{equation}{%s}\n" number)
             (car args)))))
  (apply orig-func args))

(defun +scimax-toggle-latex-equation-numbering (&optional enable)
  "Toggle whether LaTeX fragments are numbered."
  (interactive)
  (if (or enable (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.")))

(defun +scimax-org-inject-latex-fragment (orig-func &rest args)
  "Advice function to inject latex code before and/or after the equation in a latex fragment.
You can use this to set \\mathversion{bold} for example to make
it bolder. The way it works is by defining
:latex-fragment-pre-body and/or :latex-fragment-post-body in the
variable `org-format-latex-options'. These strings will then be
injected before and after the code for the fragment before it is
made into an image."
```

```
  (setf (car args)
        (concat
         (or (plist-get org-format-latex-options :latex-fragment-pre-body) "")
         (car args)
         (or (plist-get org-format-latex-options :latex-fragment-post-body) "")))
  (apply orig-func args))

(defun +scimax-toggle-inject-latex ()
  "Toggle whether you can insert latex in fragments."
  (interactive)
  (if (not (get '+scimax-org-inject-latex-fragment 'enabled))
      (progn
        (advice-add 'org-create-formula-image :around #'+scimax-org-inject-latex-fragment)
        (put '+scimax-org-inject-latex-fragment 'enabled t)
        (message "Inject latex enabled"))
    (advice-remove 'org-create-formula-image #'+scimax-org-inject-latex-fragment)
    (put '+scimax-org-inject-latex-fragment 'enabled nil)
    (message "Inject latex disabled")))

;; Enable renumbering by default
(+scimax-toggle-latex-equation-numbering t)
```

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

Large tables Use *Partial Horizontal Scroll* to display long tables without breaking them.

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

9.2.5 Bibliography

```
(setq bibtex-completion-bibliography +my/biblio-libraries-list
      bibtex-completion-library-path +my/biblio-storage-list
      bibtex-completion-notes-path +my/biblio-notes-path
      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))
```

```
(after! oc
  (setq org-cite-csl-styles-dir +my/biblio-styles-path)
        ;; org-cite-global-bibliography +my/biblio-libraries-list)

  (defun +org-ref-to-org-cite ()
    "Simple conversion of org-ref citations to org-cite syntax."
    (interactive)
    (save-excursion
      (goto-char (point-min))
      (while (re-search-forward "\\[cite\\(.*\\):\\([^]]*\\)\\]" nil t)
        (let* ((old (substring (match-string 0) 1 (1- (length (match-string 0)))))
               (new (s-replace "&" "@" old)))
          (message "Replaced citation %s with %s" old new)
          (replace-match new))))))
```

Org-cite

```
(after! citar
  (setq citar-library-paths +my/biblio-storage-list
        citar-notes-paths  (list +my/biblio-notes-path)
        citar-bibliography  +my/biblio-libraries-list
        citar-symbol-separator "  ")

  (when (display-graphic-p)
    (setq citar-symbols
          `((file ,(all-the-icons-octicon "file-pdf"      :face 'error) . " ")
            (note ,(all-the-icons-octicon "file-text"     :face 'warning) . " ")
            (link ,(all-the-icons-octicon "link-external" :face 'org-link) . " "))))

(use-package! citar-org-roam
  :after citar org-roam
  :no-require
  :config (citar-org-roam-mode)
  :init
  ;; Modified form: https://jethrokuan.github.io/org-roam-guide/
  (defun +org-roam-node-from-cite (entry-key)
    (interactive (list (citar-select-ref)))
    (let ((title (citar-format--entry
                  "${author editor} (${date urldate}) :: ${title}"
                  (citar-get-entry entry-key))))
      (org-roam-capture- :templates
                         '(("r" "reference" plain
                            "%?"
                            :if-new (file+head "references/${citekey}.org"
                                               ":properties:
:roam_refs: [cite:@${citekey}]
:end:
#+title: ${title}\n")
                            :immediate-finish t
                            :unnarrowed t))
                         :info (list :citekey entry-key)
                         :node (org-roam-node-create :title title)
                         :props '(:finalize find-file)))))
```

Citar

9.2.6 Exporting

General settings By default, Org only exports the first three levels of headings as *headings*, the rest is considered as paragraphs. Let's increase this to 5 levels.

```
(setq org-export-headline-levels 5)
```

Let's make use of the `:ignore:` tag from `ox-extra`, which provides a way to ignore exporting a heading, while exporting the content residing under it (different from `:noexport:`).

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

```
(setq org-export-creator-string
      (format "Made with Emacs %s and Org %s" emacs-version (org-release)))
```

LaTeX export

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

Compiling

```
;; 'svg' package depends on inkscape, imagemagik and ghostscript
(when (+all (mapcar 'executable-find '("inkscape" "magick" "gs")))
  (add-to-list 'org-latex-packages-alist '("" "svg")))

(add-to-list 'org-latex-packages-alist '("svgnames" "xcolor"))
;; (add-to-list 'org-latex-packages-alist '("" "fontspec")) ;; for xelatex
;; (add-to-list 'org-latex-packages-alist '("utf8" "inputenc"))
```

Org LaTeX packages

Export PDFs with syntax highlighting This is for code syntax highlighting in export. You need to use `-shell-escape` with latex, and install the `python-pygments` package.

```
;; Should be configured per document, as a local variable
;; (setq org-latex-listings 'minted)
;; (add-to-list 'org-latex-packages-alist '("" "minted"))

;; Default `minted` options, can be overwritten in file/dir locals
(setq org-latex-minted-options
      '(("frame"         "lines")
        ("fontsize"      "\\footnotesize")
        ("tabsize"       "2")
        ("breaklines"    "true")
        ("breakanywhere" "true") ;; break anywhere, no just on spaces
        ("style"         "default")
        ("bgcolor"       "GhostWhite")
```

```
        ("linenos"       "true")))

;; Link some org-mode blocks languages to lexers supported by minted
;; via (pygmentize), you can see supported lexers by running this command
;; in a terminal: `pygmentize -L lexers'
(dolist (pair '((ipython    "python")
                (jupyter    "python")
                (scheme     "scheme")
                (lisp-data  "lisp")
                (conf-unix  "unixconfig")
                (conf-space "unixconfig")
                (authinfo   "unixconfig")
                (gdb-script "unixconfig")
                (conf-toml  "yaml")
                (conf       "ini")
                (gitconfig  "ini")
                (systemd    "ini")))
  (unless (member pair org-latex-minted-langs)
    (add-to-list 'org-latex-minted-langs pair)))
```

```
(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
   '("lettre"
     "\\documentclass{lettre}"
     ("\\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
   '("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
   '("ieeeconf"
     "\\documentclass{ieeeconf}"
```

```
      ("\\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
   '("sagej"
     "\\documentclass{sagej}"
      ("\\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

Export multi-files Org documents Let's say we have a multi-files document, with `main.org` as the entry point. Supposing a document with a structure like this:

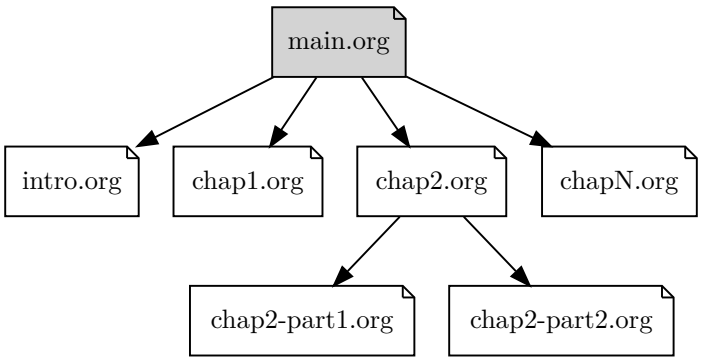


Figure 1: Example of a multi-files document structure

Files `intro.org`, `chap1.org`, … are included in `main.org` using the Org command . In such a setup, we will spend most of our time writing in a chapter files, and not the `main.org`, where when want to export the document, we would need to open the top-level file `main.org` before exporting.

A quick solution is **to admit the following convention**:

> If a file named `main.org` is present beside any other Org file, it should be considered as the entry point; and whenever we export to PDF (from any of the Org files like: `intro.org`, `chap1.org`, …), we automatically jump to the `main.org`, and run the export there.

This can be achieved by adding an Emacs-Lisp *advice* around the `(org-latex-export-to-pdf)` to switch to `main.org` (if it exists) before running the export.

You can also set the variable `+org-export-to-pdf-main-file` to the main file, in `.dir-locals.el` or as a file local variable.

```
(defvar +org-export-to-pdf-main-file nil
  "The main (entry point) Org file for a multi-files document.")

(advice-add
 'org-latex-export-to-pdf :around
 (lambda (orig-fn &rest orig-args)
   (message
    "PDF exported to: %s."
    (let ((main-file (or (bound-and-true-p +org-export-to-pdf-main-file) "main.org")))
      (if (file-exists-p (expand-file-name main-file))
          (with-current-buffer (find-file-noselect main-file)
            (apply orig-fn orig-args))
        (apply orig-fn orig-args))))))
```

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)
(setq org-hugo-auto-set-lastmod t)
```

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
    (unless (derived-mode-p 'org-mode)
      ;; 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 French apostrophes

```
(defun +helper--in-buffer-replace (old new)
  "Replace OLD with NEW in the current buffer."
  (save-excursion
    (goto-char (point-min))
    (let ((case-fold-search nil)
          (cnt 0))
      (while (re-search-forward old nil t)
        (replace-match new)
        (setq cnt (1+ cnt)))
      cnt)))

(defun +helper-clear-frenchy-ponctuations ()
  "Replace french apostrophes (’) by regular quotes (')."
  (interactive)
  (let ((chars '((" " . "") ("’" . "'")))
        (cnt 0))
    (dolist (pair chars)
      (setq cnt (+ cnt (+helper--in-buffer-replace (car pair) (cdr pair)))))
    (message "Replaced %d matche(s)." cnt)))
```

9.3.4 Yanking multi-lines paragraphs

```
(defun +helper-paragraphized-yank ()
  "Copy, then remove newlines and Org styling (/*_~)."
  (interactive)
  (copy-region-as-kill nil nil t)
  (with-temp-buffer
    (yank)
    ;; Remove newlines, and Org styling (/*_~)
    (goto-char (point-min))
    (let ((case-fold-search nil))
      (while (re-search-forward "[\n/*_~]" nil t)
        (replace-match (if (s-matches-p (match-string 0) "\n") " " "") t)))
    (kill-region (point-min) (point-max))))

(map! :localleader
      :map (org-mode-map markdown-mode-map latex-mode-map text-mode-map)
      :desc "Paragraphized yank" "y" #'+helper-paragraphized-yank)
```

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 -s "/tmp/emacs$UID/server" %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 "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'
ExecStop=emacsclient -s "/tmp/emacs$UID/server" --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 -s "/tmp/emacs$UID/server" --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 -s "/tmp/emacs$UID/server" --tty -create-frame --alternate-editor="/usr/bin/emacs" "${args[@]}"
else
  if ! $force_wait
  then
    args+=(--no-wait)
  fi

  emacsclient -s "/tmp/emacs$UID/server" -create-frame --alternate-editor="/usr/bin/emacs" "${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 Oh-my-Zsh

10.6.1 Path

Path to your oh-my-zsh installation.

```
export ZSH="$HOME/.oh-my-zsh"
```

10.6.2 Themes and customization:

Set name of the theme to load, if set to `"random"`, it will load a random theme each time oh-my-zsh is loaded, in which case, to know which specific one was loaded, run: `echo $RANDOM_THEME` See github.com/ohmyzsh/ohmyzsh/wiki/Themes.

```
# Typewritten customizations
TYPEWRITTEN_RELATIVE_PATH="adaptive"
TYPEWRITTEN_CURSOR="underscore"

ZSH_THEME="typewritten/typewritten"

# Set list of themes to pick from when loading at random
# Setting this variable when ZSH_THEME=random will cause zsh to load
# a theme from this variable instead of looking in $ZSH/themes/
# If set to an empty array, this variable will have no effect.
# ZSH_THEME_RANDOM_CANDIDATES=( "robbyrussell" "agnoster" )
```

10.6.3 Behavior

```
# Uncomment the following line to use case-sensitive completion.
# CASE_SENSITIVE="true"

# Uncomment the following line to use hyphen-insensitive completion.
# Case-sensitive completion must be off. _ and - will be interchangeable.
# HYPHEN_INSENSITIVE="true"

# Uncomment the following line to disable bi-weekly auto-update checks.
# DISABLE_AUTO_UPDATE="true"

# Uncomment the following line to automatically update without prompting.
DISABLE_UPDATE_PROMPT="true"

# Uncomment the following line to change how often to auto-update (in days).
export UPDATE_ZSH_DAYS=3

# Uncomment the following line if pasting URLs and other text is messed up.
# DISABLE_MAGIC_FUNCTIONS="true"

# Uncomment the following line to disable colors in ls.
# DISABLE_LS_COLORS="true"

# Uncomment the following line to disable auto-setting terminal title.
# DISABLE_AUTO_TITLE="true"

# Uncomment the following line to enable command auto-correction.
# ENABLE_CORRECTION="true"

# Uncomment the following line to display red dots whilst waiting for completion.
# COMPLETION_WAITING_DOTS="true"

# Uncomment the following line if you want to disable marking untracked files
# under VCS as dirty. This makes repository status check for large repositories
# much, much faster.
# DISABLE_UNTRACKED_FILES_DIRTY="true"
```

```
36
37 # Uncomment the following line if you want to change the command execution time
38 # stamp shown in the history command output.
39 # You can set one of the optional three formats:
40 # "mm/dd/yyyy"|"dd.mm.yyyy"|"yyyy-mm-dd"
41 # or set a custom format using the strftime function format specifications,
42 # see 'man strftime' for details.
43 # HIST_STAMPS="mm/dd/yyyy"
```

10.6.4 Plugins

```
1  # Would you like to use another custom folder than $ZSH/custom?
2  ZSH_CUSTOM=$HOME/.config/my_ohmyzsh_customizations
3
4  # Which plugins would you like to load?
5  # Standard plugins can be found in $ZSH/plugins/
6  # Custom plugins may be added to $ZSH_CUSTOM/plugins/
7  # Example format: plugins=(rails git textmate ruby lighthouse)
8  # Add wisely, as too many plugins slow down shell startup.
9  plugins=(
10   zsh-autosuggestions
11   zsh-navigation-tools
12   zsh-interactive-cd
13   archlinux
14   ssh-agent
15   sudo
16   docker
17   systemd
18   tmux
19   python
20   pip
21   rust
22   repo
23   git
24   cp
25   rsync
26   ripgrep
27   fzf
28   fd
29   z
30 )
```

10.6.5 Bootstrap Oh-my-Zsh

```
1  source $ZSH/oh-my-zsh.sh
```

10.6.6 Aliases

```
1  # Aliases
2  alias zshconfig="vim ~/.zshrc"
3  alias ohmyzsh="ranger $ZSH"
```

10.7 Zsh user configuration

10.7.1 `pbcopy` and `pbpaste`

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
```

10.7.2 `netpaste`

Define a `netpaste` command to paste to a Pastebin server.

```
alias netpaste='curl -F file=@- 0x0.st' # OR 'curl -F f:1=<- ix.io '
```

10.7.3 Sudo GUI!

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
```

10.7.4 Neovim

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
```

10.7.5 ESP-IDF

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
```

10.7.6 CLI wttrin client

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'
```

10.7.7 Minicom

Enable Meta key and colors in `minicom`:

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

10.7.8 Rust

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.

10.7.9 Clang-format

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

10.7.10 CMake

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

```
export CMAKE_PREFIX_PATH=$HOME/Softwares/src/install
export PATH=$PATH:$HOME/Softwares/src/install/bin
```

10.7.11 Node

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…)

10.7.12 tmux

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
```

10.7.13 Other stuff

```
# You may need to manually set your language environment
# export LANG=en_US.UTF-8

# Preferred editor for local and remote sessions
# if [[ -n $SSH_CONNECTION ]]; then
#   export EDITOR='vim'
# else
#   export EDITOR='mvim'
# fi

# Compilation flags
# export ARCHFLAGS="-arch x86_64"

# 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/bin to path (for DOOM Emacs stuff)
export PATH=$PATH:$HOME/.config/emacs/bin

export TEXMFHOME=$HOME/.texmf
```

Define some environment variables.

```
export DS_DIR=~/PhD/datasets-no/experiment_images/
export DSO_BIN_DIR=~/PhD/workspace-no/vo/orig/dso/build/release/bin
export DSO_RES_DIR=~/PhD/workspace-no/vo/orig/dso_results
```

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

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

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"
```

```
28
29  # First, silently try to perform the mount, which would succeed if the appropriate
30  # key is available in the keyring
31  if /sbin/mount.ecryptfs_private >/dev/null 2>&1
32  then
33    exit 0
34  fi
35
36  # Otherwise, interactively prompt for the user's password
37  if [ -f "$WRAPPED_PASSPHRASE_FILE" -a -f "$MOUNT_PASSPHRASE_SIG_FILE" ]
38  then
39    tries=0
40
41    while [ $tries -lt $PW_ATTEMPTS ]
42    do
43      LOGINPASS=`zenity --password --title "eCryptFS: $MESSAGE"`
44      if [ $(wc -l < "$MOUNT_PASSPHRASE_SIG_FILE") = "1" ]
45      then
46        # No filename encryption; only insert fek
47        if printf "%s\0" "$LOGINPASS" | ecryptfs-unwrap-passphrase "$WRAPPED_PASSPHRASE_FILE" - |
          ↪  ecryptfs-add-passphrase -
48        then
49          break
50        else
51          zenity --error --title "eCryptfs" --text "Error: Your passphrase is incorrect"
52          tries=$(($tries + 1))
53          continue
54        fi
55      else
56        if printf "%s\0" "$LOGINPASS" | ecryptfs-insert-wrapped-passphrase-into-keyring
          ↪  "$WRAPPED_PASSPHRASE_FILE" -
57        then
58          break
59        else
60          zenity --error --title "eCryptfs" --text "Error: Your passphrase is incorrect"
61          tries=$(($tries + 1))
62          continue
63        fi
64      fi
65    done
66
67    if [ $tries -ge $PW_ATTEMPTS ]
68    then
69      zenity --error --title "eCryptfs" --text "Too many incorrect password attempts, exiting"
70      exit 1
71    fi
72
73    /sbin/mount.ecryptfs_private
74  else
75    zenity --error --title "eCryptfs" --text "Encrypted private directory is not setup properly"
76    exit 1
77  fi
78
79  if grep -qs "$HOME/.Private $PWD ecryptfs " /proc/mounts 2>/dev/null; then
80    zenity --info --title "eCryptfs" --text "Your private directory has been mounted."
81  fi
82
83  dolphin "$HOME/Private"
84  exit 0
```

10.9.2 Desktop integration

```
1  [Desktop Entry]
2  Type=Application
3  Version=1.0
4  Name=eCryptfs Unlock Private Directory
5  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 it won't evolve too much, as it is recommended to keep the `.gdbinit` clean and simple. For the moment, it does just enable pretty printing, and defines the `c` and `n` commands to wrap `continue` and `next` with a post `refresh`, which is helpful with 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

# When debugging my apps, debug information of system libraries
# aren't that important
set debuginfod enabled off

# Set pretty print
set print pretty on

# I hate stepping into system libraries when I'm debugging my
# crappy stuff, so lets add system headers to skipped files
skip pending on
python
import os

# Add paths here, they will be explored recursivly
LIB_PATHS = ["/usr/include" "/usr/local/include"]

for lib_path in LIB_PATHS:
  for root, dirs, files in os.walk(lib_path):
    for file in files:
      cmd = f"skip file {os.path.join(root, file)}"
      gdb.execute(cmd, True, to_string=True)
end
skip enable

skip pending on
guile
<<gdb-init-guile>>
end
skip enable

# 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
```

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

# As I'm using KDE, use Qt based pinentry tool instead of default GTK+
pinentry-program /usr/bin/pinentry-qt

# Allow pinentry in Emacs minibuffer (combined with epg-pinentry-mode)
allow-loopback-pinentry
allow-emacs-pinentry
```

10.12 OCR This

```
#!/bin/bash

IMG=$(mktemp -u --suffix=".png")
scrot -s "$IMG" -q 100
mogrify -modulate 100,0 -resize 400% "$IMG"
tesseract "$IMG" - -l eng 2> /dev/null | xsel -ib
```

10.13 Slack

This script is called at system startup.

```
#!/bin/bash

WEEK_DAY=$(date +%u)
HOUR=$(date +%H)
SLACK=$(which slack)

if [ ! "$WEEK_DAY" == "6" ] && [ ! "$WEEK_DAY" == "7" ] && [ "$HOUR" -gt 7 ] && [ "$HOUR" -lt 20 ] ; then
  $SLACK -u %U
else
  echo "It is not work time!"
fi
```

10.14 Arch Linux packages

Here, we install Arch packages

```
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
}

PKGS_LIST=(
    git ripgrep fd gnupg fzf the_silver_searcher
    ttf-ibm-plex ttf-fira-code ttf-roboto-mono ttf-overpass ttf-lato ttf-input
    ttf-cascadia-code ttf-jetbrains-mono ttf-fantasque-sans-mono
    ttc-iosevka ttf-iosevka-nerd ttc-iosevka-slab ttc-iosevka-curly
    ttc-iosevka-curly-slab ttc-iosevka-etoile ttc-iosevka-ss09
    ccls cppcheck clang gcc gdb lldb valgrind rr openocd
    sbcl cmucl clisp chez-scheme mit-scheme chibi-scheme chicken
    vls vlang rustup semgrep-bin
    mu isync msmtp xsel xorg-xhost
    mpc mpv mpd vlc yt-dlp
    maxima fricas octave scilab-bin graphviz jupyterlab jupyter-notebook r
    djvulibre catdoc unrtf perl-image-exiftool wkhtmltopdf
    chezmoi neovim repo ecryptfs-utils
    pandoc hugo inkscape imagemagick
    aspell aspell-en aspell-fr aspell-ar grammalecte language-tool ltex-ls-bin
    libvterm brave zotero bitwarden-cli binutils
    poppler ffmpegthumbnailer mediainfo imagemagick tar unzip
)

for PKG in "${PKGS_LIST[@]}"; do
    check_and_install_pkg "$PKG"
done
```

10.15 KDE Plasma

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

```
export PLASMA_USE_QT_SCALING=1
```