

# Doom Emacs Configuration

Emacs configuration for work and life!

Abdelhak Bougouffa\*

September 11, 2022

## Contents

<b>1</b>	<b>This repository</b>	<b>5</b>
1.1	How to install . . . . .	6
1.2	Emacs stuff . . . . .	6
<b>2</b>	<b>Intro</b>	<b>6</b>
2.1	This file . . . . .	7
<b>3</b>	<b>Doom configuration files</b>	<b>7</b>
3.1	Pseudo early-init . . . . .	7
3.1.1	Useful functions . . . . .	7
3.1.2	Fixes . . . . .	9
3.1.3	Check for external tools . . . . .	10
3.2	Doom modules ( <code>init.el</code> ) . . . . .	10
3.2.1	File skeleton . . . . .	10
3.2.2	Input ( <code>:input</code> ) . . . . .	11
3.2.3	General ( <code>:config</code> ) . . . . .	11
3.2.4	Completion ( <code>:completion</code> ) . . . . .	11
3.2.5	User interface ( <code>:ui</code> ) . . . . .	11
3.2.6	Editor ( <code>:editor</code> ) . . . . .	12
3.2.7	Emacs builtin stuff ( <code>:emacs</code> ) . . . . .	12
3.2.8	Terminals ( <code>:term</code> ) . . . . .	12
3.2.9	Checkers ( <code>:checkers</code> ) . . . . .	12
3.2.10	Tools ( <code>:tools</code> ) . . . . .	13
3.2.11	Operating system ( <code>:os</code> ) . . . . .	13
3.2.12	Language support ( <code>:lang</code> ) . . . . .	13
3.2.13	Email ( <code>:email</code> ) . . . . .	14
3.2.14	Apps ( <code>:app</code> ) . . . . .	14
3.2.15	Private . . . . .	14
3.3	Additional packages ( <code>packages.el</code> ) . . . . .	14
<b>4</b>	<b>General Emacs settings</b>	<b>14</b>
4.1	User information . . . . .	14
4.2	Common variables . . . . .	14
4.3	Secrets . . . . .	15
4.4	Better defaults . . . . .	15
4.4.1	File deletion . . . . .	15
4.4.2	Window . . . . .	15
4.4.3	Messages buffer . . . . .	15
4.4.4	Undo and auto-save . . . . .	16

---

\*abougouffa@fedoraproject.org

4.4.5	Editing . . . . .	17
4.4.6	Emacs sources . . . . .	17
4.4.7	Frame . . . . .	17
4.4.8	Browsers . . . . .	17
<b>5</b>	<b>Emacs daemon . . . . .</b>	<b>18</b>
5.1	Initialization . . . . .	18
5.2	Tweaks . . . . .	18
5.2.1	Save recent files . . . . .	18
<b>6</b>	<b>Package configuration . . . . .</b>	<b>18</b>
6.1	User interface . . . . .	18
6.1.1	Font . . . . .	18
6.1.2	Theme . . . . .	19
6.1.3	Modeline . . . . .	21
6.1.4	Set transparency . . . . .	21
6.1.5	Dashboard . . . . .	22
6.1.6	Which key . . . . .	22
6.1.7	Window title . . . . .	22
6.1.8	SVG tag and <code>svg-lib</code> . . . . .	23
6.1.9	Focus . . . . .	23
6.1.10	Scrolling . . . . .	24
6.1.11	All the icons . . . . .	24
6.1.12	Tabs . . . . .	24
6.2	Highlight indent guides . . . . .	24
6.3	Editing . . . . .	24
6.3.1	File templates . . . . .	24
6.3.2	Scratch buffer . . . . .	25
6.3.3	Mouse buttons . . . . .	25
6.3.4	Very large files . . . . .	25
6.3.5	Evil . . . . .	25
6.3.6	Aggressive indent . . . . .	25
6.3.7	YASnippet . . . . .	26
6.4	Completion & IDE . . . . .	26
6.4.1	Company . . . . .	26
6.4.2	Treemacs . . . . .	27
6.4.3	Projectile . . . . .	28
6.4.4	Tramp . . . . .	28
6.4.5	Eros-eval . . . . .	28
6.4.6	<code>dir-locals.el</code> . . . . .	28
6.4.7	Language Server Protocol . . . . .	29
6.4.8	Cppcheck . . . . .	32
6.4.9	Project CMake . . . . .	32
6.4.10	Clang-format . . . . .	32
6.4.11	Auto-include C++ headers . . . . .	32
6.4.12	C/C++ preprocessor conditions . . . . .	33
6.4.13	Erefactor . . . . .	33
6.4.14	Lorem ipsum . . . . .	33
6.4.15	Coverage test . . . . .	33
6.5	Debugging . . . . .	33
6.5.1	DAP . . . . .	33
6.5.2	RealGUD . . . . .	34
6.5.3	GDB . . . . .	36
6.5.4	WIP <code>launch.json</code> support for GUD and RealGUD . . . . .	38
6.5.5	Valgrind . . . . .	42
6.6	Symbols . . . . .	42

6.6.1	Emojify . . . . .	42
6.6.2	Ligatures . . . . .	43
6.7	Checkers (spell & grammar) . . . . .	43
6.7.1	Spell-Fu . . . . .	43
6.7.2	Grammarly . . . . .	44
6.7.3	Grammalecte . . . . .	45
6.7.4	LT <sub>E</sub> X/LanguageTool . . . . .	46
6.7.5	Go Translate (Google, Bing and DeepL) . . . . .	46
6.8	System tools . . . . .	48
6.8.1	Disk usage . . . . .	48
6.8.2	Chezmoi . . . . .	48
6.8.3	Aweshell . . . . .	49
6.8.4	Lemon . . . . .	49
6.8.5	eCryptfs . . . . .	49
6.9	Features . . . . .	50
6.9.1	Workspaces . . . . .	50
6.9.2	Weather . . . . .	51
6.9.3	OpenStreetMap . . . . .	51
6.9.4	Islamic prayer times . . . . .	51
6.9.5	Info colors . . . . .	52
6.9.6	Zotero Zotxt . . . . .	52
6.9.7	CRDT . . . . .	52
6.9.8	The Silver Searcher . . . . .	53
6.9.9	Page break lines . . . . .	53
6.9.10	Emacs Application Framework . . . . .	53
6.9.11	Bitwarden . . . . .	56
6.9.12	PDF tools . . . . .	57
6.9.13	LTDR . . . . .	57
6.9.14	FZF . . . . .	58
6.9.15	Binary files . . . . .	58
6.9.16	Objdump mode . . . . .	59
6.10	Fun . . . . .	59
6.10.1	Speed Type . . . . .	59
6.10.2	2048 Game . . . . .	60
6.10.3	Snow . . . . .	60
6.10.4	xkcd . . . . .	60
<b>7</b>	<b>Applications</b> . . . . .	<b>60</b>
7.1	Calendar . . . . .	60
7.2	e-Books (nov) . . . . .	60
7.3	News feed (elfeed) . . . . .	61
7.4	VPN configuration . . . . .	62
7.4.1	NetExtender wrapper . . . . .	62
7.4.2	Emacs + NetExtender . . . . .	62
7.5	Email (mu4e) . . . . .	63
7.5.1	IMAP (mbsync) . . . . .	63
7.5.2	SMTP (msmtp) . . . . .	65
7.5.3	Mail client and indexer (mu and mu4e) . . . . .	66
7.5.4	Dashboard . . . . .	69
7.5.5	Save all attachements . . . . .	70
7.6	IRC . . . . .	71
7.7	Multimedia . . . . .	71
7.7.1	MPD and MPC . . . . .	71
7.7.2	EMMS . . . . .	72
7.7.3	EMPV . . . . .	73
7.7.4	Keybindings . . . . .	75

7.7.5	Cycle song information in mode line . . . . .	75
7.8	Maxima . . . . .	76
7.8.1	Maxima . . . . .	76
7.8.2	IMaxima . . . . .	76
7.9	FriCAS . . . . .	77
7.10	Roam . . . . .	77
7.10.1	Basic settings . . . . .	77
7.10.2	Mode line file name . . . . .	77
7.10.3	Org Roam Capture template . . . . .	78
7.10.4	View notes in Deft . . . . .	78
<b>8</b>	<b>Programming</b>	<b>78</b>
8.1	CSV rainbow . . . . .	78
8.2	Vimrc . . . . .	79
8.3	ESS . . . . .	79
8.4	Python IDE . . . . .	79
8.5	GNU Octave . . . . .	79
8.6	ROS . . . . .	80
8.6.1	Extensions . . . . .	80
8.6.2	ROS bags . . . . .	80
8.6.3	<code>ros.el</code> . . . . .	81
8.7	Scheme . . . . .	81
8.8	Embedded systems . . . . .	81
8.8.1	Embed.el . . . . .	81
8.8.2	Arduino . . . . .	82
8.8.3	Bitbake (Yocto) . . . . .	82
8.9	Git & VC . . . . .	82
8.9.1	Magit . . . . .	82
8.9.2	Repo . . . . .	83
8.9.3	Blamer . . . . .	84
8.10	Assembly . . . . .	84
8.11	Disaster . . . . .	85
8.12	Devdocs . . . . .	85
8.13	Systemd . . . . .	85
8.14	PKGBUILD . . . . .	86
8.15	Franca IDL . . . . .	86
8.16	$\text{\LaTeX}$ . . . . .	86
8.17	Flycheck + Projectile . . . . .	86
8.18	Graphviz . . . . .	86
8.19	Modula-II . . . . .	87
8.20	Mermaid . . . . .	87
8.21	The V Programming Language . . . . .	87
8.22	Inspector . . . . .	87
<b>9</b>	<b>Office</b>	<b>88</b>
9.1	Org additional packages . . . . .	88
9.2	Org mode . . . . .	88
9.2.1	Intro . . . . .	88
9.2.2	Behavior . . . . .	89
9.2.3	Custom links . . . . .	99
9.2.4	Visuals . . . . .	100
9.2.5	Bibliography . . . . .	106
9.2.6	Exporting . . . . .	108
9.3	Text editing . . . . .	111
9.3.1	Plain text . . . . .	111
9.3.2	Academic phrases . . . . .	112

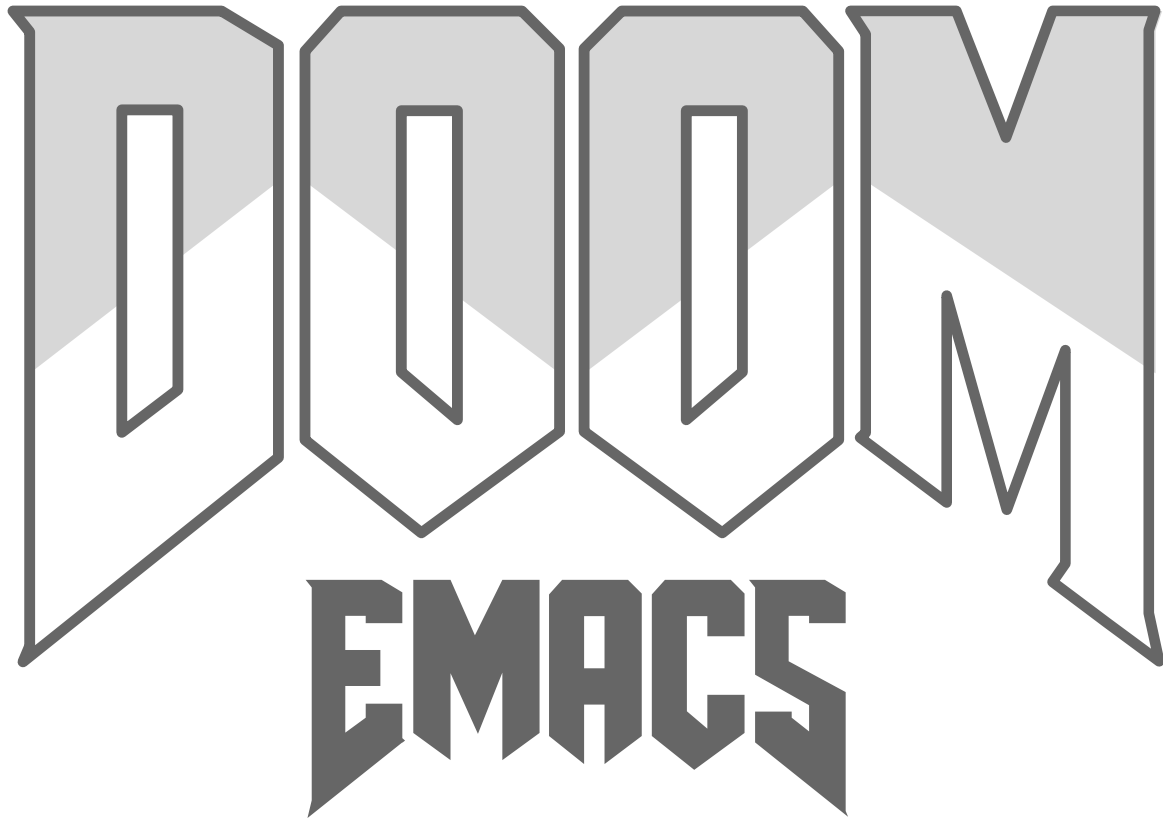
9.3.3	French apostrophes . . . . .	112
9.3.4	Yanking multi-lines paragraphs . . . . .	112
<b>10</b>	<b>System configuration</b>	<b>113</b>
10.1	Mime types . . . . .	113
10.1.1	Org mode files . . . . .	113
10.1.2	Registering <code>org-protocol://</code> . . . . .	113
10.1.3	Configuring Chrome/Brave . . . . .	113
10.2	Git . . . . .	114
10.2.1	Git diffs . . . . .	114
10.2.2	Apache Tika App wrapper . . . . .	116
10.3	Emacs' Systemd daemon . . . . .	117
10.4	Emacs client . . . . .	118
10.4.1	Desktop integration . . . . .	118
10.4.2	Command-line wrapper . . . . .	118
10.5	AppImage . . . . .	120
10.6	Oh-my-Zsh . . . . .	121
10.6.1	Path . . . . .	121
10.6.2	Themes and customization: . . . . .	121
10.6.3	Behavior . . . . .	121
10.6.4	Plugins . . . . .	122
10.6.5	Bootstrap Oh-my-Zsh . . . . .	122
10.6.6	Aliases . . . . .	123
10.7	Zsh user configuration . . . . .	123
10.7.1	<code>pbcopy</code> and <code>pbpaste</code> . . . . .	123
10.7.2	<code>netpaste</code> . . . . .	123
10.7.3	Sudo GUI! . . . . .	123
10.7.4	Neovim . . . . .	123
10.7.5	ESP-IDF . . . . .	124
10.7.6	CLI wttrin client . . . . .	124
10.7.7	Minicom . . . . .	124
10.7.8	Rust . . . . .	124
10.7.9	Clang-format . . . . .	124
10.7.10	CMake . . . . .	124
10.7.11	Node . . . . .	125
10.7.12	tmux . . . . .	125
10.7.13	Other stuff . . . . .	125
10.8	Rust format . . . . .	126
10.9	eCryptfs . . . . .	126
10.9.1	Unlock and mount script . . . . .	126
10.9.2	Desktop integration . . . . .	128
10.10	GDB . . . . .	128
10.10.1	Early init . . . . .	128
10.10.2	Init . . . . .	128
10.11	GnuPG . . . . .	129
10.12	OCR This . . . . .	129
10.13	Arch Linux packages . . . . .	129
10.14	KDE Plasma . . . . .	130

## 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.

```
1 sudo pacman -S chezmoi
2 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`:

```
1 git clone https://github.com/doomemacs/doomemacs.git ~/.config/emacs
2
3 ~/.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 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).

```
1  ;;; 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.

```
1  #!/bin/bash
2
3  # This is an automatically generated setup file, it installes some missing
4  # dependencies, configure system services, set system settings form better
5  # desktop integration... etc.
6  # Abdelhak BOUGOUFFA (c) 2022
```

Add an initial comment to the `~/.zshrc` file.

```
1  # -*- mode: sh; -*-
2
3  # This file is automatically generated from my Org literate configuration.
4  # 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.

```
1  ;;; pseudo-early-init.el -*- coding: utf-8-unix; lexical-binding: t; -*-
2
```

#### 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, particularly in early stage, so let's define here.

```
1  ;;; === Primitives ===
2
3  ;; (+bool "someval") ;; ==> t
4  (defun +bool (val) (not (null val)))
5
```

```

6   ;;; === Higher order functions ===
7
8   ;; (+foldr (lambda (a b) (message "%d + %d" a b) (+ a b)) 0 '(1 2 3 4 5)) ;; ==> 15
9   ;; (5 + 0) -> (4 + 5) -> (3 + 9) -> (2 + 12) --> (1 + 14)
10  (defun +foldr (fun acc seq)
11    (if (null seq) acc
12        (funcall fun (car seq) (+foldr fun acc (cdr seq)))))
13
14  ;; (+foldl (lambda (a b) (message "%d + %d" a b) (+ a b)) 0 '(1 2 3 4 5)) ;; ==> 15
15  ;; (0 + 1) -> (1 + 2) -> (3 + 3) -> (6 + 4) -> (10 + 5)
16  (defun +foldl (fun acc seq)
17    (if (null seq) acc
18        (+foldl fun (funcall fun acc (car seq)) (cdr seq))))
19
20  ;; (+all '(83 88 t "txt")) ;; ==> t
21  (defun +all (seq)
22    (+foldr (lambda (r l) (and r l)) t seq))
23
24  ;; (+some '(nil nil "text" nil 2)) ;; ==> t
25  (defun +some (seq)
26    (+bool (+foldr (lambda (r l) (or r l)) nil seq)))
27
28  ;; (+filter 'stringp '("A" 2 "C" nil 3)) ;; ==> ("A" "C")
29  (defun +filter (fun seq)
30    (when seq
31      (let ((head (car seq))
32            (tail (cdr seq)))
33        (if (funcall fun head)
34            (cons head (+filter fun tail))
35            (+filter fun tail)))))
36
37  ;; (+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"))
38  (defun +zip (&rest seqs)
39    (if (null (car seqs)) nil
40        (cons (mapcar #'car seqs)
41              (apply #'zip (mapcar #'cdr seqs)))))
42
43  ;;; === Strings ===
44
45  ;; (+str-join ", " '("foo" "10" "bar")) ;; ==> "foo, 10, bar"
46  (defun +str-join (sep seq)
47    (+foldl (lambda (l r) (concat l sep r))
48            (car seq) (cdr seq)))
49
50  ;; (+str-split "foo, 10, bar" ", ") ;; ==> ("foo" "10" "bar")
51  (defun +str-split (str sep)
52    (let ((s (string-search sep str)))
53      (if s (cons (substring str 0 s)
54                  (+str-split (substring str (+ s (length sep))) sep))
55          (list str))))
56
57  (defun +str-replace (old new s)
58    "Replaces OLD with NEW in S."
59    (replace-regexp-in-string (regexp-quote old) new s t))
60
61  (defun +str-replace-all (replacements s)
62    "REPLACEMENTS is a list of cons-cells. Each `car` is replaced with `cdr` in S."
63    (replace-regexp-in-string (regexp-opt (mapcar 'car replacements))
64                              (lambda (it) (cdr (assoc-string it replacements)))
65                              s 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 "\\(?:-\\(?:[: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)
8      (while (not (or unlocked (zerop try-again)))
9        (setq unlocked (zerop (shell-command "gpg -q --no-tty --logger-file /dev/null --batch -d ~/.authinfo.gpg
↪ > /dev/null" nil nil)))
10       try-again (1- try-again))
11      (unless unlocked
12        (message "GPG: failed to unlock, please try again (%d)" try-again)))
13      (unless unlocked ;; Exit Emacs, systemd will restart it
14        (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.

```

1 (defconst EAF-DIR (expand-file-name "eaf/eaf-repo" doom-data-dir))
2 (defconst IS-LUCID (string-search "LUCID" system-configuration-features))
3 (defconst FRICAS-DIR "/usr/lib/fricas/emacs")
4
5 (defconst AG-P (executable-find "ag"))
6 (defconst EAF-P (and (not IS-LUCID) (file-directory-p EAF-DIR)))
7 (defconst MPD-P (+all (mapcar #'executable-find '("mpc" "mpd"))))
8 (defconst MPV-P (executable-find "mpv"))
9 (defconst REPO-P (executable-find "repo"))
10 (defconst FRICAS-P (and (executable-find "fricas") (file-directory-p FRICAS-DIR)))
11 (defconst MAXIMA-P (executable-find "maxima"))
12 (defconst TUNTOX-P (executable-find "tuntox"))
13 (defconst ROSBAG-P (executable-find "rosbag"))
14 (defconst ZOTERO-P (executable-find "zotero"))
15 (defconst CHEZMOI-P (executable-find "chezmoi"))
16 (defconst OBJDUMP-P (executable-find "objdump"))
17 (defconst STUNNEL-P (executable-find "stunnel"))
18 (defconst ECRYPTFS-P (+all (mapcar #'executable-find '("ecryptfs-add-passphrase"
19   ↪ "/sbin/mount.ecryptfs_private"))))
20 (defconst BITWARDEN-P (executable-find "bw"))
21 (defconst YOUTUBE-DL-P (+some (mapcar #'executable-find '("yt-dlp" "youtube-dl"))))
22 (defconst NETEXTENDER-P (and (executable-find "netExtender") (+all (mapcar #'file-exists-p
23   ↪ '("~/local/bin/netextender" "~/ssh/sslvpn.gpg")))))
24 (defconst CLANG-FORMAT-P (executable-find "clang-format"))
25 (defconst LANGUAGE-TOOL-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>>`.

```

1 ;; init.el -*- coding: utf-8-unix; lexical-binding: t; -*-
2
3 ;; This file controls what Doom modules are enabled and what order they load in.
4 ;; Press 'K' on a module to view its documentation, and 'gd' to browse its directory.
5
6 ;; I add some special stuff wich I want to load very early.
7 (load! "pseudo-early-init.el")
8
9 (doom!
10   :input
11   <<doom-input>>
12
13   :completion
14   <<doom-completion>>
15
16   :ui
17   <<doom-ui>>
18
19   :editor
20   <<doom-editor>>
21
22   :emacs
23   <<doom-emacs>>

```

```

24
25 :term
26 <<doom-term>>
27
28 :checkers
29 <<doom-checkers>>
30
31 :tools
32 <<doom-tools>>
33
34 :os
35 <<doom-os>>
36
37 :lang
38 <<doom-lang>>
39
40 :email
41 <<doom-email>>
42
43 :app
44 <<doom-app>>
45
46 :config
47 <<doom-config>>
48
49 :private
50 <<doom-private>>
51 )

```

### 3.2.2 Input (:input)

Enable bidirectional languages support (*bidi*).

```

1  bidi

```

### 3.2.3 General (:config)

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

```

1  literate
2  (default +bindings +smartparens)

```

### 3.2.4 Completion (:completion)

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

```

1  (vertico +icons)
2  (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.

```

1  zen
2  deft
3  doom
4  hydra
5  hl-todo

```

```

6  ophints
7  modeline
8  nav-flash
9  workspaces
10 indent-guides
11 doom-dashboard
12 (treemacs +lsp)
13 (ligatures +extra)
14 (popup +all +defaults)
15 (emoji +ascii +github)
16 (window-select +numbers)
17 (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.

```

1  (evil +everywhere)
2  file-templates
3  fold
4  format
5  multiple-cursors
6  parinfer
7  snippets
8  word-wrap

```

### 3.2.7 Emacs builtin stuff (:emacs)

Beautify Emacs builtin packages.

```

1  vc
2  undo
3  (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).

```

1  term
2  vterm
3  shell
4  eshell

```

### 3.2.9 Checkers (:checkers)

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

```

1  (syntax +childframe)
2  (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.

```

1  ein
2  pdf
3  rgb
4  gist
5  make
6  tmux
7  direnv
8  upload
9  biblio
10 tree-sitter
11 editorconfig
12 (lsp +peek)
13 (docker +lsp)
14 (magit +forge)
15 (debugger +lsp)
16 (eval +overlay)
17 (lookup +docsets +dictionary +offline)

```

### 3.2.11 Operating system (:os)

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

```

1  (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.

```

1  qt
2  data
3  plantuml
4  emacs-lisp
5  common-lisp
6  (ess +lsp)
7  (yaml +lsp)
8  (markdown +grip)
9  (csharp +dotnet)
10 (racket +lsp +xp)
11 (lua +lsp +fennel)
12 (web +tree-sitter)
13 (latex +lsp +latexmk)
14 (cc +lsp +tree-sitter)
15 (sh +lsp +tree-sitter)
16 (json +lsp +tree-sitter)
17 (rust +lsp +tree-sitter)
18 (julia +lsp +tree-sitter)
19 (python +lsp +pyenv +pyright +tree-sitter)
20 (scheme +chez +mit +chicken +gauche +guile +chibi)
21 (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.

```
1 (: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`.

```
1 irc
2 rss
3 emms
4 calendar
5 everywhere
```

### 3.2.15 Private

```
1 (grammar +lsp)
2 (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.

```
1 ;; -*- coding: utf-8-unix; no-byte-compile: t; -*-
```

## 4 General Emacs settings

### 4.1 User information

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

### 4.2 Common variables

```
1 (defvar +my/lang-mother-tongue "ar")
2 (defvar +my/lang-main "en")
3 (defvar +my/lang-secondary "fr")
4
5 (defvar +my/biblio-libraries-list (list (expand-file-name "~/Zotero/library.bib")))
6 (defvar +my/biblio-storage-list (list (expand-file-name "~/Zotero/storage/")))
7 (defvar +my/biblio-notes-path (expand-file-name "~/PhD/bibliography/notes/"))
8 (defvar +my/biblio-styles-path (expand-file-name "~/Zotero/styles/"))
9
10 ;; Set it early, to avoid creating "~/org" at startup
11 (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.

```
1 (setq auth-sources '("~/authinfo.gpg")
2   auth-source-do-cache t
3   auth-source-cache-expiry 86400 ; All day, default is 2h (7200)
4   password-cache t
5   password-cache-expiry 86400)
6
7 (after! epa
8   (setq-default epa-file-encrypt-to '("F808A020A3E1AC37")))
```

## 4.4 Better defaults

### 4.4.1 File deletion

Delete files by moving them to trash.

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

### 4.4.2 Window

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

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

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

```
1 (setq evil-vsplt-window-right t
2   evil-split-window-below t)
```

Show list of buffers when splitting.

```
1 (defadvice! prompt-for-buffer (&rest _)
2   :after '(evil-window-split evil-window-vsplt)
3   (consult-buffer))
```

### 4.4.3 Messages buffer

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

```
1 (defvar +messages--auto-tail-enabled nil)
2
3 (defun +messages--auto-tail-a (&rest arg)
4   "Make *Messages* buffer auto-scroll to the end after each message."
5   (let* ((buf-name (buffer-name (messages-buffer)))
6         ;; Create *Messages* buffer if it does not exist
7         (buf (get-buffer-create buf-name)))
8     ;; Activate this advice only if the point is _not_ in the *Messages* buffer
9     ;; to begin with. This condition is required; otherwise you will not be
10    ;; able to use `isearch' and other stuff within the *Messages* buffer as
11    ;; the point will keep moving to the end of buffer :P
12    (when (not (string= buf-name (buffer-name)))
13      ;; Go to the end of buffer in all *Messages* buffer windows that are
14      ;; *live* (`get-buffer-window-list' returns a list of only live windows).
15      (dolist (win (get-buffer-window-list buf-name nil :all-frames))
```

```

16      (with-selected-window win
17        (goto-char (point-max))))
18      ;; Go to the end of the *Messages* buffer even if it is not in one of
19      ;; the live windows.
20      (with-current-buffer buf
21        (goto-char (point-max)))))
22
23 (defun +messages-auto-tail-toggle ()
24   "Auto tail the '*Messages*' buffer."
25   (interactive)
26   (if +messages--auto-tail-enabled
27       (progn
28         (advice-remove 'message '+messages--auto-tail-a)
29         (setq +messages--auto-tail-enabled nil)
30         (message "+messages-auto-tail: Disabled."))
31       (advice-add 'message :after '+messages--auto-tail-a)
32       (setq +messages--auto-tail-enabled t)
33       (message "+messages-auto-tail: Enabled.")))

```

#### 4.4.4 Undo and auto-save

```

1 (package! super-save
2   :disable t)

```

##### Auto-save

```

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

```

```

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

```

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

```

1 ;; Increase undo history limits even more
2 (after! undo-fu
3   (setq undo-limit      100000000 ;; 1MB (default is 160kB, Doom's default is 400kB)
4         undo-strong-limit 1000000000 ;; 100MB (default is 240kB, Doom's default is 3MB)
5         undo-outer-limit 1000000000) ;; 1GB (default is 24MB, Doom's default is 48MB)
6
7   (after! evil
8     (setq evil-want-fine-undo t) ;; By default while in insert all changes are one big blob

```

```

1 (package! vundo
2   :recipe (:host github
3           :repo "casouri/vundo"))

```



### Visual undo (vundo)

```

1 (use-package! vundo
2   :defer t
3   :init
4   (defconst +vundo-unicode-symbols
5     '((selected-node . ?)
6       (node . ?)
7       (vertical-stem . ?)
8       (branch . ?)
9       (last-branch . ?)
10      (horizontal-stem . ?)))
11
12   (map! :leader
13         (:prefix ("o")
14          :desc "vundo" "v" #'vundo))
15
16   :config
17   (setq vundo-glyph-alist +vundo-unicode-symbols
18         vundo-compact-display t
19         vundo-window-max-height 6))

```

### 4.4.5 Editing

```

1 ;; Stretch cursor to the glyph width
2 (setq-default x-stretch-cursor t)
3
4 ;; Enable relative line numbers
5 (setq display-line-numbers-type 'relative)
6
7 ;; Iterate through CamelCase words
8 (global-subword-mode 1)

```

### 4.4.6 Emacs sources

```

1 (setq source-directory
2       (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.

```

1 ;; NOTE: Not tangled, not working
2 (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

```

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

```

## 5 Emacs daemon

### 5.1 Initialization

```

1 (defun +daemon-startup ()
2   ;; mu4e
3   (when (require 'mu4e nil t)
4     ;; Automatically start `mu4e' in background.
5     (when (load! "mu-lock.el" (expand-file-name "email/mu4e/autoload" doom-modules-dir) t)
6       (setq +mu4e-lock-greedy t
7             +mu4e-lock-relaxed t)
8       (when (+mu4e-lock-available t)
9         ;; Check each 5m, if `mu4e' if closed, start it in background.
10        (run-at-time nil ;; Launch now
11                     (* 60 5) ;; Check each 5 minutes
12                     (lambda ()
13                       (when (and (not (mu4e-running-p)) (+mu4e-lock-available))
14                         (mu4e--start)
15                         (message "Started `mu4e' in background.")))))))
16
17   ;; RSS
18   (when (require 'elfeed nil t)
19     (run-at-time nil (* 2 60 60) #'elfeed-update))) ;; Check every 2h
20
21 (when (daemonp)
22   ;; At daemon startup
23   (add-hook 'emacs-startup-hook #' +daemon-startup)
24
25   ;; After creating a new frame (via emacsclient)
26   ;; Reload Doom's theme
27   (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.

```

1 (when (daemonp)
2   (add-hook '(delete-frame-functions delete-terminal-functions)
3     (let ((inhibit-message t))
4       (recentf-save-list)
5       (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 xlfed 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

```

1 (setq doom-font (font-spec :family "Iosevka Fixed Curly Slab" :size 20)
2   doom-big-font (font-spec :family "Iosevka Fixed Curly Slab" :size 30 :weight 'light)
3   doom-variable-pitch-font (font-spec :family "Iosevka Fixed Curly Slab")
4   doom-unicode-font (font-spec :family "JuliaMono")
5   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

```

1 (setq doom-theme 'doom-one-light)
2 (remove-hook 'window-setup-hook #'doom-init-theme-h)
3 (add-hook 'after-init-hook #'doom-init-theme-h 'append)

```

```
(package! modus-themes)
```

## 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)))
```

```

64 ;; Tweak `display-fill-column-indicator-mode'
65 `(fill-column-indicator ((,class :height 0.3 :background ,bg-inactive :foreground ,bg-inactive)))
66 ;; Tweak `mmm-mode'
67 `(mmm-cleanup-submode-face ((,class :background ,yellow-refine-bg)))
68 `(mmm-code-submode-face ((,class :background ,bg-active)))
69 `(mmm-comment-submode-face ((,class :background ,blue-refine-bg)))
70 `(mmm-declaration-submode-face ((,class :background ,cyan-refine-bg)))
71 `(mmm-default-submode-face ((,class :background ,bg-alt)))
72 `(mmm-init-submode-face ((,class :background ,magenta-refine-bg)))
73 `(mmm-output-submode-face ((,class :background ,red-refine-bg)))
74 `(mmm-special-submode-face ((,class :background ,green-refine-bg))))))
75
76 (add-hook 'modus-themes-after-load-theme-hook #'modus-themes-tweak-packages)
77
78 :config
79 (modus-themes-load-operandi)
80 (map! :leader
81      :prefix "t" ;; toggle
82      :desc "Toggle Modus theme" "m" #'modus-themes-toggle))

```

### 6.1.3 Modeline

**Clock** Display time and set the format to 24h.

```

1 (after! doom-modeline
2   (setq display-time-string-forms
3     '((propertyize (concat " " 24-hours ":" minutes))))
4   (display-time-mode 1) ; Enable time in the mode-line
5
6   ;; Add padding to the right
7   (doom-modeline-def-modeline 'main
8     '(bar matches buffer-info remote-host buffer-position parrot selection-info)
9     '(misc-info minor-modes checker input-method buffer-encoding major-mode process vcs " "))) ; <-- added
↪ padding here

```

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

```

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

```

```

1 (after! doom-modeline
2   (setq doom-modeline-bar-width 4
3         doom-modeline-mu4e t
4         doom-modeline-major-mode-icon t
5         doom-modeline-major-mode-color-icon t
6         doom-modeline-buffer-file-name-style 'truncate-upto-project))

```

## Mode line customization

### 6.1.4 Set transparency

```

1 ;; NOTE: Not tangled
2 (set-frame-parameter (selected-frame) 'alpha '(85 100))
3 (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
<code>emacs-e.svg</code>
<code>gnu-emacs-white.svg</code>
<code>gnu-emacs-flat.svg</code>
<code>blackhole-lines.svg</code>
<code>doom-emacs-white.svg</code>
<code>doom-emacs-dark.svg</code>
<code>doom-emacs-gray.svg</code>

```

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

```

```

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

```

## Dashboard

### 6.1.6 Which key

Make `which-key` popup faster.

```

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

```

I'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.

```

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

```

### 6.1.7 Window title

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

```

1 (setq frame-title-format
2   '(" "
3     (:eval
4       (if (s-contains-p org-roam-directory (or buffer-file-name ""))
5         (replace-regexp-in-string ".*[0-9]*-?" " "

```

```

6                                     (subst-char-in-string ?_ ? buffer-file-name))
7         "%b"))
8     (:eval
9       (let* ((project-name (projectile-project-name))
10              (project-name (if (string= "-" project-name)
11                                (ignore-errors (file-name-base (string-trim-right (vc-root-dir))))
12                                project-name)))
13         (when project-name
14           (format (if (buffer-modified-p) " %s" " %s") project-name))))))

```

### 6.1.8 SVG tag and svg-lib

```

1 (package! svg-tag-mode)

```

```

1 (use-package! svg-tag-mode
2   :commands svg-tag-mode
3   :config
4   (setq svg-tag-tags
5     '(("~\*. * \\. \([A-Za-z0-9]+\)" .
6       ((lambda (tag)
7         (svg-tag-make
8           tag
9           :beg 1
10          :font-family "Roboto Mono"
11          :font-size 10
12          :height 0.8
13          :padding 0
14          :margin 0))))
15      ("\\([A-Za-z0-9]+:\\)" .
16        ((lambda (tag)
17          (svg-tag-make
18            tag
19            :beg 1
20            :end -1
21            :font-family "Roboto Mono"
22            :font-size 10
23            :height 0.8
24            :padding 0
25            :margin 0)))))))

```

```

1 (after! svg-lib
2   ;; Set `svg-lib' cache directory
3   (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.

```

1 (package! focus)

```

```

1 (use-package! focus
2   :commands focus-mode)

```

### 6.1.10 Scrolling

```

1 (package! good-scroll
2   :disable EMACS29+)

1 (use-package! good-scroll
2   :unless EMACS29+
3   :config (good-scroll-mode 1))
4
5 (when EMACS29+
6   (pixel-scroll-precision-mode 1))
7
8 (setq hscroll-step 1
9       hscroll-margin 0
10      scroll-step 1
11      scroll-margin 0
12      scroll-conservatively 101
13      scroll-up-aggressively 0.01
14      scroll-down-aggressively 0.01
15      scroll-preserve-screen-position 'always
16      auto-window-vscroll nil
17      fast-but-imprecise-scrolling nil)

```

### 6.1.11 All the icons

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

```

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

```

### 6.1.12 Tabs

```

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

```

## 6.2 Highlight indent guides

```

1 (after! highlight-indent-guides
2   (setq highlight-indent-guides-character ?|
3         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.



```

1 (set-file-template! "\\\\.tex$" :trigger "__" :mode 'latex-mode)
2 (set-file-template! "\\\\.org$" :trigger "__" :mode 'org-mode)
3 (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`.

```

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

```

### 6.3.3 Mouse buttons

Map extra mouse buttons to jump between buffers

```

1 (map! :n [mouse-8] #'better-jumper-jump-backward
2       :n [mouse-9] #'better-jumper-jump-forward)
3
4 ;; Enable horizontal scrolling with the second mouse wheel or the touchpad
5 (setq mouse-wheel-tilt-scroll t
6       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.

```

1 (package! vlf)

```

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

```

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

```

### 6.3.5 Evil

```

1 (after! evil
2   ;; This fixes https://github.com/doomemacs/doomemacs/issues/6478
3   ;; Ref: https://github.com/emacs-evil/evil/issues/1630
4   (evil-select-search-module 'evil-search-module 'isearch)
5
6   (setq evil-kill-on-visual-paste nil)) ; Don't put overwritten text in the kill ring

```

```

1 (package! evil-escape :disable t)

```

### 6.3.6 Aggressive indent

```

1 (package! aggressive-indent)

```

```

1 (use-package! aggressive-indent
2   :commands (aggressive-indent-mode))

```

### 6.3.7 YASnippet

Nested snippets are good, enable that.

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

## 6.4 Completion & IDE

### 6.4.1 Company

I do not find company useful in Org files.

```
1 (setq company-global-modes
2   '(not erc-mode
3     circe-mode
4     message-mode
5     help-mode
6     gud-mode
7     vterm-mode
8     org-mode))
```

```
1 (after! company-box
2   (when (daemonp)
3     (defun +company-box--reload-icons-h ()
4       (setq company-box-icons-all-the-icons
5         (let ((all-the-icons-scale-factor 0.8))
6           `( (Unknown      . , (all-the-icons-faicon "code"          :face 'all-the-icons-purple))
7             (Text         . , (all-the-icons-material "text_fields"  :face 'all-the-icons-green))
8             (Method       . , (all-the-icons-faicon "cube"          :face 'all-the-icons-red))
9             (Function     . , (all-the-icons-faicon "cube"          :face 'all-the-icons-red))
10            (Constructor  . , (all-the-icons-faicon "cube"          :face 'all-the-icons-red))
11            (Field        . , (all-the-icons-faicon "tag"           :face 'all-the-icons-red))
12            (Variable     . , (all-the-icons-material "adjust"       :face 'all-the-icons-blue))
13            (Class        . , (all-the-icons-material "class"       :face 'all-the-icons-red))
14            (Interface    . , (all-the-icons-material "tune"        :face 'all-the-icons-red))
15            (Module       . , (all-the-icons-faicon "cubes"         :face 'all-the-icons-red))
16            (Property     . , (all-the-icons-faicon "wrench"        :face 'all-the-icons-red))
17            (Unit         . , (all-the-icons-material "straighten"   :face 'all-the-icons-red))
18            (Value        . , (all-the-icons-material "filter_1"    :face 'all-the-icons-red))
19            (Enum         . , (all-the-icons-material "plus_one"    :face 'all-the-icons-red))
20            (Keyword      . , (all-the-icons-material "filter_center_focus" :face 'all-the-icons-red-alt))
21            (Snippet      . , (all-the-icons-faicon "expand"        :face 'all-the-icons-red))
22            (Color        . , (all-the-icons-material "colorize"    :face 'all-the-icons-red))
23            (File         . , (all-the-icons-material "insert_drive_file" :face 'all-the-icons-red))
24            (Reference    . , (all-the-icons-material "collections_bookmark" :face 'all-the-icons-red))
25            (Folder       . , (all-the-icons-material "folder"      :face 'all-the-icons-red-alt))
26            (EnumMember   . , (all-the-icons-material "people"      :face 'all-the-icons-red))
27            (Constant     . , (all-the-icons-material "pause_circle_filled" :face 'all-the-icons-red))
28            (Struct       . , (all-the-icons-material "list"        :face 'all-the-icons-red))
29            (Event        . , (all-the-icons-material "event"       :face 'all-the-icons-red))
30            (Operator     . , (all-the-icons-material "control_point" :face 'all-the-icons-red))
31            (TypeParameter . , (all-the-icons-material "class"      :face 'all-the-icons-red))
32            (Template     . , (all-the-icons-material "settings_ethernet" :face 'all-the-icons-green))
33            (ElispFunction . , (all-the-icons-faicon "cube"          :face 'all-the-icons-red))
34            (ElispVariable . , (all-the-icons-material "adjust"     :face 'all-the-icons-blue))
35            (ElispFeature . , (all-the-icons-material "stars"       :face 'all-the-icons-orange))
36            (ElispFace    . , (all-the-icons-material "format_paint" :face 'all-the-icons-pink))))))
37
38 ;; Replace Doom defined icons with mine
39 (when (memq #' +company-box--load-all-the-icons server-after-make-frame-hook)
40   (remove-hook 'server-after-make-frame-hook #' +company-box--load-all-the-icons))
41 (add-hook 'server-after-make-frame-hook #' +company-box--reload-icons-h))
```

## Tweak company-box

## 6.4.2 Treemacs

```

1 (unpin! treemacs)
2 (unpin! lsp-treemacs)

1 (after! treemacs
2   (require 'dired)
3
4   ;; My custom stuff (from tecosaur's config)
5   (setq +treemacs-file-ignore-extensions
6     ';; LaTeX
7       "aux" "ptc" "fdb_latexmk" "fls" "synctex.gz" "toc"
8       ;; LaTeX - bibliography
9       "bbl"
10      ;; LaTeX - glossary
11      "glg" "glo" "gls" "glsdefs" "ist" "acn" "acr" "alg"
12      ;; LaTeX - pgfplots
13      "mw"
14      ;; LaTeX - pdfx
15      "pdfa.xmpi"
16      ;; Python
17      "pyc"))
18
19   (setq +treemacs-file-ignore-globs
20     ';; LaTeX
21       "*/_minted-*"
22       ;; AucTeX
23       "*/.auctex-auto"
24       "*/_region_.log"
25       "*/_region_.tex"
26       ;; Python
27       "*/__pycache__")
28
29   ;; Reload treemacs theme
30   (setq doom-themes-treemacs-enable-variable-pitch nil
31         doom-themes-treemacs-theme "doom-colors")
32   (doom-themes-treemacs-config)
33
34   (setq treemacs-show-hidden-files nil
35         treemacs-hide-dot-git-directory t
36         treemacs-width 30)
37
38   (defvar +treemacs-file-ignore-extensions '()
39     "File extension which `treemacs-ignore-filter' will ensure are ignored")
40
41   (defvar +treemacs-file-ignore-globs '()
42     "Globs which will be transformed to `+treemacs-file-ignore-regexps' which `+treemacs-ignore-filter' will
43     ↪ ensure are ignored")
44
45   (defvar +treemacs-file-ignore-regexps '()
46     "RegExps to be tested to ignore files, generated from `+treemacs-file-ignore-globs'")
47
48   (defun +treemacs-file-ignore-generate-regexps ()
49     "Generate `+treemacs-file-ignore-regexps' from `+treemacs-file-ignore-globs'"
50     (setq +treemacs-file-ignore-regexps (mapcar 'dired-glob-regexp +treemacs-file-ignore-globs)))
51
52   (unless (equal +treemacs-file-ignore-globs '())
53     (+treemacs-file-ignore-generate-regexps))
54
55   (defun +treemacs-ignore-filter (file full-path)
56     "Ignore files specified by `+treemacs-file-ignore-extensions', and `+treemacs-file-ignore-regexps'"
57     (or (member (file-name-extension file) +treemacs-file-ignore-extensions)
58         (let ((ignore-file nil))
59           (dolist (regexp +treemacs-file-ignore-regexps ignore-file)
80             (setq ignore-file (or ignore-file (if (string-match-p regexp full-path) t nil)))))))

```

```

60
61 (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.

```

1  ;; Run `M-x projectile-discover-projects-in-search-path' to reload paths from this variable
2  (setq projectile-project-search-path
3    '("~/PhD/papers"
4      "~/PhD/workspace"
5      "~/PhD/workspace-no"
6      "~/PhD/workspace-no/ez-wheel/swd-starter-kit-repo"
7      ("~/Projects/foss" . 2))) ;; ("dir" . depth)
8
9  (setq projectile-ignored-projects
10    '("/tmp"
11      "~/."
12      "~/.cache"
13      "~/.doom.d"
14      "~/.emacs.d/.local/straight/repos/"))
15
16  (setq +projectile-ignored-roots
17    '("~/."
18      ;; No need for this one, as `doom-project-ignored-p' checks for files in `doom-local-dir'
19      "~/.emacs.d/.local/straight/"))
20
21  (defun +projectile-ignored-project-function (filepath)
22    "Return t if FILEPATH is within any of `+projectile-ignored-roots'"
23    (require 'cl-lib)
24    (or (doom-project-ignored-p filepath) ;; Used by default by doom with `projectile-ignored-project-function'
25        (cl-some (lambda (root) (file-in-directory-p (expand-file-name filepath) (expand-file-name root)))
26                  +projectile-ignored-roots)))
27
28  (setq projectile-ignored-project-function #' +projectile-ignored-project-function)

```

### 6.4.4 Tramp

Let's try to make tramp handle prompts better

```

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

```

### 6.4.5 Eros-eval

This makes the result of evals slightly prettier.

```

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

```

### 6.4.6 dir-locals.el

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

```

1 (defun +dir-locals-reload-for-current-buffer ()
2   "reload dir locals for the current buffer"
3   (interactive)
4   (let ((enable-local-variables :all))
5     (hack-dir-local-variables-non-file-buffer)))
6
7 (defun +dir-locals-reload-for-all-buffers-in-this-directory ()
8   "For every buffer with the same `default-directory` as the
9   current buffer's, reload dir-locals."
10  (interactive)
11  (let ((dir default-directory))
12    (dolist (buffer (buffer-list))
13      (with-current-buffer buffer
14        (when (equal default-directory dir)
15          (+dir-locals-reload-for-current-buffer))))))
16
17 (defun +dir-locals-enable-autoreload ()
18   (when (and (buffer-file-name)
19             (equal dir-locals-file (file-name-nondirectory (buffer-file-name))))
20     (message "Dir-locals will be reloaded after saving.")
21     (add-hook 'after-save-hook '+dir-locals-reload-for-all-buffers-in-this-directory nil t)))
22
23 (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`:

```

1 (after! eglot
2   ;; A hack to make it works with projectile
3   (defun projectile-project-find-function (dir)
4     (let* ((root (projectile-project-root dir))
5            (and root (cons 'transient root))))
6
7     (with-eval-after-load 'project
8       (add-to-list 'project-find-functions 'projectile-project-find-function))
9
10  ;; Use clangd with some options
11  (set-eglot-client! 'c++-mode '("clangd" "-j=3" "--clang-tidy")))

```

### LSP mode

```

1 (unpin! lsp-mode)

```

#### Unpin package

**Tweak 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.

```

1 (after! lsp-ui
2   (setq lsp-ui-sideline-enable t
3         lsp-ui-sideline-show-code-actions t
4         lsp-ui-sideline-show-diagnostics t
5         lsp-ui-sideline-show-hover nil
6         lsp-log-io nil
7         lsp-lens-enable t ;; not working properly with ccls!
8         lsp-diagnostics-provider :auto)

```

```

9      lsp-semantic-tokens-enable t ;; hide unreachable ifdefs
10     lsp-enable-symbol-highlighting t
11     lsp-headerline-breadcrumb-enable nil
12     lsp-headerline-breadcrumb-segments '(symbols)))

```

```

1 (after! lsp-clangd
2   (setq lsp-clients-clangd-args
3     '("--j=4"
4       "--background-index"
5       "--clang-tidy"
6       "--completion-style=detailed"
7       "--header-insertion=never"
8       "--header-insertion-decorators=0"))
9   (set-lsp-priority! 'clangd 1))

```

### LSP mode with clangd

```

1 ;; NOTE: Not tangled, using the default ccls
2 (after! ccls
3   (setq ccls-initialization-options
4     '(:index (:comments 2
5              :trackDependency 1
6              :threads 4)
7       :completion (:detailedLabel t)))
8   (set-lsp-priority! 'ccls 2)) ; optional as ccls is the default in Doom

```

### LSP mode with ccls

#### Enable lsp over tramp

##### 1. Python

```

1 (after! tramp
2   (require 'lsp-mode)
3   ;; (require 'lsp-pyright)
4
5   (setq lsp-enable-snippet nil
6         lsp-log-io nil
7         ;; To bypass the "lsp--document-highlight fails if
8         ;; textDocument/documentHighlight is not supported" error
9         lsp-enable-symbol-highlighting nil)
10
11   (lsp-register-client
12     (make-lsp-client
13       :new-connection (lsp-tramp-connection "pyls")
14       :major-modes '(python-mode)
15       :remote? t
16       :server-id 'pyls-remote)))

```

##### 2. C/C++ with ccls

```

1 ;; NOTE: WIP: Not tangled
2 (after! tramp
3   (require 'lsp-mode)
4   (require 'ccls)

```

```

5
6 (setq lsp-enable-snippet nil
7   lsp-log-io nil
8   lsp-enable-symbol-highlighting t)
9
10 (lsp-register-client
11  (make-lsp-client
12   :new-connection
13   (lsp-tramp-connection
14    (lambda ()
15      (cons ccls-executable ; executable name on remote machine 'ccls'
16            ccls-args)))
17   :major-modes '(c-mode c++-mode objc-mode cuda-mode)
18   :remote? t
19   :server-id 'ccls-remote))
20
21 (add-to-list 'tramp-remote-path 'tramp-own-remote-path))

```

### 3. C/C++ with clangd

```

1 (after! tramp
2   (require 'lsp-mode)
3
4   (setq lsp-enable-snippet nil
5         lsp-log-io nil
6         ;; To bypass the "lsp--document-highlight fails if
7         ;; textDocument/documentHighlight is not supported" error
8         lsp-enable-symbol-highlighting nil)
9
10  (lsp-register-client
11   (make-lsp-client
12    :new-connection
13    (lsp-tramp-connection
14     (lambda ()
15       (cons "clangd-12" ; executable name on remote machine 'ccls'
16             lsp-clients-clangd-args)))
17    :major-modes '(c-mode c++-mode objc-mode cuda-mode)
18    :remote? t
19    :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 from pip through the `pyghdl` package, so let's use `rust_hdl` instead.

```

1 (use-package! vhd1-mode
2   :hook (vhd1-mode . #'lsp-vhd1-ls-load)
3   :init
4   (defun +lsp-vhd1-ls-load ()
5     (interactive)
6     (lsp t)
7     (flycheck-mode t))
8
9   :config
10  ;; Required unless vhd1_ls is on the $PATH
11  (setq lsp-vhd1-server-path "~/Projects/foss/repos/rust_hdl/target/release/vhd1_ls"
12        lsp-vhd1-server 'vhd1-ls
13        lsp-vhd1--params nil)
14  (require 'lsp-vhd1))

```

```

1 (package! lsp-sonarlint
2   :disable t)

```

### SonarLint

```

1 (use-package! lsp-sonarlint)

```

#### 6.4.8 Cppcheck

Check for everything!

```

1 (after! flycheck
2   (setq flycheck-cppcheck-checks '("information"
3     "missingInclude"
4     "performance"
5     "portability"
6     "style"
7     "unusedFunction"
8     "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`.

```

1 (package! project-cmake
2   :disable (not (modulep! :tools lsp +eglot)) ; Enable only if (lsp +eglot) is used
3   :recipe (:host github
4     :repo "juanjosegarciaripoll/project-cmake"))

```

```

1 (use-package! project-cmake
2   :config
3   (require 'eglot)
4   (project-cmake-scan-kits)
5   (project-cmake-eglot-integration))

```

#### 6.4.10 Clang-format

```

1 (package! clang-format)

```

```

1 (use-package! clang-format
2   :when CLANG-FORMAT-P
3   :commands (clang-format-region))

```

#### 6.4.11 Auto-include C++ headers

```

1 (package! cpp-auto-include
2   :recipe (:host github
3     :repo "emacsorphanage/cpp-auto-include"))

```



```

1 (use-package! cpp-auto-include
2   :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).

```

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

```

#### 6.4.13 Erefactor

```

1 (package! erefactor
2   :recipe (:host github
3           :repo "mhayashi1120/Emacs-erefactor"))

```

```

1 (use-package! erefactor
2   :defer t)

```

#### 6.4.14 Lorem ipsum

```

1 (package! emacs-lorem-ipsum
2   :recipe (:host github
3           :repo "jschaf/emacs-lorem-ipsum"))

```

```

1 (use-package! lorem-ipsum
2   :commands (lorem-ipsum-insert-sentences
3             lorem-ipsum-insert-paragraphs
4             lorem-ipsum-insert-list))

```

#### 6.4.15 Coverage test

```

1 (package! cov)

```

## 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.

```

1 (unpin! dap-mode)

1 (after! dap-mode
2   ;; Set latest versions
3   (setq dap-cpptools-extension-version "1.11.5")
4   (require 'dap-cpptools)
5
6   (setq dap-codelldb-extension-version "1.7.4")
7   (require 'dap-codelldb)
8
9   (setq dap-gdb-lldb-extension-version "0.26.0")
10  (require 'dap-gdb-lldb)
11
12  ;; More minimal UI
13  (setq dap-auto-configure-features '(breakpoints locals expressions tooltip)
14        dap-auto-show-output nil ;; Hide the annoying server output
15        lsp-enable-dap-auto-configure t)
16
17  ;; Automatically trigger dap-hydra when a program hits a breakpoint.
18  (add-hook 'dap-stopped-hook (lambda (arg) (call-interactively #'dap-hydra)))
19
20  ;; Automatically delete session and close dap-hydra when DAP is terminated.
21  (add-hook 'dap-terminated-hook
22            (lambda (arg)
23              (call-interactively #'dap-delete-session)
24              (dap-hydra/nil)))
25
26  ;; A workaround to correctly show breakpoints
27  ;; from: https://github.com/emacs-lsp/dap-mode/issues/374#issuecomment-1140399819
28  (add-hook! +dap-running-session-mode
29            (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.)

```

1 (defun +debugger/clear-last-session ()
2   "Clear the last stored session"
3   (interactive)
4   (doom-store-clear "+debugger"))
5
6 (map! :leader :prefix ("l" . "custom")
7       (:when (modulep! :tools debugger +lsp)
8             :prefix ("d" . "debugger")
9             :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).

```

1 (after! realgud
2   (require 'hydra)
3
4   ;; Add some missing gdb/rr commands
5   (defun +realgud:cmd-start (arg)

```

```

6   "start = break main + run"
7   (interactive "p")
8   (realgud-command "start"))
9
10  (defun +realgud:cmd-reverse-next (arg)
11    "Reverse next"
12    (interactive "p")
13    (realgud-command "reverse-next"))
14
15  (defun +realgud:cmd-reverse-step (arg)
16    "Reverse step"
17    (interactive "p")
18    (realgud-command "reverse-step"))
19
20  (defun +realgud:cmd-reverse-continue (arg)
21    "Reverse continue"
22    (interactive "p")
23    (realgud-command "reverse-continue"))
24
25  (defun +realgud:cmd-reverse-finish (arg)
26    "Reverse finish"
27    (interactive "p")
28    (realgud-command "reverse-finish"))
29
30  ;; Define a hydra binding
31  (defhydra realgud-hydra (:color pink :hint nil :foreign-keys run)
32    "
33    Stepping | _n_: next      | _i_: step   | _o_: finish | _c_: continue | _R_: restart | _u_:
34    ↪ until-here
35    Reverse  | _rn_: next    | _ri_: step  | _ro_: finish | _rc_: continue |
36    Breakpts | _ba_: break   | _bD_: delete | _bt_: tbreak | _bd_: disable  | _be_: enable  | _tr_:
37    ↪ backtrace
38    Eval      | _ee_: at-point | _er_: region | _eE_: eval   |
39    | _!_: shell   | _Qk_: kill   | _Qq_: quit   | _Sg_: gdb     | _Ss_: start
40    "
41    ("n" realgud:cmd-next)
42    ("i" realgud:cmd-step)
43    ("o" realgud:cmd-finish)
44    ("c" realgud:cmd-continue)
45    ("R" realgud:cmd-restart)
46    ("u" realgud:cmd-until-here)
47    ("rn" +realgud:cmd-reverse-next)
48    ("ri" +realgud:cmd-reverse-step)
49    ("ro" +realgud:cmd-reverse-finish)
50    ("rc" +realgud:cmd-reverse-continue)
51    ("ba" realgud:cmd-break)
52    ("bt" realgud:cmd-tbreak)
53    ("bD" realgud:cmd-delete)
54    ("be" realgud:cmd-enable)
55    ("bd" realgud:cmd-disable)
56    ("ee" realgud:cmd-eval-at-point)
57    ("er" realgud:cmd-eval-region)
58    ("tr" realgud:cmd-backtrace)
59    ("eE" realgud:cmd-eval)
60    ("!" realgud:cmd-shell)
61    ("Qk" realgud:cmd-kill)
62    ("Sg" realgud:gdb)
63    ("Ss" +realgud:cmd-start)
64    ("q" nil "quit" :color blue) ;; :exit
65    ("Qq" realgud:cmd-quit :color blue)) ;; :exit
66
67  (defun +debugger/realgud:gdb-hydra ()
68    "Run `realgud-hydra'."
69    (interactive)
70    (realgud-hydra/body))
71
72  (map! :leader :prefix ("l" . "custom")
73    (:when (modulep! :tools debugger)
74      :prefix ("d" . "debugger")
75      :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](#).

```

1 (after! realgud
2   (defun +debugger/rr-replay ()
3     "Launch `rr replay'."
4     (interactive)
5     (realgud:gdb (+str-replace "gdb" "rr replay" realgud:gdb-command-name)))
6
7   (defun +debugger/rr-record ()
8     "Launch `rr record' with parameters from launch.json or `+launch-json-debug-config'."
9     (interactive)
10    (let* ((conf (launch-json--config-choice))
11           (args (launch-json--substitute-special-vars (plist-get conf :program) (plist-get conf :args))))
12      (unless (make-process :name "rr-record"
13                           :buffer "*rr record*"
14                           :command (append '("rr" "record") args))
15        (message "Cannot start the 'rr record' process"))))
16
17  (map! :leader :prefix ("l" . "custom")
18        (:when (modulep! :tools debugger)
19              :prefix ("d" . "debugger")
20              :desc "rr record" "r" #' +debugger/rr-record
21              :desc "rr replay" "R" #' +debugger/rr-replay)))

```

```

1 (package! realgud-lldb)
2 (package! realgud-ipdb)
3 (package! realgud-trepan-xpy :recipe (:host github :repo "realgud/trepan-xpy"))
4 (package! realgud-maxima :recipe (:host github :repo "realgud/realgud-maxima"))

```

## 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 its C module), and it defines some easy to use UI, with Visual Studio like keybindings.

```

1 (package! gdb-mi
2   :disable t
3   :recipe (:host github
4             :repo "weirdNox/emacs-gdb"
5             :files ("*.el" "*.c" "*.h" "Makefile")))

```

```

1 (use-package! gdb-mi
2   :init
3   (fmakeunbound 'gdb)
4   (fmakeunbound 'gdb-enable-debug)
5
6   :config
7   (setq gdb-window-setup-function #'gdb--setup-windows ;; TODO: Customize this
8         gdb-ignore-gdbinit nil) ;; I use gdbinit to define some useful stuff
9   ;; History

```

```

10 (defvar +gdb-history-file "~/gdb_history")
11 (defun +gud-gdb-mode-hook-setup ()
12   "GDB setup."
13
14   ;; Suposes "~/gdbinit" contains:
15   ;; set history save on
16   ;; set history filename ~/gdb_history
17   ;; set history remove-duplicates 2048
18   (when (and (ring-empty-p comint-input-ring)
19             (file-exists-p +gdb-history-file))
20     (setq comint-input-ring-file-name +gdb-history-file)
21     (comint-read-input-ring t)))
22
23 (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.

```

1 (setq gdb-many-windows nil)
2
3 (defun set-gdb-layout(&optional c-buffer)
4   (if (not c-buffer)
5       (setq c-buffer (window-buffer (selected-window)))) ;; save current buffer
6
7   ;; from http://stackoverflow.com/q/39762833/846686
8   (set-window-dedicated-p (selected-window) nil) ;; unset dedicate state if needed
9   (switch-to-buffer gud-comint-buffer)
10  (delete-other-windows) ;; clean all
11
12  (let* ((w-source (selected-window)) ;; left top
13         (w-gdb (split-window w-source nil 'right)) ;; right bottom
14         (w-locals (split-window w-gdb nil 'above)) ;; right middle bottom
15         (w-stack (split-window w-locals nil 'above)) ;; right middle top
16         (w-breakpoints (split-window w-stack nil 'above)) ;; right top
17         (w-io (split-window w-source (floor(* 0.9 (window-body-height))) 'below))) ;; left bottom
18    (set-window-buffer w-io (gdb-get-buffer-create 'gdb-inferior-io))
19    (set-window-dedicated-p w-io t)
20    (set-window-buffer w-breakpoints (gdb-get-buffer-create 'gdb-breakpoints-buffer))
21    (set-window-dedicated-p w-breakpoints t)
22    (set-window-buffer w-locals (gdb-get-buffer-create 'gdb-locals-buffer))
23    (set-window-dedicated-p w-locals t)
24    (set-window-buffer w-stack (gdb-get-buffer-create 'gdb-stack-buffer))
25    (set-window-dedicated-p w-stack t)
26
27    (set-window-buffer w-gdb gud-comint-buffer)
28
29    (select-window w-source)
30    (set-window-buffer w-source c-buffer)))
31
32 (defadvice gdb (around args activate)
33   "Change the way to gdb works."
34   (setq global-config-editing (current-window-configuration)) ;; to restore: (set-window-configuration c-editin_
35   g)
36   (let ((c-buffer (window-buffer (selected-window)))) ;; save current buffer
37     ad-do-it
38     (set-gdb-layout c-buffer)))
39
40 (defadvice gdb-reset (around args activate)
41   "Change the way to gdb exit."
42   ad-do-it
43   (set-window-configuration global-config-editing))

```

```

1 (defvar gud-overlay
2   (let* ((ov (make-overlay (point-min) (point-min))))
3     (overlay-put ov 'face 'secondary-selection)
4     ov)
5   "Overlay variable for GUD highlighting.")
6
7 (defadvice gud-display-line (after my-gud-highlight act)
8   "Highlight current line."
9   (let* ((ov gud-overlay)
10          (bf (gud-find-file true-file)))
11     (with-current-buffer bf
12       (move-overlay ov (line-beginning-position) (line-beginning-position 2)
13                     ;; (move-overlay ov (line-beginning-position) (line-end-position)
14                     (current-buffer)))))
15
16 (defun gud-kill-buffer ()
17   (if (derived-mode-p 'gud-mode)
18       (delete-overlay gud-overlay)))
19
20 (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.

```

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

```

This variable should have the same structure of a `launch.json` file, in Emacs, 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:

```

1 ;; Example entry in .dir-locals.el
2 ((nil . ((+launch-json-debug-config . '(:name "Debug my_prog with csv data"
3                                          :type "realgud:gdb" ;; One config with `realgud:gdb`
4                                          :program "${workspaceFolder}/build/bin/my_prog"
5                                          :args ("--in_file=${workspaceFolder}/some/file.csv"
6                                                  "--some-parameter" "-a" "-b"
7                                                  "--out_file=/tmp/some_random_file"
8                                                  "-a")))
9
10      (:name "Debug my_prog with options '-a' and '-b'"
11          :type "gdb-mi" ;; Another config with `gdb-mi`
12          :program "${workspaceFolder}/build/bin/my_prog"
13          :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.

```

1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "name": "Emacs::RealGUD:GDB (view_trajectory)",
6       "type": "realgud:gdb",
7       "request": "launch",
8       "dap-compilation": "cmake --build build/debug -- -j 8",
9       "dap-compilation-dir": "${workspaceFolder}",
10      "program": "${workspaceFolder}/build/debug/bin/view_trajectory",
11      "args": [
12        "htraj=${workspaceFolder}/data/seq1/h_poses.csv",
13        "traj=${workspaceFolder}/data/seq1/poses.csv"
14      ],
15      "stopAtEntry": false,
16      "cwd": "${workspaceFolder}",
17      "environment": [],
18      "externalConsole": false
19    }
20  ]
21 }
```

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

```

1 (defvar launch-json--gud-debugger-regex
2   (rx (seq bol (group-n 1 (or "gdb" "gud-gdb" "perlldb" "pdb" "jdb" "guiler" "dbx" "sdb" "xdb") eol))))
3
4 (defvar launch-json--realgud-debugger-regex
5   (rx (seq bol (or (seq "realgud:" (group-n 1 (or "gdb" "pdb"
6     "bashdb" "kshdb" "zshd"
7     "perlldb" "rdebug" "remake"
8     "trepan" "trepan2" "trepan3k" "trepanjs" "trepan.pl"))
9     (seq "realgud-" (group-n 1 (or "gub"))))
10    ;; Additional debuggers
11    (seq "realgud:" (group-n 1 (or "xdebug" "pry" "jdb" "ipdb" "trepan-xy" "trepan-ni"
12    ↪ "node-inspect"))))
13    ;; `realgud-lldb' defines the debug command as `realgud--lldb',
14    ;; We accept both `realgud:lldb' and `realgud--lldb' in the config
15    (seq "realgud" (or ":" "--") (group-n 1 (or "lldb")))) eol))))
```

```

16 ;; Define aliases for realgud-lldb
17 (with-eval-after-load 'realgud-llldb
18   (defalias 'realgud:lldb 'realgud--lldb)
19   (defalias 'realgud:lldb-command-name 'realgud--lldb-command-name))
20
21 ;; Define aliases for realgud-ipdb
22 (with-eval-after-load 'realgud-ipdb
23   (defalias 'realgud:ipdb-command-name 'realgud--ipdb-command-name))
24
25 (defvar launch-json--last-config nil)
26
27 (defun launch-json-last-config-clear ()
28   (interactive)
29   (setq-local launch-json--last-config nil))
30
31 (defun launch-json--substitute-special-vars (program &optional args)
32   "Substitute variables in PROGRAM and ARGS.
33   Return a list, in which processed PROGRAM is the first element, followed by ARGS."
34   (let* ((curr-file (ignore-errors (expand-file-name (buffer-file-name))))
35          (ws-root (string-trim-right
36                    (expand-file-name
37                      (or (projectile-project-root)
38                          (ignore-errors (file-name-directory curr-file))
39                          ".")
40                      "/"))
41          (ws-basename (file-name-nondirectory ws-root)))
42   ;; Replace special variables
43   (mapcar
44    (lambda (str)
45      (+str-replace-all
46       (append
47        (list
48         (cons "${workspaceFolder}" ws-root)
49         (cons "${workspaceFolderBasename}" ws-basename)
50         (cons "${userHome}" (or (getenv "HOME") (expand-file-name "~")))
51         (cons "${pathSeparator}" (if (memq system-type
52                                         '(windows-nt ms-dos cygwin))
53                                         "\\\" "/""))
54         (cons "${selectedText}" (if (use-region-p)
55                                     (buffer-substring-no-properties
56                                      (region-beginning) (region-end)) "")))
56       ; To avoid problems if launched from a non-file buffer
57       (when curr-file
58         (list
59          (cons "${file}" curr-file)
60          (cons "${relativeFile}" (file-relative-name curr-file ws-root))
61          (cons "${relativeFileDirname}" (file-relative-name
62                                          (file-name-directory curr-file) ws-root))
63          (cons "${fileBasename}" (file-name-nondirectory curr-file))
64          (cons "${fileBasenameNoExtension}" (file-name-base curr-file))
65          (cons "${fileDirname}" (file-name-directory curr-file))
66          (cons "${fileExtname}" (file-name-extension curr-file))
67          (cons "${lineNumber}" (line-number-at-pos (point) t))))
68      str))
69   (cons program args))))
70
71 (defun launch-json--debugger-params (type)
72   (let* ((front/backend
73          (cond ((string-match launch-json--realgud-debugger-regex type)
74                 (cons 'realgud (intern (match-string 1 type))))
75                ((string-match launch-json--gud-debugger-regex type)
76                 (cons 'gud (intern (match-string 1 type))))
77                (t
78                 (cons 'unknown 'unknown)))))
79   (frontend (car front/backend))
80   (backend (cdr front/backend))
81   (cmd-sym (unless (eq frontend 'unknown)
82                   (intern (format (cond ((eq frontend 'gud) "gud-%s-%s"
83                                         ((eq frontend 'realgud) "%s-%s"
84                                         (t "%s-%s"))
85

```



```

86                                     type
87                                     "command-name")))))
88 (message "[launch-json:params]: Found type: %s -> { frontend: %s | backend: %s }"
89         type (symbol-name frontend) (symbol-name backend))
90 (cond ((memq backend '(gud-gdb gdb))
91        ;; Special case for '(gud . gdb), uses `gdb-mi'
92        (let ((use-gdb-mi (equal front/backend '(gud . gdb))))
93          `(:type ,type
94            :debug-cmd ,(if use-gdb-mi 'gdb (intern type))
95            :args-format " --args %s %s"
96            :cmd ,cmd-sym
97            :require ,(if use-gdb-mi 'gdb-mi frontend))))
98        ((eq backend 'lldb)
99         `(:type ,type
100          :debug-cmd ,(intern type)
101          :args-format " -- %s %s"
102          :cmd ,cmd-sym
103          :require ,(intern (if (eq frontend 'realgud)
104                               (+str-replace-all '("--" . "-") (":" . "-")) type)
105                             type))))
106        (t ;; TODO: to be expanded for each debugger
107         `(:type ,type
108          :debug-cmd ,(intern type)
109          :args-format " %s %s"
110          :cmd ,(if (equal front/backend '(realgud . ipdb)) 'realgud--ipdb-command-name cmd-sym)
111          :require ,(cond ((equal front/backend '(realgud . trepan-ni)) 'realgud-trepan-ni)
112                          (t frontend))))))
113
114 (defun launch-json--debug-command (params debuggee-args)
115   "Return the debug command for PARAMS with DEBUGGEE-ARGS."
116   (when-let* ((prog (car debuggee-args))
117              (cmd (plist-get params :cmd))
118              (pkg (plist-get params :require)))
119     (if (or (not pkg) (eq pkg 'unknown))
120         (progn (message "[launch-json:command]: Unknown debugger"
121                          nil)
122               (if (require (plist-get params :require) nil t)
123                   (let ((args (+str-join " " (cdr debuggee-args))))
124                     (when args (setq args (format (plist-get params :args-format) prog args)))
125                     (if (bound-and-true-p cmd)
126                         (concat (eval cmd) (if args args ""))
127                         (message "[launch-json:command]: Invalid command for type %s" (plist-get params :type))
128                         nil))
129                   (message "[launch-json:command]: Cannot add package %s" (symbol-name pkg))
130                   nil))))
131
132 (defun launch-json-read (&optional file)
133   "Return the configurations section from a launch.json FILE.
134   If FILE is nil, launch.json will be searched in the current project,
135   if it is set to a launch.json file, it will be used instead."
136   (let ((launch-json (expand-file-name (or file "launch.json") (or (projectile-project-root) "."))))
137     (when (file-exists-p launch-json)
138       (message "[launch-json]: Found \"launch.json\" at %s" launch-json)
139       (let* ((launch (with-temp-buffer
140                      (insert-file-contents launch-json)
141                      (json-parse-buffer :object-type 'plist :array-type 'list :null-object nil :false-object
142                                         ↪ nil))))
143         (configs (plist-get launch :configurations)))
144         (+filter (lambda (conf)
145                   (or (string-match-p launch-json--gud-debugger-regex (plist-get conf :type))
146                     (string-match-p launch-json--realgud-debugger-regex (plist-get conf :type))))
147                 configs))))
148
149 (defun launch-json--config-choice (&optional file)
150   (let* ((confs (or (launch-json-read file)
151                     +launch-json-debug-config))
152          (candidates (mapcar (lambda (conf)
153                                (cons (format "%s [%s]" (plist-get conf :name) (plist-get conf :type))
154                                      conf))
155                              confs)))
156

```

```

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

[illegible]

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.

```

1 (defun emojiify--replace-text-with-emoji (orig-fn emoji text buffer start end &optional target)
2   "Modify `emojiify--proptertize-text-for-emoji' to replace ascii/github emoticons with unicode emojis, on the
   ↪ fly."
3   (if (or (not emoticon-to-emoji) (= 1 (length text)))
4       (funcall orig-fn emoji text buffer start end target)
5       (delete-region start end)
6       (insert (ht-get emoji "unicode"))))
7
8 (define-minor-mode emoticon-to-emoji
9   "Write ascii/gh emojis, and have them converted to unicode live."
10  :global nil
11  :init-value nil
12  (if emoticon-to-emoji
13      (progn
14        (setq-local emojiify-emoji-styles '(ascii github unicode))
15        (advice-add 'emojiify--proptertize-text-for-emoji :around #'emojiify--replace-text-with-emoji)
16        (unless emojiify-mode
17          (emojiify-turn-on-emojiify-mode)))
18      (setq-local emojiify-emoji-styles (default-value 'emojiify-emoji-styles))
19      (advice-remove 'emojiify--proptertize-text-for-emoji #'emojiify--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.

```

1 (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  $\lambda$ , however, this cause miss formatting and sometimes messes up with the parenthesis, so let's disable ligatures on Lisps.

```

1 (defun +appened-to-negation-list (head tail)
2   (if (sequencep head)
3       (delete-dups
4         (if (eq (car tail) 'not)
5             (append head tail)
6             (append tail head)))
7       tail))
8
9 (when (modulep! :ui ligatures)
10  (setq +ligatures-extras-in-modes
11        (+appened-to-negation-list
12          +ligatures-extras-in-modes
13          '(not c-mode c++-mode emacs-lisp-mode python-mode scheme-mode racket-mode rust-mode)))
14
15  (setq +ligatures-in-modes
16        (+appened-to-negation-list
17          +ligatures-in-modes
18          '(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!".

```

1 (after! spell-fu
2   (defun +spell-fu-register-dictionary (lang)
3     "Add `LANG` to spell-fu multi-dict, with a personal dictionary."
4     ;; Add the dictionary
5     (spell-fu-dictionary-add (spell-fu-get-ispell-dictionary lang))
6     (let ((personal-dict-file (expand-file-name (format "aspell.%s.pws" lang) doom-user-dir)))
7       ;; Create an empty personal dictionary if it doesn't exists
8       (unless (file-exists-p personal-dict-file) (write-region "" nil personal-dict-file))
9       ;; Add the personal dictionary
10      (spell-fu-dictionary-add (spell-fu-get-personal-dictionary (format "%s-personal" lang)
11        ↪ personal-dict-file))))
12
13 (add-hook 'spell-fu-mode-hook
14   (lambda ()
15     (+spell-fu-register-dictionary +my/lang-main)
16     (+spell-fu-register-dictionary +my/lang-secondary))))

```

### 6.7.2 Grammarly

Use either eglot-grammarly or lsp-grammarly.

```

1 (package! grammarly
2   :recipe (:host github
3           :repo "emacs-grammarly/grammarly"))

```

```

1 (use-package! grammarly
2   :config
3   (grammarly-load-from-authinfo))

```

```

1 (package! eglot-grammarly
2   :disable (not (modulep! :tools lsp +eglot))
3   :recipe (:host github
4           :repo "emacs-grammarly/eglot-grammarly"))

```

### Eglot

```

1 (use-package! eglot-grammarly
2   :when (modulep! :tools lsp +eglot)
3   :commands (+lsp-grammarly-load)
4   :init
5   (defun +lsp-grammarly-load ()
6     "Load Grammarly LSP server for Eglot."
7     (interactive)
8     (require 'eglot-grammarly)
9     (call-interactively #'eglot)))

```

```

1 (package! lsp-grammarly
2   :disable (or (not (modulep! :tools lsp)) (modulep! :tools lsp +eglot))
3   :recipe (:host github
4           :repo "emacs-grammarly/lsp-grammarly"))

```

### LSP Mode

```

1 (use-package! lsp-grammarly
2   :when (and (modulep! :tools lsp) (not (modulep! :tools lsp +eglot)))
3   :commands (+lsp-grammarly-load +lsp-grammarly-toggle)
4   :init
5   (defun +lsp-grammarly-load ()
6     "Load Grammarly LSP server for LSP Mode."
7     (interactive)
8     (require 'lsp-grammarly)
9     (lsp-deferred)) ;; or (lsp)
10
11  (defun +lsp-grammarly-enabled-p ()
12    (not (member 'grammarly-ls lsp-disabled-clients)))
13
14  (defun +lsp-grammarly-enable ()
15    "Enable Grammarly LSP."
16    (interactive)
17    (when (not (+lsp-grammarly-enabled-p))
18      (setq lsp-disabled-clients (remove 'grammarly-ls lsp-disabled-clients))
19      (message "Enabled grammarly-ls"))
20    (+lsp-grammarly-load))
21
22  (defun +lsp-grammarly-disable ()
23    "Disable Grammarly LSP."
24    (interactive)
25    (when (+lsp-grammarly-enabled-p)
26      (add-to-list 'lsp-disabled-clients 'grammarly-ls)
27      (lsp-disconnect)
28      (message "Disabled grammarly-ls")))
29
30  (defun +lsp-grammarly-toggle ()
31    "Enable/disable Grammarly LSP."
32    (interactive)
33    (if (+lsp-grammarly-enabled-p)
34        (+lsp-grammarly-disable)
35        (+lsp-grammarly-enable)))
36
37  (after! lsp-mode
38    ;; Disable by default
39    (add-to-list 'lsp-disabled-clients 'grammarly-ls))
40
41  :config
42  (set-lsp-priority! 'grammarly-ls 1))

```

### 6.7.3 Grammalecte

```

1 (package! flycheck-grammalecte
2   :recipe (:host github
3           :repo "milouse/flycheck-grammalecte"))

```

```

1 (use-package! flycheck-grammalecte
2   :commands (flycheck-grammalecte-correct-error-at-point
3             grammalecte-conjugate-verb
4             grammalecte-define
5             grammalecte-define-at-point
6             grammalecte-find-synonyms
7             grammalecte-find-synonyms-at-point)
8   :init
9   (setq grammalecte-settings-file (expand-file-name "grammalecte/grammalecte-cache.el" doom-data-dir)
10        grammalecte-python-package-directory (expand-file-name "grammalecte/grammalecte" doom-data-dir))
11
12  (setq flycheck-grammalecte-report-spellcheck t
13        flycheck-grammalecte-report-grammar t
14        flycheck-grammalecte-report-apos nil
15        flycheck-grammalecte-report-esp nil)

```

```

16 flycheck-grammalecte-report-nbsp nil
17 flycheck-grammalecte-filters
18 '("(?m)^# ?-+.$"
19   ;; Ignore LaTeX equations (inline and block)
20   "\\$.*?\\$"
21   "(?s)\\\\begin{\\{(?:\\{?:equation.\\|align.\\|\\}\\}\\}\\}\\}\\end{\\{1}\\}")
22
23 (map! :leader :prefix ("l" . "custom")
24       (:prefix ("g" . "grammalecte")
25         :desc "Correct error at point" "p" #'flycheck-grammalecte-correct-error-at-point
26         :desc "Conjugate a verb" "V" #'grammalecte-conjugate-verb
27         :desc "Define a word" "W" #'grammalecte-define
28         :desc "Conjugate a verb at point" "w" #'grammalecte-define-at-point
29         :desc "Find synonyms" "S" #'grammalecte-find-synonyms
30         :desc "Find synonyms at point" "s" #'grammalecte-find-synonyms-at-point))
31
32 :config
33 (grammalecte-download-grammalecte)
34 (flycheck-grammalecte-setup)
35 (add-to-list 'flycheck-grammalecte-enabled-modes 'fountain-mode))

```

#### 6.7.4 LTeX/LanguageTool

Originally, LTeX LS stands for *L<sup>A</sup>T<sub>E</sub>X Language Server*, it acts as a Language Server for L<sup>A</sup>T<sub>E</sub>X, but not only. It can check the grammar and the spelling of several markup languages such as BibTeX, ConTeXt, L<sup>A</sup>T<sub>E</sub>X, Markdown, Org, reStructuredText... and others. Alongside, it provides interfacing with LanguageTool to implement natural language checking.

```

1 (after! lsp-ltex
2   (setq lsp-ltex-language "auto"
3         lsp-ltex-mother-tongue +my/lang-mother-tongue
4         flycheck-checker-error-threshold 1000)
5
6   (advice-add
7     '+lsp-ltex-setup :before
8     (lambda ()
9       (setq-local lsp-ui-sideline-show-diagnostics nil
10                  lsp-ui-sideline-show-code-actions nil
11                  lsp-ui-sideline-enable nil))))

```

#### 6.7.5 Go Translate (Google, Bing and DeepL)

```

1 (package! go-translate
2   :recipe (:host github
3           :repo "lorniu/go-translate"))

```

```

1 (use-package! go-translate
2   :commands (gts-do-translate
3             +gts-yank-translated-region
4             +gts-translate-with)
5   :init
6   ;; Your languages pairs
7   (setq gts-translate-list (list (list +my/lang-main +my/lang-secondary)
8                                   (list +my/lang-main +my/lang-mother-tongue)
9                                   (list +my/lang-secondary +my/lang-mother-tongue)
10                                  (list +my/lang-secondary +my/lang-main)))
11
12   (map! :localleader
13         :map (org-mode-map markdown-mode-map latex-mode-map text-mode-map)
14         :desc "Yank translated region" "R" #' +gts-yank-translated-region)
15

```

```

16 (map! :leader :prefix "l"
17       (:prefix ("G" . "go-translate")
18               :desc "Bing" "b" (lambda () (interactive) (+gts-translate-with 'bing))
19               :desc "DeepL" "d" (lambda () (interactive) (+gts-translate-with 'deepl))
20               :desc "Google" "g" (lambda () (interactive) (+gts-translate-with)))
21       :desc "Yank translated region" "R" #'(+gts-yank-translated-region)
22       :desc "gts-do-translate" "t" #'(gts-do-translate))
23
24 :config
25 ;; Config the default translator, which will be used by the command `gts-do-translate'
26 (setq gts-default-translator
27       (gts-translator
28         ;; Used to pick source text, from, to. choose one.
29         :picker (gts-prompt-picker)
30         ;; One or more engines, provide a parser to give different output.
31         :engines (gts-google-engine :parser (gts-google-summary-parser))
32         ;; Render, only one, used to consumer the output result.
33         :render (gts-buffer-render)))
34
35 ;; Custom texter which remove newlines in the same paragraph
36 (defclass +gts-translate-paragraph (gts-texter) ())
37
38 (cl-defmethod gts-text ((_ +gts-translate-paragraph))
39   (when (use-region-p)
40     (let ((text (buffer-substring-no-properties (region-beginning) (region-end))))
41       (with-temp-buffer
42         (insert text)
43         (goto-char (point-min))
44         (let ((case-fold-search nil))
45           (while (re-search-forward "\n[^\n]" nil t)
46             (replace-region-contents
47              (- (point) 2) (- (point) 1)
48              (lambda (&optional a b) " ")))
49           (buffer-string)))))
50
51 ;; Custom picker to use the paragraph texter
52 (defclass +gts-paragraph-picker (gts-picker)
53   ((texter :initarg :texter :initform (+gts-translate-paragraph))))
54
55 (cl-defmethod gts-pick ((o +gts-paragraph-picker))
56   (let ((text (gts-text (oref o texter))))
57     (when (or (null text) (zerop (length text)))
58       (user-error "Make sure there is any word at point, or selection exists"))
59     (let ((path (gts-path o text)))
60       (setq gts-picker-current-path path)
61       (cl-values text path))))
62
63 (defun +gts-yank-translated-region ()
64   (interactive)
65   (gts-translate
66    (gts-translator
67     :picker (+gts-paragraph-picker)
68     :engines (gts-google-engine)
69     :render (gts-kill-ring-render))))
70
71 (defun +gts-translate-with (&optional engine)
72   (interactive)
73   (gts-translate
74    (gts-translator
75     :picker (+gts-paragraph-picker)
76     :engines
77     (cond ((eq engine 'deepl)
78            (gts-deepl-engine
79             :auth-key ;; Get API key from ~/.authinfo.gpg (machine api-free.deepl.com)
80             (funcall
81              (plist-get (car (auth-source-search :host "api-free.deepl.com" :max 1))
82                        :secret))
83             :pro nil))
84            ((eq engine 'bing) (gts-bing-engine))
85            (t (gts-google-engine)))))

```

```
86 :render (gts-buffer-render))))))
```

## 6.8 System tools

### 6.8.1 Disk usage

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

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

### 6.8.2 Chezmoi

```
1 (package! chezmoi)
```

```
1 (use-package! chezmoi
2   :when CHEZMOI-P
3   :commands (chezmoi-write
4               chezmoi-magit-status
5               chezmoi-diff
6               chezmoi-ediff
7               chezmoi-find
8               chezmoi-write-files
9               chezmoi-open-other
10              chezmoi-template-buffer-display
11              chezmoi-mode)
12   :config
13   ;; Company integration
14   (when (modulep! :completion company)
15     (defun +chezmoi--company-backend-h ()
16       (require 'chezmoi-company)
17       (if chezmoi-mode
18         (add-to-list 'company-backends 'chezmoi-company-backend)
19         (delete 'chezmoi-company-backend 'company-backends)))
20
21     (add-hook 'chezmoi-mode-hook #' +chezmoi--company-backend-h))
22
23   ;; Integrate with evil mode by toggling template display when entering insert mode.
24   (when (modulep! :editor evil)
25     (defun +chezmoi--evil-insert-state-enter-h ()
26       "Run after evil-insert-state-entry."
27       (chezmoi-template-buffer-display nil (point))
28       (remove-hook 'after-change-functions #'chezmoi-template--after-change 1))
29
30     (defun +chezmoi--evil-insert-state-exit-h ()
31       "Run after evil-insert-state-exit."
32       (chezmoi-template-buffer-display nil)
33       (chezmoi-template-buffer-display t)
34       (add-hook 'after-change-functions #'chezmoi-template--after-change nil 1))
35
36     (defun +chezmoi--evil-h ()
37       (if chezmoi-mode
38         (progn
39           (add-hook 'evil-insert-state-entry-hook #' +chezmoi--evil-insert-state-enter-h nil 1)
40           (add-hook 'evil-insert-state-exit-hook #' +chezmoi--evil-insert-state-exit-h nil 1))
41         (progn
42           (remove-hook 'evil-insert-state-entry-hook #' +chezmoi--evil-insert-state-enter-h 1)
43           (remove-hook 'evil-insert-state-exit-hook #' +chezmoi--evil-insert-state-exit-h 1))))))
44
```



```

45 (add-hook 'chezmoi-mode-hook #'chezmoi--evil-h)))
46
47 (map! :leader :prefix ("l" . "custom")
48 (:prefix ("t" . "tools")
49 (:when CHEZMOI-P
50 :prefix ("c" . "chezmoi")
51 :desc "Magit status" "g" #'chezmoi-magit-status
52 :desc "Write" "w" #'chezmoi-write
53 :desc "Write files" "W" #'chezmoi-write-files
54 :desc "Find source" "f" #'chezmoi-find
55 :desc "Sync files" "s" #'chezmoi-sync-files
56 :desc "Diff" "d" #'chezmoi-diff
57 :desc "EDiff" "e" #'chezmoi-ediff
58 :desc "Open other" "o" #'chezmoi-open-other)))

```

### 6.8.3 Aweshell

```

1 (package! aweshell
2 :recipe (:host github
3 :repo "manateelazycat/aweshell"))

```

```

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

```

### 6.8.4 Lemon

```

1 (package! lemon
2 :recipe (:host nil
3 :repo "https://codeberg.org/emacs-weirdware/lemon.git"))

```

```

1 (use-package! lemon
2 :commands (lemon-mode lemon-display)
3 :config
4 (require 'lemon-cpu)
5 (require 'lemon-memory)
6 (require 'lemon-network)
7 (setq lemon-delay 5
8       lemon-refresh-rate 2
9       lemon-monitors
10      (list '((lemon-cpufreq-linux :display-opts '(:sparkline (:type gridded)))
11              (lemon-cpu-linux)
12              (lemon-memory-linux)
13              (lemon-linux-network-tx)
14              (lemon-linux-network-rx)))))

```

### 6.8.5 eCryptfs

```

1 (when ECRYPTFS-P
2 (defvar +ecryptfs-private-dir "Private")
3 (defvar +ecryptfs-buffer-name "*emacs-ecryptfs*")
4 (defvar +ecryptfs-config-dir (expand-file-name "~/.ecryptfs"))
5 (defvar +ecryptfs-passphrase-gpg (expand-file-name "~/.ecryptfs/my-pass.gpg"))
6 (defvar +ecryptfs--wrapping-independent-p (not (null (expand-file-name "wrapping-independent"
↪ +ecryptfs-config-dir))))
7 (defvar +ecryptfs--wrapped-passphrase-file (expand-file-name "wrapped-passphrase" +ecryptfs-config-dir))
8 (defvar +ecryptfs--mount-passphrase-sig-file (concat (expand-file-name +ecryptfs-private-dir
↪ +ecryptfs-config-dir) ".sig"))

```

```

9 (defvar +ecryptfs--mount-private-cmd "/sbin/mount.ecryptfs_private")
10 (defvar +ecryptfs--umount-private-cmd "/sbin/umount.ecryptfs_private")
11 (defvar +ecryptfs--passphrase
12   (lambda ()
13     (s-trim-right ;; To remove the new line
14       (epg-decrypt-file (epg-make-context)
15         +ecryptfs-passphrase-gpg
16         nil))))
17 (defvar +ecryptfs--encrypt-filenames-p
18   (not (eq 1
19     (with-temp-buffer
20       (insert-file-contents +ecryptfs--mount-passphrase-sig-file)
21       (count-lines (point-min) (point-max))))))
22 (defvar +ecryptfs--command-format
23   (if +ecryptfs--encrypt-filenames-p
24     "ecryptfs-insert-wrapped-passphrase-into-keyring %s '%s'"
25     "ecryptfs-unwrap-passphrase %s '%s' | ecryptfs-add-passphrase -"))
26
27 (defun +ecryptfs-mount-private ()
28   (interactive)
29   (unless (and (file-exists-p +ecryptfs--wrapped-passphrase-file)
30     (file-exists-p +ecryptfs--mount-passphrase-sig-file))
31     (error "Encrypted private directory \"%s\" is not setup properly."
32       +ecryptfs-private-dir)
33     (return))
34
35   (let ((try-again t))
36     (while (and
37       ;; In the first iteration, we try to silently mount the ecryptfs private directory,
38       ;; this would succeed if the key is available in the keyring.
39       (shell-command +ecryptfs--mount-private-cmd
40         +ecryptfs-buffer-name)
41       try-again)
42       (setq try-again nil)
43       (message "Encrypted filenames mode [%s]." (if +ecryptfs--encrypt-filenames-p "ENABLED" "DISABLED")))
44     (shell-command
45       (format +ecryptfs--command-format
46         +ecryptfs--wrapped-passphrase-file
47         (funcall +ecryptfs--passphrase))
48       +ecryptfs-buffer-name)
49     (message "Ecryptfs mount private.")))
50
51 (defun +ecryptfs-umount-private ()
52   (interactive)
53   (while (string-match-p "Sessions still open, not unmounting"
54     (shell-command-to-string +ecryptfs--umount-private-cmd)))
55   (message "Unmounted private directory."))
56
57 (map! :leader :prefix ("l" . "custom")
58   (:prefix ("t" . "tools")
59     (:when ECRYPTFS-P
60       :prefix ("e" . "ecryptfs")
61       :desc "eCryptfs mount private" "e" #' +ecryptfs-mount-private
62       :desc "eCryptfs un-mount private" "E" #' +ecryptfs-umount-private)))

```

## 6.9 Features

### 6.9.1 Workspaces

```

1 (map! :leader
2   (:when (modulep! :ui workspaces)
3     :prefix ("TAB" . "workspace")
4     :desc "Display tab bar" "TAB" #' +workspace/display
5     :desc "Switch workspace" "." #' +workspace/switch-to
6     :desc "Switch to last workspace" "$" #' +workspace/other ;; Modified
7     :desc "New workspace" "n" #' +workspace/new

```

```

8      :desc "New named workspace"      "N"    #'workspace/new-named
9      :desc "Load workspace from file" "l"    #'workspace/load
10     :desc "Save workspace to file"    "s"    #'workspace/save
11     :desc "Delete session"           "x"    #'workspace/kill-session
12     :desc "Delete this workspace"     "d"    #'workspace/delete
13     :desc "Rename workspace"          "r"    #'workspace/rename
14     :desc "Restore last session"       "R"    #'workspace/restore-last-session
15     :desc "Next workspace"            ">"    #'workspace/switch-right ;; Modified
16     :desc "Previous workspace"        "<"    #'workspace/switch-left  ;; Modified
17     :desc "Switch to 1st workspace"    "1"    #'workspace/switch-to-0
18     :desc "Switch to 2nd workspace"    "2"    #'workspace/switch-to-1
19     :desc "Switch to 3rd workspace"    "3"    #'workspace/switch-to-2
20     :desc "Switch to 4th workspace"    "4"    #'workspace/switch-to-3
21     :desc "Switch to 5th workspace"    "5"    #'workspace/switch-to-4
22     :desc "Switch to 6th workspace"    "6"    #'workspace/switch-to-5
23     :desc "Switch to 7th workspace"    "7"    #'workspace/switch-to-6
24     :desc "Switch to 8th workspace"    "8"    #'workspace/switch-to-7
25     :desc "Switch to 9th workspace"    "9"    #'workspace/switch-to-8
26     :desc "Switch to final workspace" "0"    #'workspace/switch-to-final))

```

### 6.9.2 Weather

```

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

```

```

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

```

### 6.9.3 OpenStreetMap

```

1 (package! osm)

1 (use-package! osm
2 :commands (osm-home
3            osm-search
4            osm-server
5            osm-goto
6            osm-gpx-show
7            osm-bookmark-jump)
8
9 :custom
10 ;; Take a look at the customization group `osm' for more options.
11 (osm-server 'default) ;; Configure the tile server
12 (osm-copyright t)    ;; Display the copyright information
13
14 :init
15 (setq osm-tile-directory (expand-file-name "osm" doom-data-dir))
16 ;; Load Org link support
17 (with-eval-after-load 'org
18   (require 'osm-ol)))

```

### 6.9.4 Islamic prayer times

```

1 (package! awqat
2   :recipe (:host github
3           :repo "zkry/awqat"))

```

```

1 (use-package! awqat
2   :commands (awqat-display-prayer-time-mode awqat-times-for-day)
3   :config
4   ;; Make sure `calendar-latitude' and `calendar-longitude' are set,
5   ;; otherwise, set them here.
6   (setq awqat-asr-hanafi nil
7         awqat-mode-line-format "  ${prayer} (${hours}h${minutes}m ")
8   (awqat-set-preset-french-muslims))

```

### 6.9.5 Info colors

Better colors for manual pages.

```

1 (package! info-colors)

```

```

1 (use-package! info-colors
2   :commands (info-colors-fontify-node))
3
4 (add-hook 'Info-selection-hook 'info-colors-fontify-node)

```

### 6.9.6 Zotero Zotxt

```

1 (package! zotxt)

```

```

1 (use-package! zotxt
2   :when ZOTERO-P
3   :commands org-zotxt-mode)

```

### 6.9.7 CRDT

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

```

1 (package! crdt)

```

```

1 (use-package! crdt
2   :commands (crdt-share-buffer
3             crdt-connect)
4   :init
5   (cond
6     (TUNTOX-P (setq crdt-use-tuntox t
7                     crdt-tuntox-password-in-url t))
8     (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`.

```
1 (package! ag)

1 (use-package! ag
2   :when AG-P
3   :commands (ag
4             ag-files
5             ag-regexp
6             ag-project
7             ag-project-files
8             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.

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

1 (use-package! page-break-lines
2   :diminish
3   :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.

```
1 (use-package! eaf
2   :when EAF-P
3   :load-path EAF-DIR
4   :commands (eaf-open
5             eaf-open-browser
6             eaf-open-jupyter
7             +eaf-open-mail-as-html)
8   :init
9   (defvar +eaf-enabled-apps
10    '(org browser mindmap jupyter org-previewer markdown-previewer file-sender video-player))
11
12   (defun +eaf-app-p (app-symbol)
13     (memq app-symbol +eaf-enabled-apps))
14
15   (when (+eaf-app-p 'browser)
16     ;; Make EAF Browser my default browser
17     (setq browse-url-browser-function #'eaf-open-browser)
18     (defalias 'browse-web #'eaf-open-browser)
19
20     (map! :localleader
21          :map (mu4e-headers-mode-map mu4e-view-mode-map)
22          :desc "Open mail as HTML" "h" #'eaf-open-mail-as-html
23          :desc "Open URL (EAF)" "o" #'eaf-open-browser))
24
25   (when (+eaf-app-p 'pdf-viewer)
```

```

26 (after! org
27   ;; Use EAF PDF Viewer in Org
28   (defun +eaf--org-open-file-fn (file &optional link)
29     "An wrapper function on `eaf-open'."
30     (eaf-open file))
31
32   ;; use `emacs-application-framework' to open PDF file: link
33   (add-to-list 'org-file-apps '("\\.pdf\\\\" . +eaf--org-open-file-fn)))
34
35 (after! latex
36   ;; Link EAF with the LaTeX compiler in emacs. When a .tex file is open,
37   ;; the Command>Compile and view (C-c C-a) option will compile the .tex
38   ;; file into a .pdf file and display it using EAF. Double clicking on the
39   ;; PDF side jumps to editing the clicked section.
40   (add-to-list 'TeX-command-list '("XeLaTeX" "%\xelatex --synctex=1%(mode)%" %t TeX-run-TeX nil t))
41   (add-to-list 'TeX-view-program-list '("eaf" eaf-pdf-synctex-forward-view))
42   (add-to-list 'TeX-view-program-selection '(output-pdf "eaf"))))
43
44 :config
45 ;; Generic
46 (setq eaf-start-python-process-when-require t
47       eaf-kill-process-after-last-buffer-closed t
48       eaf-fullscreen-p nil)
49
50 ;; Debug
51 (setq eaf-enable-debug nil)
52
53 ;; Web engine
54 (setq eaf-webengine-font-family (symbol-name (font-get doom-font :family))
55       eaf-webengine-fixed-font-family (symbol-name (font-get doom-font :family))
56       eaf-webengine-serif-font-family (symbol-name (font-get doom-serif-font :family))
57       eaf-webengine-font-size 16
58       eaf-webengine-fixed-font-size 16
59       eaf-webengine-enable-scrollbar t
60       eaf-webengine-scroll-step 200
61       eaf-webengine-default-zoom 1.25
62       eaf-webengine-show-hover-link t
63       eaf-webengine-download-path "~/Downloads"
64       eaf-webengine-enable-plugin t
65       eaf-webengine-enable-javascript t
66       eaf-webengine-enable-javascript-access-clipboard t)
67
68 (when (display-graphic-p)
69   (require 'eaf-all-the-icons)
70   (mapc (lambda (v) (eaf-all-the-icons-icon (car v)))
71         eaf-all-the-icons-alist))
72
73 ;; Browser settings
74 (when (+eaf-app-p 'browser)
75   (setq eaf-browser-continue-where-left-off t
76         eaf-browser-dark-mode nil ;; "follow"
77         eaf-browser-enable-adblocker t
78         eaf-browser-enable-autofill nil
79         eaf-browser-remember-history t
80         eaf-browser-ignore-history-list '("google.com/search" "file:///")
81         eaf-browser-text-selection-color "auto"
82         eaf-browser-translate-language +my/lang-main
83         eaf-browser-blank-page-url "https://www.duckduckgo.com"
84         eaf-browser-chrome-history-file "~/.config/google-chrome/Default/History"
85         eaf-browser-default-search-engine "duckduckgo"
86         eaf-browser-continue-where-left-off t
87         eaf-browser-aria2-auto-file-renaming t)
88
89   (require 'eaf-browser)
90
91   (defun +eaf-open-mail-as-html ()
92     "Open the html mail in EAF Browser."
93     (interactive)
94     (let ((msg (mu4e-message-at-point t)))
95       ;; Bind browse-url-browser-function locally, so it works

```

```

96      ;; even if EAF Browser is not set as a default browser.
97      (browse-url-browser-function #'eaf-open-browser))
98      (if msg
99        (mu4e-action-view-in-browser msg)
100        (message "No message at point.))))))
101
102  ;; File manager settings
103  (when (+eaf-app-p 'file-manager)
104    (setq eaf-file-manager-show-preview nil
105          eaf-find-alternate-file-in-dired t
106          eaf-file-manager-show-hidden-file t
107          eaf-file-manager-show-icon t)
108    (require 'eaf-file-manager))
109
110  ;; File Browser
111  (when (+eaf-app-p 'file-browser)
112    (require 'eaf-file-browser))
113
114  ;; PDF Viewer settings
115  (when (+eaf-app-p 'pdf-viewer)
116    (setq eaf-pdf-dark-mode "follow"
117          eaf-pdf-show-progress-on-page nil
118          eaf-pdf-dark-exclude-image t
119          eaf-pdf-notify-file-changed t)
120    (require 'eaf-pdf-viewer))
121
122  ;; Org
123  (when (+eaf-app-p 'rss-reader)
124    (setq eaf-rss-reader-split-horizontally nil
125          eaf-rss-reader-web-page-other-window t)
126    (require 'eaf-org))
127
128  ;; Org
129  (when (+eaf-app-p 'org)
130    (require 'eaf-org))
131
132  ;; Mail
133  ;; BUG The `eaf-open-mail-as-html' is not working,
134  ;; I use `+eaf-open-mail-as-html' instead
135  (when (+eaf-app-p 'mail)
136    (require 'eaf-mail))
137
138  ;; Org Previewer
139  (when (+eaf-app-p 'org-previewer)
140    (setq eaf-org-dark-mode "follow")
141    (require 'eaf-org-previewer))
142
143  ;; Markdown Previewer
144  (when (+eaf-app-p 'markdown-previewer)
145    (setq eaf-markdown-dark-mode "follow")
146    (require 'eaf-markdown-previewer))
147
148  ;; Jupyter
149  (when (+eaf-app-p 'jupyter)
150    (setq eaf-jupyter-dark-mode "follow"
151          eaf-jupyter-font-family (symbol-name (font-get doom-font :family))
152          eaf-jupyter-font-size 13)
153    (require 'eaf-jupyter))
154
155  ;; Mindmap
156  (when (+eaf-app-p 'mindmap)
157    (setq eaf-mindmap-dark-mode "follow"
158          eaf-mindmap-save-path "~/Dropbox/Mindmap")
159    (require 'eaf-mindmap))
160
161  ;; File Sender
162  (when (+eaf-app-p 'file-sender)
163    (require 'eaf-file-sender))
164
165  ;; Music Player

```

```

166 (when (+eaf-app-p 'music-player)
167   (require 'eaf-music-player))
168
169 ;; Video Player
170 (when (+eaf-app-p 'video-player)
171   (setq eaf-video-player-keybinding
172     '(("p" . "toggle_play")
173       ("q" . "close_buffer")
174       ("h" . "play_backward")
175       ("l" . "play_forward")
176       ("j" . "decrease_volume")
177       ("k" . "increase_volume")
178       ("f" . "toggle_fullscreen")
179       ("R" . "restart"))))
180   (require 'eaf-video-player))
181
182 ;; Image Viewer
183 (when (+eaf-app-p 'image-viewer)
184   (require 'eaf-image-viewer))
185
186 ;; Git
187 (when (+eaf-app-p 'git)
188   (require 'eaf-git))
189
190 ;; Fix EVIL keybindings
191 (after! evil
192   (require 'eaf-evil)
193   (define-key key-translation-map (kbd "SPC")
194     (lambda (prompt)
195       (if (derived-mode-p 'eaf-mode)
196         (pcase eaf--buffer-app-name
197           ("browser" (if (eaf-call-sync "execute_function" eaf--buffer-id "is_focus")
198                         (kbd "SPC")
199                         (kbd eaf-evil-leader-key)))
200           ("pdf-viewer" (kbd eaf-evil-leader-key))
201           ("image-viewer" (kbd eaf-evil-leader-key))
202           ("music-player" (kbd eaf-evil-leader-key))
203           ("video-player" (kbd eaf-evil-leader-key))
204           ("file-sender" (kbd eaf-evil-leader-key))
205           ("mindmap" (kbd eaf-evil-leader-key))
206           (_ (kbd "SPC"))))
207         (kbd "SPC")))))

```

### 6.9.11 Bitwarden

```

1 (package! bitwarden
2   :recipe (:host github
3           :repo "seanfarley/emacs-bitwarden"))

```

```

1 (use-package! bitwarden
2   ;;:config
3   ;;(bitwarden-auth-source-enable)
4   :when BITWARDEN-P
5   :init
6   (setq bitwarden-automatic-unlock
7     (lambda ()
8       (require 'auth-source)
9       (if-let* ((matches (auth-source-search :host "bitwarden.com" :max 1))
10                (entry (nth 0 matches))
11                (email (plist-get entry :user))
12                (pass (plist-get entry :secret)))
13         (progn
14           (setq bitwarden-user email)
15           (if (functionp pass) (funcall pass) pass))
16         ""))))

```



### 6.9.12 PDF tools

**Dark mode** 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`.

```

1 (after! pdf-tools
2   (add-hook! 'pdf-view-mode-hook
3     (when (memq doom-theme '(modus-vivendi doom-one doom-dark+ doom-vibrant))
4       ;; TODO: find a more generic way to detect if we are in a dark theme
5       (pdf-view-midnight-minor-mode 1)))
6
7   ;; Color the background, so we can see the PDF page borders
8   ;; https://protesilaos.com/emacs/modus-themes#h:ff69dfe1-29c0-447a-915c-b5ff7c5509cd
9   (defun +pdf-tools-backdrop ()
10     (face-remap-add-relative
11       'default
12       `(:background ,(if (memq doom-theme '(modus-vivendi modus-operandi))
13                           (modus-themes-color 'bg-alt)
14                           (doom-color 'bg-alt)))))
15
16   (add-hook 'pdf-tools-enabled-hook #' +pdf-tools-backdrop))
17
18 (after! pdf-links
19   ;; Tweak for Modus and `pdf-links'
20   (when (memq doom-theme '(modus-vivendi modus-operandi))
21     ;; https://protesilaos.com/emacs/modus-themes#h:2659d13e-b1a5-416c-9a89-7c3ce3a76574
22     (let ((spec (apply #'append
23                       (mapcar
24                        (lambda (name)
25                          (list name
26                                (face-attribute 'pdf-links-read-link
27                                                  name nil 'default))))
28                        '(:family :width :weight :slant)))))
29       (setq pdf-links-read-link-convert-commands
30         `("-density"      "96"
31           "-family"      ,(plist-get spec :family)
32           "-stretch"      ,(let* ((width (plist-get spec :width))
33                                   (name (symbol-name width)))
34                               (replace-regexp-in-string "-" ""
35                                                         (capitalize name)))
35           "-weight"      ,(pcase (plist-get spec :weight)
36                               ('ultra-light "Thin")
37                               ('extra-light "ExtraLight")
38                               ('light       "Light")
39                               ('semi-bold   "SemiBold")
40                               ('bold        "Bold")
41                               ('extra-bold   "ExtraBold")
42                               ('ultra-bold   "Black")
43                               (_weight     "Normal"))
44           "-style"        ,(pcase (plist-get spec :slant)
45                               ('italic      "Italic")
46                               ('oblique     "Oblique")
47                               (_slant       "Normal"))
48           "-pointsize"    "%P"
49           "-undercolor"    "%f"
50           "-fill"          "%b"
51           "-draw"          "text %X,%Y '%c'")))))
52

```

### 6.9.13 LTDR

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

```
1 (package! tldr)
```

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

### 6.9.14 FZF

```
1 (package! fzf)
```

```
1 (after! evil
2   (evil-define-key 'insert fzf-mode-map (kbd "ESC") #'term-kill-subjob))
3
4 (define-minor-mode fzf-mode
5   "Minor mode for the FZF buffer"
6   :init-value nil
7   :lighter " FZF"
8   :keymap '(("C-c" . term-kill-subjob)))
9
10 (defadvice! doom-fzf--override-start-args-a (original-fn &rest args)
11   "Set the FZF minor mode with the fzf buffer."
12   :around #'fzf/start
13   (message "called with args %S" args)
14   (apply original-fn args))
15
16 ;; set the FZF buffer to fzf-mode so we can hook ctrl+c
17 (set-buffer "*fzf*")
18 (fzf-mode))
19
20 (defvar fzf/args
21   "-x --print-query -m --tiebreak=index --expect=ctrl-v,ctrl-x,ctrl-t")
22
23 (use-package! fzf
24   :commands (fzf fzf-projectile fzf-hg fzf-git fzf-git-files fzf-directory fzf-git-grep))
```

### 6.9.15 Binary files

Inspired by this discussion.

Add the new `nhexl-mode` which allows editing files in Hex mode.

```
1 (package! nhexl-mode)
```

```
1 (defun +buffer-binary-p (&optional buffer)
2   "Return whether BUFFER or the current buffer is binary.
3
4   A binary buffer is defined as containing at least one null byte.
5
6   Returns either nil, or the position of the first null byte."
7   (with-current-buffer (or buffer (current-buffer))
8     (save-excursion (goto-char (point-min))
9       (search-forward (string ?\x00) nil t 1))))
10
11 (defun +hexl--buffer-p ()
12   (and (+buffer-binary-p)
13     ;; Executables are viewed with objdump mode
14     (not (+buffer-objdump-p))))
```

```

15
16 (defun +hexl-if-binary ()
17   "If `hexl-mode' is not already active, and the current buffer
18   is binary, activate `hexl-mode'."
19   (interactive)
20   (unless (eq major-mode 'hexl-mode)
21     (when (+hexl--buffer-p)
22       (hexl-mode))))
23
24 (add-to-list 'magic-fallback-mode-alist '(+hexl--buffer-p . +hexl-if-binary) t)

```

### 6.9.16 Objdump mode

Define a major mode (`objdump-disassemble-mode`) to display executable files as assembly code using `objdump`. The file types are detected using the `file` utility.

```

1 (defun +buffer-objdump-p (&optional buffer file)
2   "Can the BUFFER be viewed as a disassembled code with objdump."
3   (when-let ((file (or file (buffer-file-name (or buffer (current-buffer))))))
4     (and
5       (file-exists-p file)
6       (not (file-directory-p file))
7       (not (zerop (file-attribute-size (file-attributes file))))
8       (not (string-match-p
9         "file format not recognized"
10        (with-temp-buffer
11          (shell-command (format "objdump --file-headers %s"
12                                (shell-quote-argument "/home/hacko/Softwares/Kasparov/Kasparov
↪ Chessmate/KasparovChess.Stats"))
13          (current-buffer))
14          (buffer-string)))))))
15
16 (when OBJDUMP-P
17   (define-derived-mode objdump-disassemble-mode
18     asm-mode "Objdump Mode"
19     "Major mode for viewing executable files disassembled using objdump."
20     (if (not (+buffer-objdump-p))
21       (message "Objdump can not be used with this buffer.")
22       (let ((file (buffer-file-name))
23             (buffer-read-only nil))
24         (erase-buffer)
25         (message "Disassembling file \"%s\" using objdump." (file-name-nondirectory file))
26         (call-process "objdump" nil (current-buffer) nil "-d" file)
27         (set-buffer-modified-p nil)
28         (goto-char (point-min))
29         (view-mode)
30         (set-visited-file-name nil t))))
31
32 (add-to-list 'magic-fallback-mode-alist '(+buffer-objdump-p . objdump-disassemble-mode) t))

```

## 6.10 Fun

### 6.10.1 Speed Type

A game to practice speed typing in Emacs.

```

1 (package! speed-type)

1 (use-package! speed-type
2   :commands (speed-type-text))

```

### 6.10.2 2048 Game

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

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

### 6.10.3 Snow

Let it snow in Emacs!

```
1 (package! snow)
```

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

### 6.10.4 xkcd

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

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

## 7 Applications

### 7.1 Calendar

```
1 (setq calendar-latitude 48.7
2       calendar-longitude 2.17
3       calendar-location-name "Orsay, FR"
4       calendar-time-display-form
5       '(24-hours ":" minutes
6         (if time-zone " (" time-zone (if time-zone ")"))))
```

### 7.2 e-Books (nov)

```
1 (package! nov)
```

Use nov to read EPUB e-books.

```

1 (use-package! nov
2   :mode ("\\.epub\\" . nov-mode)
3   :config
4   (map! :map nov-mode-map
5     :n "RET" #'nov-scroll-up)
6
7   (defun doom-modeline-segment--nov-info ()
8     (concat " "
9       (propertize (cdr (assoc 'creator nov-metadata))
10         'face 'doom-modeline-project-parent-dir)
11       " "
12       (cdr (assoc 'title nov-metadata))
13       " "
14       (propertize (format "%d/%d" (1+ nov-documents-index) (length nov-documents))
15         'face 'doom-modeline-info)))
16
17   (advice-add 'nov-render-title :override #'ignore)
18
19   (defun +nov-mode-setup ()
20     (face-remap-add-relative 'variable-pitch
21       :family "Merriweather"
22       :height 1.4
23       :width 'semi-expanded)
24     (face-remap-add-relative 'default :height 1.3)
25     (setq-local line-spacing 0.2
26       next-screen-context-lines 4
27       shr-use-colors nil)
28     (require 'visual-fill-column nil t)
29     (setq-local visual-fill-column-center-text t
30       visual-fill-column-width 80
31       nov-text-width 80)
32     (visual-fill-column-mode 1)
33     (hl-line-mode -1)
34
35     (add-to-list '+lookup-definition-functions
36       #'lookup/dictionary-definition)
37
38     (setq-local mode-line-format
39       `(:eval
40         (doom-modeline-segment--workspace-name))
41         (:eval
42         (doom-modeline-segment--window-number))
43         (:eval
44         (doom-modeline-segment--nov-info))
45         ,propertize
46         " %P "
47         'face 'doom-modeline-buffer-minor-mode)
48         ,propertize
49         " "
50         'face (if (doom-modeline--active) 'mode-line 'mode-line-inactive)
51         'display `((space
52           :align-to
53           (- (+ right right-fringe right-margin)
54             ,(* (let ((width (doom-modeline--font-width)))
55               (or (and (= width 1) 1)
56                 (/ width (frame-char-width) 1.0)))
57             (string-width
58               (format-mode-line (cons "" '(:eval
59 ↪ (doom-modeline-segment--major-mode))))))))))
59         (:eval (doom-modeline-segment--major-mode))))))
60
61   (add-hook 'nov-mode-hook #' +nov-mode-setup))

```

## 7.3 News feed (elfeed)

Set RSS news feeds

```

1 (setq elfeed-feeds
2   '("https://this-week-in-rust.org/rss.xml"
3     "https://planet.emacslife.com/atom.xml"
4     "https://www.omgubuntu.co.uk/feed"
5     "https://itsfoss.com/feed"
6     "https://linuxhandbook.com/feed"
7     "https://spectrum.ieee.org/rss/robotics/fulltext"
8     "https://spectrum.ieee.org/rss/aerospace/fulltext"
9     "https://spectrum.ieee.org/rss/computing/fulltext"
10    "https://spectrum.ieee.org/rss/blog/automaton/fulltext"
11    "https://developers.redhat.com/blog/feed"
12    "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:

```

1 echo "-u <USERNAME> -d <DOMAINE> -p <PASSWORD> -s <SERVER_IP>" \
2 | gpg -c > sslvpn.gpg

```

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

```

1 #!/bin/bash
2
3 if ! command -v netExtender &> /dev/null
4 then
5   echo "netExtender not found, installing from AUR using 'yay'"
6   yay -S netextender
7 fi
8
9 MY_LOGIN_PARAMS_FILE="$HOME/.ssh/sslvpn.gpg"
10
11 echo "Y\n" | netExtender --auto-reconnect \
12 $(gpg -q --for-your-eyes-only --no-tty -d "${MY_LOGIN_PARAMS_FILE}")

```

### 7.4.2 Emacs + NetExtender

```

1 (when NETEXTENDER-P
2   (defvar +netextender-process-name "netextender")
3   (defvar +netextender-buffer-name " *NetExtender*")
4   (defvar +netextender-command '("~/local/bin/netextender"))
5
6   (defun +netextender-start ()
7     "Launch a NetExtender VPN session"
8     (interactive)
9     (unless (get-process +netextender-process-name)
10      (if (make-process :name +netextender-process-name
11                        :buffer +netextender-buffer-name
12                        :command +netextender-command)
13          (message "Started NetExtender VPN session")
14          (message "Cannot start NetExtender"))))
15
16   (defun +netextender-kill ()
17     "Kill the created NetExtender VPN session"
18     (interactive)
19     (when (get-process +netextender-process-name)
20      (if (kill-buffer +netextender-buffer-name)
21          (message "Killed NetExtender VPN session")

```

```

22      (message "Cannot kill NetExtender")))))
23
24  (map! :leader
25        :prefix ("l")
26        (:prefix ("t")
27              (:prefix ("n" . "netExtender")
28                    :desc "Start" "s" #'netextender-start
29                    :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 msmtplib)
- 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:

```

1  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:

```

1  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:

```

1  # mbsync config file
2  # GLOBAL OPTIONS
3  BufferLimit 50mb           # Global option: Default buffer size is 10M, too small for modern machines.
4  Sync All                   # Channels global: Sync everything "Pull Push New ReNew Delete Flags" (default
   ↪ option)
5  Create Both                # Channels global: Automatically create missing mailboxes on both sides
6  Expunge Both               # Channels global: Delete messages marked for deletion on both sides
7  CopyArrivalDate yes        # Channels global: Propagate arrival time with the messages
8
9  # SECTION (IMAP4 Accounts)
10 IMAPAccount work           # IMAP Account name
11 Host mail.host.ccc         # The host to connect to
12 User user@host.ccc         # Login user name
13 SSLVersions TLSv1.2 TLSv1.1 # Supported SSL versions
14 # Extract password from encrypted ~/.authinfo.gpg
15 # File format: "machine <SERVER> login <LOGIN> port <PORT> password <PASSWORD>"
16 # This uses sed to extract <PASSWORD> from line matching the account's <SERVER>
17 PassCmd "gpg2 -q --for-your-eyes-only --no-tty --logger-file /dev/null --batch -d ~/.authinfo.gpg | awk
   ↪ '/machine smtp.domain.tld/ {print $NF}'"
18 AuthMechs *                # Authentication mechanisms
19 SSLType IMAPS               # Protocol (STARTTLS/IMAPS)
20 CertificateFile /etc/ssl/certs/ca-certificates.crt
21 # END OF SECTION
22 # IMPORTANT NOTE: you need to keep the blank line after each section
23
24 # SECTION (IMAP Stores)
25 IMAPStore work-remote      # Remote storage name
26 Account work               # Associated account
27 # END OF SECTION
28
29 # SECTION (Maildir Stores)
30 MaildirStore work-local    # Local storage (create directories with mkdir -p ~/Maildir/<ACCOUNT-NAME>)
31 Path ~/Maildir/work/       # The local store path
32 Inbox ~/Maildir/work/Inbox # Location of the INBOX
33 SubFolders Verbatim        # Download all sub-folders
34 # END OF SECTION
35
36 # Connections specify links between remote and local folders
37 # they are specified using patterns, which match remote mail
38 # folders. Some commonly used patterns include:
39 #
40 # - "*" to match everything
41 # - "!DIR" to exclude "DIR"
42 # - "DIR" to match DIR
43 #
44 # SECTION (Channels)
45 Channel work               # Channel name
46 Far :work-remote:          # Connect remote store
47 Near :work-local:          # to the local one
48 Patterns "INBOX" "Drafts" "Sent" "Archives/*" "Spam" "Trash"
49 SyncState *                # Save state in near side mailbox file ".mbsyncstate"
50 # END OF SECTION
51
52 # =====
53
54 IMAPAccount gmail
55 Host imap.gmail.com
56 User user@gmail.com
57 PassCmd "gpg2 -q --for-your-eyes-only --no-tty --logger-file /dev/null --batch -d ~/.authinfo.gpg | awk
   ↪ '/machine smtp.domain.tld/ {print $NF}'"
58 AuthMechs LOGIN
59 SSLType IMAPS
60 CertificateFile /etc/ssl/certs/ca-certificates.crt
61
62 IMAPStore gmail-remote
63 Account gmail
64
65 MaildirStore gmail-local
66 Path ~/Maildir/gmail/
67 Inbox ~/Maildir/gmail/Inbox

```



```

68
69 # For Gmail, I like to make multiple channels, one for each remote directory
70 # this is a trick to rename remote "[Gmail]/mailbox" to "mailbox"
71 Channel gmail-inbox
72 Far :gmail-remote:
73 Near :gmail-local:
74 Patterns "INBOX"
75 SyncState *
76
77 Channel gmail-trash
78 Far :gmail-remote: "[Gmail]/Trash"
79 Near :gmail-local: "Trash"
80 SyncState *
81
82 Channel gmail-drafts
83 Far :gmail-remote: "[Gmail]/Drafts"
84 Near :gmail-local: "Drafts"
85 SyncState *
86
87 Channel gmail-sent
88 Far :gmail-remote: "[Gmail]/Sent Mail"
89 Near :gmail-local: "Sent Mail"
90 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 `~/.authinfo.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, `~/.msmtp.rc`, 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 `~/.msmtp.rc`.

```

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 passwordval          "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 passwordval          "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 msmtplib instead of smtpmail
17 (setq sendmail-program (executable-find "msmtplib"))
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 +mu4e-long-query (query oper arg-list)
66     (concat "(" (+str-join (concat " " oper " ") (mapcar (lambda (addr) (format "%s:%s" query addr))
67       arg-list)) ")))
68
69   ;; Build a query to match mails send from "me" with "me" in BCC
70   (let ((bcc-query (+mu4e-long-query "bcc" "or" +my-addresses))
71         (from-query (+mu4e-long-query "from" "or" +my-addresses)))
72     (add-to-list
73       'mu4e-bookmarks
74       (list :name "My black copies" :query (format "maildir:/*inbox/ and %s and %s" from-query bcc-query)
75         :key ?k) t)))
76
77 ;; `mu4e-alert' configuration
78 ;; Use a nicer icon in alerts
79 (setq mu4e-alert-icon "/usr/share/icons/Papirus/64x64/apps/mail-client.svg")
80
81 (defun +mu4e-alert-helper-name-or-email (msg)
82   (let* ((from (car (plist-get msg :from)))
83          (name (plist-get from :name)))
84     (if (or (null name) (eq name ""))
85         (plist-get from :email)
86         name)))

```

```

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              "\n• "
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))))
107              filtered-mails))))))
108
109 ;; I use auto-hiding task manager, setting window
110 ;; urgency shows the entier task bar (in KDE), which I find annoying.
111 (setq mu4e-alert-set-window-urgency nil
112       mu4e-alert-grouped-mail-notification-formatter #' +mu4e-alert-grouped-mail-notif-formatter)
113
114 ;; Org-Msg stuff
115 ;; org-msg-[signature/greeting-fmt] are separately set for each account
116 (setq mail-user-agent 'mu4e-user-agent) ;; Needed by OrgMsg
117 (require 'org-msg)
118 (setq org-msg-convert-citation t
119       org-msg-default-alternatives
120       '(new . (utf-8 html))
121         (reply-to-html . (utf-8 html))
122         (reply-to-text . (utf-8 html))))
123
124 (map! :map org-msg-edit-mode-map
125      :after org-msg
126      :n "G" #'org-msg-goto-body)
127
128 (map! :localleader
129      :map (mu4e-headers-mode-map mu4e-view-mode-map)
130      :desc "Open URL in Brave" "b" #'browse-url-chrome ;; Brave
131      :desc "Open URL in Firefox" "f" #'browse-url-firefox)
132
133 ;; I like to always BCC myself
134 (defun +bbc-me ()
135   "Add my email to BCC."
136   (save-excursion (message-add-header (format "Bcc: %s\n" +my-bcc-trash))))
137
138 (add-hook 'mu4e-compose-mode-hook '+bbc-me)
139
140 ;; Load my accounts
141 (load! "lisp/private/+mu4e-accounts.el")
142
143 ;; iCalendar / Org
144 (mu4e-icalendar-setup)
145 (setq mu4e-icalendar-trash-after-reply nil
146       mu4e-icalendar-diary-file "~/Dropbox/Org/diary-invitations.org"
147       gnus-icalendar-org-capture-file "~/Dropbox/Org/notes.org"
148       gnus-icalendar-org-capture-headline '("Calendar"))
149
150 ;; To enable optional iCalendar->Org sync functionality
151 ;; NOTE: both the capture file and the headline(s) inside must already exist
152 (gnus-icalendar-org-setup))

```

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

```

1 (set-email-account!
2   "Work" ;; Account label
3
4   ;; Mu4e folders
5   '( (mu4e-sent-folder      . "/work-dir/Sent")
6       (mu4e-drafts-folder   . "/work-dir/Drafts")
7       (mu4e-trash-folder    . "/work-dir/Trash")
8       (mu4e-refile-folder    . "/work-dir/Archive")
9
10      ;; Org-msg template (signature and greeting)
11      (org-msg-greeting-fmt   . "Hello%s,")
12      (org-msg-signature      . "
13
14  Regards,
15
16  #+begin_signature
17  -----
18  *Abdelhak BOUGOUFFA* \\\
19  /PhD. Candidate in Robotics | R&D Engineer/ \\\
20  /Paris-Saclay University - SATIE/MOSS | ez-Wheel/ \\\
21  #+end_signature")
22
23      ;; 'smtpmail' options, no need for these when using 'msmtp'
24      (smtpmail-smtp-user      . "username@server.com")
25      (smtpmail-smtp-server     . "smtps.server.com")
26      (smtpmail-stream-type     . ssl)
27      (smtpmail-smtp-service    . 465)
28
29      ;; By default, 'smtpmail' will try to send mails without authentication, and if rejected,
30      ;; it tries to send credentials. This behavior broke my configuration. So I set this
31      ;; variable to tell 'smtpmail' to require authentication for our server (using a regex).
32      (smtpmail-servers-requiring-authorization . "smtps\\.server\\.com"))
33
34   t) ;; Use as default/fallback account
35
36   ;; Set another account
37   (set-email-account!
38     "Gmail"
39     '( (mu4e-sent-folder      . "/gmail-dir/Sent")
40         (mu4e-drafts-folder   . "/gmail-dir/Drafts")
41         (mu4e-trash-folder    . "/gmail-dir/Trash")
42         (mu4e-refile-folder    . "/gmail-dir/Archive")
43         (org-msg-greeting-fmt   . "Hello%s,")
44         (org-msg-signature      . "-- SIGNATURE")
45
46         ;; No need for these when using 'msmtp'
47         (smtpmail-smtp-user      . "username@gmail.com")
48         (smtpmail-smtp-server     . "smtp.googlemail.com")
49         (smtpmail-stream-type     . starttls)
50         (smtpmail-smtp-service    . 587)
51         ...)
52
53     ;; Tell Doom's mu4e module to override some commands to fix issues on Gmail accounts
54     (setq +mu4e-gmail-accounts '(("username@gmail.com" . "/gmail-dir"))))

```

### 7.5.4 Dashboard

```

1 (after! mu4e
2   ;; Fix icons
3   (defun +mu4e-initialise-icons ()
4     (setq mu4e-use-fancy-chars t
5           mu4e-headers-draft-mark (cons "D" (+mu4e-normalised-icon "edit" :set "material"))
6           mu4e-headers-flagged-mark (cons "F" (+mu4e-normalised-icon "flag" :set "material"))
7           mu4e-headers-new-mark (cons "N" (+mu4e-normalised-icon "file_download" :set "material" :color
8
9   ↪ "dred"))))

```

```

8      mu4e-headers-passed-mark      (cons "P" (+mu4e-normalised-icon "forward" :set "material"))
9      mu4e-headers-replied-mark     (cons "R" (+mu4e-normalised-icon "reply" :set "material"))
10     mu4e-headers-seen-mark         (cons "S" "")
11     mu4e-headers-trashed-mark      (cons "T" (+mu4e-normalised-icon "delete" :set "material"))
12     mu4e-headers-attach-mark       (cons "a" (+mu4e-normalised-icon "attach_file" :set "material"))
13     mu4e-headers-encrypted-mark    (cons "x" (+mu4e-normalised-icon "lock" :set "material"))
14     mu4e-headers-signed-mark       (cons "s" (+mu4e-normalised-icon "verified_user" :set "material" :color
↪ "dpurple"))
15     mu4e-headers-unread-mark       (cons "u" (+mu4e-normalised-icon "remove_red_eye" :set "material" :color
↪ "dred"))
16     mu4e-headers-list-mark         (cons "l" (+mu4e-normalised-icon "list" :set "material"))
17     mu4e-headers-personal-mark      (cons "p" (+mu4e-normalised-icon "person" :set "material"))
18     mu4e-headers-calendar-mark     (cons "c" (+mu4e-normalised-icon "date_range" :set "material"))))
19
20     (+mu4e-initialise-icons))

```

### 7.5.5 Save all attachements

```

1  (after! mu4e
2  ;; From https://github.com/sje30/emacs/blob/d7e21b94c79a5b6f244f33faf514036226e183c/mu4e-view-save-all-attach_
↪ ments.el
3
4  (defun +cleanse-subject (sub)
5    (replace-regexp-in-string "[^A-Z0-9]+" "-" (downcase sub)))
6
7  (defun +mu4e-view-save-all-attachments (&optional arg)
8    "Save all MIME parts from current mu4e gnus view buffer."
9    ;; Copied from mu4e-view-save-attachments
10    (interactive "P")
11    (cl-assert (and (eq major-mode 'mu4e-view-mode)
12                    (derived-mode-p 'gnus-article-mode)))
13    (let* ((msg (mu4e-message-at-point))
14           (id (+cleanse-subject (mu4e-message-field msg :subject)))
15           (attachdir (expand-file-name id mu4e-attachment-dir))
16           (parts (mu4e-view-gather-mime-parts))
17           (handles '())
18           (files '())
19           (dir)
20           (mkdir attachdir t)
21           (dolist (part parts)
22             (let ((fname (or (cdr (assoc 'filename (assoc "attachment" (cdr part))))
23                              (seq-find #'stringp
24                                          (mapcar (lambda (item) (cdr (assoc 'name item)))
25                                                  (seq-filter 'listp (cdr part)))))))
26               (when fname
27                 (push `(,fname . ,(cdr part)) handles)
28                 (push fname files))))
29           (if files
30             (progn
31               (setq dir
32                    (if arg (read-directory-name "Save to directory: ")
33                     attachdir))
34               (cl-loop for (f . h) in handles
35                         when (member f files)
36                         do (mm-save-part-to-file h
37                                                  (+file-name-incremental
38                                                  (expand-file-name f dir)))))
39             (mu4e-message "No attached files found"))))
40
41  (map! :map mu4e-view-mode-map
42        :ne "P" #' +mu4e-view-save-all-attachments))

```

## 7.6 IRC

```

1 ;; TODO: Not tangled
2 (defun +fetch-my-password (&rest params)
3   (require 'auth-source)
4   (let ((match (car (apply #'auth-source-search params))))
5     (if match
6       (let ((secret (plist-get match :secret)))
7         (if (functionp secret)
8             (funcall secret)
9             secret))
10      (error "Password not found for %S" params))))
11
12 (defun +my-nickserver-password (server)
13   (+fetch-my-password :user "abougouffa" :host "irc.libera.chat"))
14
15 (set-irc-server! "irc.libera.chat"
16   '(:tls t
17     :port 6697
18     :nick "abougouffa"
19     :sasl-password +my-nickserver-password
20     :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

```

1 ;; Not sure if it is required!
2 (after! mpc
3   (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:

```

1 (defun +mpd-daemon-start ()
2   "Start MPD, connects to it and syncs the metadata cache."
3   (interactive)
4   (let ((mpd-daemon-running-p (+mpd-daemon-running-p)))
5     (unless mpd-daemon-running-p
6       ;; Start the daemon if it is not already running.
7       (setq mpd-daemon-running-p (+systemd-start "mpd")))
8     (cond ((+mpd-daemon-running-p)
9            (+mpd-mpc-update)
10            (emms-player-mpd-connect)
11            (emms-cache-set-from-mpd-all)
12            (message "Connected to MPD!"))
13          (t
14            (warn "An error occurred when trying to start Systemd mpd.service."))))
15
16 (defun +mpd-daemon-stop ()
17   "Stops playback and kill the MPD daemon."
18   (interactive)
19   (emms-stop)
20   (+systemd-stop "mpd")
21   (message "MPD stopped!"))
22
23 (defun +mpd-daemon-running-p ()
24   "Check if the MPD service is running."
25   (+systemd-running-p "mpd"))
26
27 (defun +mpd-mpc-update ()

```

```

28 "Updates the MPD database synchronously."
29 (interactive)
30 (if (zerop (call-process "mpc" nil nil nil "update"))
31     (message "MPD database updated!")
32     (warn "An error occurred 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.

```

1 (after! emms
2   ;; EMMS basic configuration
3   (require 'emms-setup)
4
5   (when MPD-P
6     (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)))
77
78   (setq +emms-notification-icon "/usr/share/icons/Papirus/64x64/apps/enjoy-music-player.svg")
79
80   (defun +notify-via-kdialog (title msg icon)
81     "Send notification with TITLE, MSG, and ICON via `KDialog'."
82     (call-process "kdialog"
83                   nil nil nil
84                   "--title" title
85                   "--passivepopup" msg "5"
86                   "--icon" icon))
87
88   (defun +notify-via-freedesktop-notifications (title msg icon)
89     "Send notification with TITLE, MSG, and ICON via `D-Bus'."
90     (notifications-notify
91      :title title
92      :body msg
93      :app-icon icon
94      :urgency 'low))
95
96   (defun +notify-via-messages (title msg icon)
97     "Send notification with TITLE, MSG to message. ICON is ignored."
98     (message "%s %s" title msg))
99
100   (add-hook 'emms-player-started-hook
101     (lambda () (funcall +emms-notifier-function
102                          "EMMS is now playing:"
103                          (emms-track-description (emms-playlist-current-selected-track))
104                          +emms-notification-icon))))

```

### 7.7.3 EMPV

```

1 (package! empv
2   :recipe (:host github
3           :repo "isamert/empv.el"))

```

```

1 (use-package! empv
2   :when MPV-P
3   :init
4   (map! :leader :prefix ("l m")
5         (:prefix ("v" . "empv")
6           :desc "Play" "p" #'empv-play
7           :desc "Seach Youtube" "y" #'consult-empv-youtube
8           :desc "Play radio" "r" #'empv-play-radio
9           :desc "Save current playlist" "s" #'empv-save-playlist-to-file))
10  :config

```

```

11 ;; See https://docs.invidious.io/instances/
12 (setq empv-invidious-instance "https://invidious.projectsegfau.lt/api/v1"
13     empv-audio-dir "~/Music"
14     empv-video-dir "~/Videos"
15     empv-max-directory-search-depth 6
16     empv-radio-log-file (expand-file-name "logged-radio-songs.org" org-directory)
17     empv-audio-file-extensions '("webm" "mp3" "ogg" "wav" "m4a" "flac" "aac" "opus")
18     ;; Links from https://www.radio-browser.info
19     empv-radio-channels
20     '(("El-Bahdja FM" . "http://webradio.tda.dz:8001/ElBahdja_64K.mp3")
21       ("El-Chaabia" . "https://radio-dzair.net/proxy/chaabia?mp=/stream")
22       ("Quran Radio" . "http://stream.radiojar.com/0tpy1h0kxtzuv")
23       ("Algeria International" . "https://webradio.tda.dz/Internationale_64K.mp3")
24       ("JOW Radio" . "https://str0.creacast.com/jowradio")
25       ("Europe1" . "http://ais-live.cloud-services.paris:8000/europe1.mp3")
26       ("France Iter" . "http://direct.franceinter.fr/live/franceinter-hifi.aac")
27       ("France Info" . "http://direct.franceinfo.fr/live/franceinfo-midfi.mp3")
28       ("France Culture" . "http://icecast.radiofrance.fr/franceculture-hifi.aac")
29       ("France Musique" . "http://icecast.radiofrance.fr/francemusique-hifi.aac")
30       ("FIP" . "http://icecast.radiofrance.fr/fip-hifi.aac")
31       ("Beur FM" . "http://broadcast.infomaniak.ch/beurfm-high.aac")
32       ("Skyrock" . "http://icecast.skyrock.net/s/natio_mp3_128k")))
33
34 (empv-playlist-loop-on)
35
36 ;; Hacky palylist management (only supports saving playlist,
37 ;; loading a playlist can be achieved using `empv-play-file')
38
39 (defun +empv--dl-playlist (playlist &optional dist)
40   (let ((default-directory
41         (or dist
42             (let ((d (expand-file-name "empv-downloads" empv-audio-dir)))
43               (unless (file-directory-p d) (mkdir d t)) d)))
44         (vids (+filter
45                'identity ;; Filter nils
46                (mapcar
47                 (lambda (item)
48                   (when-let
49                     ((vid (when (string-match
50                               (rx (seq "watch?v=" (group-n 1 (one-or-more (or alnum "_" "-"))))
51                               item)
52                               (match-string 1 item))))
53                     vid))
54                  playlist)))
55         (proc-name "empv-yt-dlp"))
56     (unless (zerop (length vids))
57       (message "Downloading %d songs to %s" (length vids) default-directory)
58       (when (get-process proc-name)
59         (kill-process proc-name))
60       (make-process :name proc-name
61                     :buffer (format "%s*" proc-name)
62                     :command (append
63                               (list
64                                (executable-find "yt-dlp")
65                                "--no-abort-on-error"
66                                "--no-colors"
67                                "--extract-audio"
68                                "--no-progress"
69                                "-f" "bestaudio")
70                               vids)
71                     :sentinel (lambda (prc event)
72                                (when (string= event "finished\n")
73                                  (message "Finished downloading playlist files!"))))))))
74
75 (defun +empv-download-playtlist-files (&optional path)
76   (interactive "DSave download playlist files to: ")
77   (empv--playlist-apply #' +empv--dl-playlist path))

```

### 7.7.4 Keybindings

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

```

1 (map! :leader :prefix ("l" . "custom")
2   (:when (modulep! :app emms)
3     :prefix ("m" . "media")
4     :desc "Playlist go"           "g" #'emms-playlist-mode-go
5     :desc "Add playlist"         "D" #'emms-add-playlist
6     :desc "Toggle random playlist" "r" #'emms-toggle-random-playlist
7     :desc "Add directory"        "d" #'emms-add-directory
8     :desc "Add file"             "f" #'emms-add-file
9     :desc "Smart browse"         "b" #'emms-smart-browse
10    :desc "Play/Pause"           "p" #'emms-pause
11    :desc "Start"                 "S" #'emms-start
12    :desc "Stop"                  "s" #'emms-stop))

```

Then we add MPD related keybindings if MPD is used.

```

1 (map! :leader :prefix ("l m")
2   (:when (and (modulep! :app emms) MPD-P)
3     :prefix ("m" . "mpd/mpc")
4     :desc "Start daemon"          "s" #'+mpd-daemon-start
5     :desc "Stop daemon"           "k" #'+mpd-daemon-stop
6     :desc "EMMS player (MPD update)" "R" #'emms-player-mpd-update-all-reset-cache
7     :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.

```

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

```

```

1 (use-package! emms-mode-line-cycle
2   :after emms
3   :config
4   (setq emms-mode-line-cycle-max-width 15
5         emms-mode-line-cycle-additional-space-num 4
6         emms-mode-line-cycle-any-width-p nil
7         emms-mode-line-cycle-velocity 4)
8
9   ;; Some music files do not have metadata, by default, the track title
10  ;; will be the full file path, so, if I detect what seems to be an absolute
11  ;; path, I trim the directory part and get only the file name.
12  (setq emms-mode-line-cycle-current-title-function
13    (lambda ()
14      (let ((name (emms-track-description (emms-playlist-current-selected-track))))
15        (if (file-name-absolute-p name) (file-name-base name) name))))
16
17  ;; Mode line formatting settings
18  ;; This format complements the 'emms-mode-line-format' one.
19  (setq emms-mode-line-format " %s " ;;
20    ;; To hide the playing time without stopping the cycling.
21    emms-playing-time-display-format "")
22
23  (defun +emms-mode-line-toggle-format-hook ()
24    "Toggle the 'emms-mode-line-format' string, when playing or paused."
25    (setq emms-mode-line-format (concat " " (if emms-player-paused-p " " " ") " %s ")))
26  ;; Force a sync to get the right song name over MPD in mode line

```

```

27 (when MPD-P (emms-player-mpd-sync-from-mpd))
28 ;; Trigger a forced update of mode line (useful when pausing)
29 (emms-mode-line-alter-mode-line))
30
31 ;; Hook the function to the 'emms-player-paused-hook'
32 (add-hook 'emms-player-paused-hook '+emms-mode-line-toggle-format-hook)
33
34 (emms-mode-line-cycle 1))

```

## 7.8 Maxima

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

### 7.8.1 Maxima

The `emacs-mirror/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`.

```

1 (package! maxima
2   :recipe (:host github
3           :repo "emacs-mirror/maxima"
4           :files (:defaults
5                   "keywords"
6                   "company-maxima.el"
7                   "poly-maxima.el")))

```

```

1 (use-package! maxima
2   :when MAXIMA-P
3   :commands (maxima-mode maxima-inferior-mode maxima)
4   :init
5   (require 'straight) ;; to use `straight-build-dir' and `straight-base-dir'
6   (setq maxima-font-lock-keywords-directory ;; a workaround to undo the straight workaround!
7         (expand-file-name (format "straight/%s/maxima/keywords" straight-build-dir straight-base-dir)))
8
9   ;; The `maxima-hook-function' setup `company-maxima'.
10  (add-hook 'maxima-mode-hook #'maxima-hook-function)
11  (add-hook 'maxima-inferior-mode-hook #'maxima-hook-function)
12  (add-to-list 'auto-mode-alist '("\\.ma[cs]\\$" . maxima-mode)))

```

### 7.8.2 IMaxima

For the `imaxima` (Maxima with image support), the `emacs-sattic/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`.

```

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

```

```

1 (use-package! imaxima
2   :when MAXIMA-P
3   :commands (imaxima imath-mode)
4   :init

```

```

5 (setq imaxima-use-maxima-mode-flag nil ;; otherwise, it don't render equations with LaTeX.
6   imaxima-scale-factor 2.0)
7
8 ;; Hook the `maxima-inferior-mode' to get Company completion.
9 (add-hook 'imaxima-startup-hook #'maxima-inferior-mode))

```

## 7.9 FriCAS

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

```

1 (use-package! fricas
2   :when FRICAS-P
3   :load-path FRICAS-DIR
4   :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.

```

1 (use-package! websocket
2   :after org-roam-ui)
3
4 (use-package! org-roam-ui
5   :commands org-roam-ui-open
6   :config (setq org-roam-ui-sync-theme t
7                 org-roam-ui-follow t
8                 org-roam-ui-update-on-save t
9                 org-roam-ui-open-on-start t))

```

### 7.10.1 Basic settings

```

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

```

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

```

1 (package! org-roam
2   :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

```

1 (defadvice! doom-modeline--buffer-file-name-roam-aware-a (orig-fun)
2   :around #'doom-modeline-buffer-file-name ; takes no args
3   (if (s-contains-p org-roam-directory (or buffer-file-name ""))
4       (replace-regexp-in-string
5         "\\(?:~|\\.*/|\\.*/|\\[0-9\\]\\{4\\}\\|\\[0-9\\]\\{2\\}\\|\\[0-9\\]\\{2\\}\\}\\[0-9\\]*-"
6         "(\\1-\\2-\\3) "
7         (subst-char-in-string ?_ ? buffer-file-name))
8       (funcall orig-fun)))

```

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

```
; ; 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
"\\\\)"
```

```
(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 ?\,)))
```

```

14      (n (count-matches (string separator) (point-at-bol) (point-at-eol)))
15      (colors (cl-loop for i from 0 to 1.0 by (/ 2.0 n)
16                collect (apply #'color-rgb-to-hex
17                              (color-hsl-to-rgb i 0.3 0.5))))))
18      (cl-loop for i from 2 to n by 2
19                for c in colors
20                for r = (format "~\\([~%c\\n]+%c\\)\\{\\%d\\}" separator separator i)
21                do (font-lock-add-keywords nil `((,r (1 '(face (:foreground ,c))))))))
22
23 ;; provide CSV mode setup
24 ;; (add-hook 'csv-mode-hook (lambda () (+csv-rainbow)))

```

## 8.2 Vimrc

```

1 (package! vimrc-mode
2   :recipe (:host github
3           :repo "mcandre/vimrc-mode"))

```

```

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

```

## 8.3 ESS

View data frames better with

```

1 (package! ess-view)

```

## 8.4 Python IDE

```

1 (package! elpy)

```

```

1 (use-package! elpy
2   :hook ((elpy-mode . flycheck-mode)
3         (elpy-mode . (lambda ()
4                       (set (make-local-variable 'company-backends)
5                           '((elpy-company-backend :with company-yasnippet))))))
6   :config
7   ;;:init
8   (elpy-enable))

```

## 8.5 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.

```

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

```

```

1 (defun +octave-eval-last-sexp ()
2   "Evaluate Octave sexp before point and print value into current buffer."
3   (interactive)
4   (inferior-octave t)
5   (let ((print-escape-newlines nil)
6         (opoint (point)))
7     (prin1
8      (save-excursion
9        (forward-sexp -1)
10        (inferior-octave-send-list-and-digest
11         (list (concat (buffer-substring-no-properties (point) opoint)
12                      "\n"))))
13      (mapconcat 'identity inferior-octave-output-list "\n")))))
14
15 (defun +eros-octave-eval-last-sexp ()
16   "Wrapper for `+octave-eval-last-sexp' that overlays results."
17   (interactive)
18   (eros--eval-overlay
19    (octave-eval-last-sexp)
20    (point)))
21
22 (map! :localleader
23      :map (octave-mode-map)
24      (:prefix ("e" . "eval")
25              :desc "Eval and print last sexp" "e" #' +eros-octave-eval-last-sexp))

```

## 8.6 ROS

### 8.6.1 Extensions

Add ROS specific file formats:

```

1 (add-to-list 'auto-mode-alist '("\\.rviz\\$" . conf-unix-mode))
2 (add-to-list 'auto-mode-alist '("\\.urdf\\$" . xml-mode))
3 (add-to-list 'auto-mode-alist '("\\.xacro\\$" . xml-mode))
4 (add-to-list 'auto-mode-alist '("\\.launch\\$" . xml-mode))
5
6 ;; Use gdb-script-mode for msg and srv files
7 (add-to-list 'auto-mode-alist '("\\.msg\\$" . gdb-script-mode))
8 (add-to-list 'auto-mode-alist '("\\.srv\\$" . gdb-script-mode))
9 (add-to-list 'auto-mode-alist '("\\.action\\$" . gdb-script-mode))

```

### 8.6.2 ROS bags

Mode to view ROS .bag files. Taken from code-iai/ros\_emacs\_utils.

```

1 (when ROSBAG-P
2   (define-derived-mode rosbag-view-mode
3     fundamental-mode "Rosbag view mode"
4     "Major mode for viewing ROS bag files."
5     (let ((f (buffer-file-name)))
6       (let ((buffer-read-only nil))
7         (erase-buffer)
8         (message "Calling rosbag info")
9         (call-process "rosbag" nil (current-buffer) nil
10                      "info" f)
11         (set-buffer-modified-p nil))
12       (view-mode)
13       (set-visited-file-name nil t)))
14
15 ;; rosbag view mode
16 (add-to-list 'auto-mode-alist '("\\.bag$" . rosbag-view-mode)))

```



### 8.6.3 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!).

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

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

```
1 (use-package! ros
2   :init
3   (map! :leader
4         :prefix ("l" . "custom")
5         :desc "Hydra ROS" "r" #'hydra-ros-main/body)
6   :commands (hydra-ros-main/body ros-set-workspace)
7   :config
8   (setq ros-workspaces
9         (list (ros-dump-workspace
10               :tramp-prefix (format "/docker:%s@%s:" "ros" "ros-machine")
11               :workspace "~/ros_ws"
12               :extends '("/opt/ros/noetic/"))
13               (ros-dump-workspace
14                 :tramp-prefix (format "/ssh:%s@%s:" "swd_sk" "172.16.96.42")
15                 :workspace "~/ros_ws"
16                 :extends '("/opt/ros/noetic/"))
17               (ros-dump-workspace
18                 :tramp-prefix (format "/ssh:%s@%s:" "swd_sk" "172.16.96.42")
19                 :workspace "~/ros2_ws"
20                 :extends '("/opt/ros/foxy/"))))))
```

## 8.7 Scheme

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

## 8.8 Embedded systems

### 8.8.1 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

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

```

1 (use-package! embed
2   :commands (embed-openocd-start
3               embed-openocd-stop
4               embed-openocd-gdb
5               embed-openocd-flash)
6
7   :init
8   (map! :leader :prefix ("l" . "custom")
9         (:when (modulep! :tools debugger +lsp)
10              :prefix ("e" . "embedded")
11              :desc "Start OpenOCD"      "o" #'embed-openocd-start
12              :desc "Stop OpenOCD"      "O" #'embed-openocd-stop
13              :desc "OpenOCD GDB"       "g" #'embed-openocd-gdb
14              :desc "OpenOCD flash"     "f" #'embed-openocd-flash)))

```

### 8.8.2 Arduino

```

1 (package! arduino-mode
2   :recipe (:host github
3           :repo "bookest/arduino-mode"))

```

### 8.8.3 Bitbake (Yocto)

Add support for Yocto Project files.

```

1 (package! bitbake-modes
2   :recipe (:host bitbucket
3           :repo "olaniilsson/bitbake-modes"))

```

```

1 (use-package! bitbake-modes
2   :commands (wks-mode
3               mmm-mode
4               bb-sh-mode
5               bb-scc-mode
6               bitbake-mode
7               conf-bitbake-mode
8               bitbake-task-log-mode))

```

## 8.9 Git & VC

### 8.9.1 Magit

```

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

```

```

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

```

Granular diff-highlights for *all* hunks

```

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

```

## Gravatars

```

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

```

## WIP Company for commit messages

```

1 (use-package! company-conventional-commits
2   :after (magit company)
3   :config
4   (add-hook
5     'git-commit-setup-hook
6     (lambda ()
7       (add-to-list 'company-backends 'company-conventional-commits))))

```

```

1 (package! magit-pretty-graph
2   :recipe (:host github
3           :repo "georgek/magit-pretty-graph"))

```

## Pretty graph

```

1 (use-package! magit-pretty-graph
2   :after magit
3   :init
4   (setq magit-pg-command
5     (concat "git --no-pager log"
6             " --topo-order --decorate=full"
7             " --pretty=format: \"%H%x00%P%x00%an%x00%ar%x00%s%x00%d\" "
8             " -n 2000")) ;; Increase the default 100 limit
9
10  (map! :localleader
11        :map (magit-mode-map)
12        :desc "Magit pretty graph" "p" (cmd! (magit-pg-repo (magit-toplevel)))))

```

### 8.9.2 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}"

```

```

1 (package! repo)

```

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

### 8.9.3 Blamer

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

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

```

1 (use-package! blamer
2   :commands (blamer-mode)
3   ;; :hook ((prog-mode . blamer-mode))
4   :custom
5     (blamer-idle-time 0.3)
6     (blamer-min-offset 60)
7     (blamer-prettify-time-p t)
8     (blamer-entire-formatter " %s")
9     (blamer-author-formatter " %s ")
10    (blamer-datetime-formatter "[%s], ")
11    (blamer-commit-formatter "\"%s\"")
12    :custom-face
13      (blamer-face ((t :foreground "#7a88cf"
14                        :background nil
15                        :height 125
16                        :italic t))))
17  :config
18  (when (modulep! :ui zen) ;; Disable in zen (writeroom) mode
19    (add-hook 'writeroom-mode-enable-hook
20              (when (bound-and-true-p blamer-mode)
21                (setq +blamer-mode--was-active-p t)
22                  (blamer-mode -1))))
23    (add-hook 'writeroom-mode-disable-hook
24              (when (bound-and-true-p +blamer-mode--was-active-p)
25                (blamer-mode 1)))))

```

## 8.10 Assembly

Add some packages for better assembly coding.

```
1 (package! nasm-mode)
2 (package! haxor-mode)
3 (package! mips-mode)
4 (package! riscv-mode)
5 (package! x86-lookup)
```

```
1 (use-package! nasm-mode
2   :mode "\\. [n]*\\(asm\\|s\\)\\'")
3
4 ;; Get Haxor VM from https://github.com/krzysztof-magosa/haxor
5 (use-package! haxor-mode
6   :mode "\\.hax\\'")
7
8 (use-package! mips-mode
9   :mode "\\.mips\\'")
10
11 (use-package! riscv-mode
12   :mode "\\.riscv\\'")
```

```

13
14 (use-package! x86-lookup
15   :commands (x86-lookup)
16   :config
17   (when (modulep! :tools pdf)
18     (setq x86-lookup-browse-pdf-function 'x86-lookup-browse-pdf-pdf-tools))
19   ;; Get manual from https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
20   (setq x86-lookup-pdf (expand-file-name "x86-lookup/325383-sdm-vol-2abcd.pdf" doom-data-dir)))

```

## 8.11 Disaster

```

1 (package! disaster)

```

```

1 (use-package! disaster
2   :commands (disaster)
3   :init
4   (setq disaster-assembly-mode 'nasm-mode)
5
6   (map! :localleader
7         :map (c++-mode-map c-mode-map fortran-mode)
8         :desc "Disaster" "d" #'disaster))

```

## 8.12 Devdocs

```

1 (package! devdocs
2   :recipe (:host github
3           :repo "astoff/devdocs.el"
4           :files ("*.el")))

```

```

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

```

## 8.13 Systemd

For editing systemd unit files.

```

1 (package! systemd)
2 (package! journalctl-mode)

```

```

1 (use-package! journalctl-mode
2   :commands (journalctl
3             journalctl-boot
4             journalctl-unit
5             journalctl-user-unit)
6   :init
7   (map! :map journalctl-mode-map
8         :nv "J" #'journalctl-next-chunk
9         :nv "K" #'journalctl-previous-chunk))

```

## 8.14 PKGBUILD

```
1 (package! pkgbuild-mode)
```

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

## 8.15 Franca IDL

Add support for *Franca Interface Definition Language*.

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

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

## 8.16 L<sup>A</sup>T<sub>E</sub>X

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

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

## 8.17 Flycheck + Projectile

WIP: Not working atm!

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

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

## 8.18 Graphviz

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

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

```

1 (use-package! graphviz-dot-mode
2   :commands graphviz-dot-mode
3   :mode ("\\.dot\\'" "\\.gv\\'")
4   :init
5   (after! org
6     (setcdr (assoc "dot" org-src-lang-modes) 'graphviz-dot))
7
8   :config
9   (require 'company-graphviz-dot))

```

## 8.19 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`.

```

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

```

## 8.20 Mermaid

```

1 (package! mermaid-mode)
2
3 (package! ob-mermaid
4   :recipe (:host github
5           :repo "arnm/ob-mermaid"))

```

```

1 (use-package! mermaid-mode
2   :commands mermaid-mode
3   :mode "\\.mmd\\'")
4
5 (use-package! ob-mermaid
6   :after org
7   :init
8   (after! org
9     (add-to-list 'org-babel-load-languages '(mermaid . t))))

```

## 8.21 The V Programming Language

```

1 (package! v-mode)

```

```

1 (use-package! v-mode
2   :mode ("\\(\\.v?v\\|\\.vsh\\)$" . 'v-mode)
3   :config
4   (map! :localleader
5     :map (v-mode-map)
6     :desc "v-format-buffer" "f" #'v-format-buffer
7     :desc "v-menu" "m" #'v-menu))

```

## 8.22 Inspector

```

1 (package! inspector
2   :recipe (:host github
3           :repo "mmontone/emacs-inspector"))

```

```

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

```

## 9 Office

### 9.1 Org additional packages

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

```

1 (unpin! org-roam) ;; To avoid problems with org-roam-ui
2 (package! websocket)
3 (package! org-roam-ui)
4 (package! org-wild-notifier)
5 (package! org-fragtog)
6 (package! org-appear)
7 (package! org-super-agenda)
8 (package! doct)
9
10 (package! citar-org-roam
11   :recipe (:host github
12           :repo "emacs-citar/citar-org-roam"))
13
14 (package! org-menu
15   :recipe (:host github
16           :repo "sheijk/org-menu"))
17
18 (package! caldav
19   :recipe (:host github
20           :repo "dengste/org-caldav"))
21
22 (package! org-ol-tree
23   :recipe (:host github
24           :repo "Townk/org-ol-tree"))
25
26 (package! org-modern
27   :recipe (:host github
28           :repo "minad/org-modern"))
29
30 (package! org-bib
31   :recipe (:host github
32           :repo "rougier/org-bib-mode"))
33
34 (package! academic-phrases
35   :recipe (:host github
36           :repo "nashamri/academic-phrases"))
37
38 (package! phscroll
39   :recipe (:host github
40           :repo "misohena/phscroll"))

```

### 9.2 Org mode

#### 9.2.1 Intro

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



```

1 (after! org
2   <<org-conf>>
3 )

```

### 9.2.2 Behavior

#### Tweaking defaults

```

1 (setq org-directory "~/Dropbox/Org/" ; let's put files here
2   org-use-property-inheritance t ; it's convenient to have properties inherited
3   org-log-done 'time ; having the time an item is done sounds convenient
4   org-list-allow-alphabetical t ; have a. A. a) A) list bullets
5   org-export-in-background nil ; run export processes in external emacs process
6   org-export-async-debug t
7   org-tags-column 0
8   org-catch-invisible-edits 'smart ; try not to accidentally do weird stuff in invisible regions
9   org-export-with-sub-superscripts '{}' ; don't treat lone _ / ^ as sub/superscripts, require _{} / ^{}
10  org-pretty-entities-include-sub-superscripts nil
11  org-auto-align-tags nil
12  org-special-ctrl-a/e t
13  org-startup-indented t ; Enable 'org-indent-mode' by default, override with '+#startup: noindent' for big
↪ files
14  org-insert-heading-respect-content t)

```

#### Org basics

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

```

1 (setq org-babel-default-header-args
2   '(:session . "none")
3     (:results . "replace")
4     (:exports . "code")
5     (:cache . "no")
6     (:noweb . "no")
7     (:hlines . "no")
8     (:tangle . "no")
9     (: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!

```

1 ;; stolen from https://github.com/yohan-pereira/.emacs#babel-config
2 (defun +org-confirm-babel-evaluate (lang body)
3   (not (string= lang "scheme"))) ; Don't ask for scheme
4
5 (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.

```

1 (map! :map evil-org-mode-map
2   :after evil-org
3   :n "g <up>" #'org-backward-heading-same-level
4   :n "g <down>" #'org-forward-heading-same-level
5   :n "g <left>" #'org-up-element
6   :n "g <right>" #'org-down-element)

```

```

1 (setq org-todo-keywords
2   '((sequence "IDEA(i)" "TODO(t)" "NEXT(n)" "PROJ(p)" "STRT(s)" "WAIT(w)" "HOLD(h)" "|" "DONE(d)" "KILL(k)")
3     (sequence "[ ](T)" "[-](S)" "|" "[X](D)")
4     (sequence "|" "OKAY(o)" "YES(y)" "NO(n)")))
5
6 (setq org-todo-keyword-faces
7   '(("IDEA" . (:foreground "goldenrod" :weight bold))
8     ("NEXT" . (:foreground "IndianRed1" :weight bold))
9     ("STRT" . (:foreground "OrangeRed" :weight bold))
10    ("WAIT" . (:foreground "coral" :weight bold))
11    ("KILL" . (:foreground "DarkGreen" :weight bold))
12    ("PROJ" . (:foreground "LimeGreen" :weight bold))
13    ("HOLD" . (:foreground "orange" :weight bold))))
14
15 (defun +log-todo-next-creation-date (&rest ignore)
16   "Log NEXT creation time in the property drawer under the key 'ACTIVATED'"
17   (when (and (string= (org-get-todo-state) "NEXT")
18             (not (org-entry-get nil "ACTIVATED"))))
19     (org-entry-put nil "ACTIVATED" (format-time-string "[%Y-%m-%d]"))))
20
21 (add-hook 'org-after-todo-state-change-hook #' +log-todo-next-creation-date)

```

## TODOs

```

1 (setq org-tag-persistent-alist
2   '(:startgroup . nil)
3     ("home" . ?h)
4     ("research" . ?r)
5     ("work" . ?w)
6     (:endgroup . nil)
7     (:startgroup . nil)
8     ("tool" . ?o)
9     ("dev" . ?d)
10    ("report" . ?p)
11    (:endgroup . nil)
12    (:startgroup . nil)
13    ("easy" . ?e)
14    ("medium" . ?m)
15    ("hard" . ?a)
16    (:endgroup . nil)
17    ("urgent" . ?u)
18    ("key" . ?k)
19    ("bonus" . ?b)
20    ("ignore" . ?i)
21    ("noexport" . ?x)))
22
23 (setq org-tag-faces
24   '(("home" . (:foreground "goldenrod" :weight bold))
25     ("research" . (:foreground "goldenrod" :weight bold))
26     ("work" . (:foreground "goldenrod" :weight bold))
27     ("tool" . (:foreground "IndianRed1" :weight bold))
28     ("dev" . (:foreground "IndianRed1" :weight bold))
29     ("report" . (:foreground "IndianRed1" :weight bold))
30     ("urgent" . (:foreground "red" :weight bold))
31     ("key" . (:foreground "red" :weight bold))
32     ("easy" . (:foreground "green4" :weight bold))
33     ("medium" . (:foreground "orange" :weight bold))
34     ("hard" . (:foreground "red" :weight bold))
35     ("bonus" . (:foreground "goldenrod" :weight bold))
36     ("ignore" . (:foreground "Gray" :weight bold))
37     ("noexport" . (:foreground "LimeGreen" :weight bold))))
38

```

## Tags

**Agenda** Set files for org-agenda

```
1 (setq org-agenda-files
2   (list (expand-file-name "inbox.org" org-directory)
3         (expand-file-name "agenda.org" org-directory)
4         (expand-file-name "gcal-agenda.org" org-directory)
5         (expand-file-name "notes.org" org-directory)
6         (expand-file-name "projects.org" org-directory)
7         (expand-file-name "archive.org" org-directory)))
```

Apply some styling on the standard agenda:

```
1 ;; Agenda styling
2 (setq org-agenda-block-separator ?)
3 org-agenda-time-grid
4 '(("daily today require-timed)
5   (800 1000 1200 1400 1600 1800 2000)
6   " " " ")
7 org-agenda-current-time-string
8 " now ")
```

**Super agenda**    Configure org-super-agenda

```

1 (use-package! org-super-agenda
2   :defer t
3   :config
4   (org-super-agenda-mode)
5   :init
6   (setq org-agenda-skip-scheduled-if-done t
7         org-agenda-skip-deadline-if-done t
8         org-agenda-include-deadlines t
9         org-agenda-block-separator nil
10        org-agenda-tags-column 100 ;; from testing this seems to be a good value
11        org-agenda-compact-blocks t)
12
13   (setq org-agenda-custom-commands
14         '(("o" "Overview"
15           ((agenda "" ((org-agenda-span 'day)
16                        (org-super-agenda-groups
17                          '((:name "Today"
18                             :time-grid t
19                             :date today
20                             :todo "TODAY"
21                             :scheduled today
22                             :order 1)))))
23           (alltodo "" ((org-agenda-overriding-header "")
24                        (org-super-agenda-groups
25                          '((:name "Next to do" :todo "NEXT" :order 1)
26                            (:name "Important" :tag "Important" :priority "A" :order 6)
27                            (:name "Due Today" :deadline today :order 2)
28                            (:name "Due Soon" :deadline future :order 8)
29                            (:name "Overdue" :deadline past :face error :order 7)
30                            (:name "Assignments" :tag "Assignment" :order 10)
31                            (:name "Issues" :tag "Issue" :order 12)
32                            (:name "Emacs" :tag "Emacs" :order 13)
33                            (:name "Projects" :tag "Project" :order 14)
34                            (:name "Research" :tag "Research" :order 15)
35                            (:name "To read" :tag "Read" :order 30)
36                            (:name "Waiting" :todo "WAIT" :order 20)
37                            (:name "University" :tag "Univ" :order 32)
38                            (:name "Trivial" :priority<= "E" :tag ("Trivial" "Unimportant") :todo ("SOMEDAY")
39                               :order 90)
40                            (: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")))
```

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

**TODO CalDAV** Need to be configured, see the GitHub repo.

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

## Capture Set capture files

```
1 (setq +org-capture-emails-file (expand-file-name "inbox.org" org-directory)
2   +org-capture-todo-file (expand-file-name "inbox.org" org-directory)
3   +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.

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

```
1 (after! org-capture
2   <<prettify-capture>>)
3
4 (defun +doct-icon-declaration-to-icon (declaration)
5   "Convert :icon declaration to icon"
6   (let ((name (pop declaration))
7         (set (intern (concat "all-the-icons-" (plist-get declaration :set))))
8         (face (intern (concat "all-the-icons-" (plist-get declaration :color))))
9         (v-adjust (or (plist-get declaration :v-adjust) 0.01)))
10    (apply set `(:name ,name :face ,face :v-adjust ,v-adjust)))
11
12 (defun +doct-iconify-capture-templates (groups)
13   "Add declaration's :icon to each template group in GROUPS."
14   (let ((templates (doct-flatten-lists-in groups)))
15     (setq doct-templates
16           (mapcar (lambda (template)
17                     (when-let* ((props (nthcdr (if (= (length template) 4) 2 5) template))
18                               (spec (plist-get (plist-get props :doct) :icon)))
19                       (setf (nth 1 template) (concat (+doct-icon-declaration-to-icon spec)
20                                                         "\t"
21                                                         (nth 1 template))))
22                     template)
23           templates))))
24
25 (setq doct-after-conversion-functions '(+doct-iconify-capture-templates))
26
27 (defun set-org-capture-templates ()
28   (setq org-capture-templates
29         (doct `(("Personal todo" :keys "t"
30                  :icon ("checklist" :set "octicon" :color "green"))
```

```

31      :file +org-capture-todo-file
32      :prepend t
33      :headline "Inbox"
34      :type entry
35      :template ("* TODO %?"
36                "%i %a"))
37      ("Personal note" :keys "n"
38       :icon ("sticky-note-o" :set "faicon" :color "green")
39       :file +org-capture-todo-file
40       :prepend t
41       :headline "Inbox"
42       :type entry
43       :template ("* %?"
44                 "%i %a"))
45      ("Email" :keys "e"
46       :icon ("envelope" :set "faicon" :color "blue")
47       :file +org-capture-todo-file
48       :prepend t
49       :headline "Inbox"
50       :type entry
51       :template ("* TODO %^{type|reply to|contact} %\\3 %? :email:"
52                 "Send an email %^{urgancy|soon|ASAP|anon|at some point|eventually} to
↪ %^{recipiant}"
53                 "about %^{topic}"
54                 "%U %i %a"))
55      ("Interesting" :keys "i"
56       :icon ("eye" :set "faicon" :color "lcyan")
57       :file +org-capture-todo-file
58       :prepend t
59       :headline "Interesting"
60       :type entry
61       :template ("* [ ] %^{desc}%? :%{i-type}:"
62                 "%i %a")
63       :children ((("Webpage" :keys "w"
64                    :icon ("globe" :set "faicon" :color "green")
65                    :desc "%(org-cliplink-capture) "
66                    :i-type "read:web")
67                   ("Article" :keys "a"
68                    :icon ("file-text" :set "octicon" :color "yellow")
69                    :desc ""
70                    :i-type "read:reaserch")
71                   ("Information" :keys "i"
72                    :icon ("info-circle" :set "faicon" :color "blue")
73                    :desc ""
74                    :i-type "read:info")
75                   ("Idea" :keys "I"
76                    :icon ("bubble_chart" :set "material" :color "silver")
77                    :desc ""
78                    :i-type "idea"))))
79      ("Tasks" :keys "k"
80       :icon ("inbox" :set "octicon" :color "yellow")
81       :file +org-capture-todo-file
82       :prepend t
83       :headline "Tasks"
84       :type entry
85       :template ("* TODO %? %^G%{extra}"
86                 "%i %a")
87       :children ((("General Task" :keys "k"
88                    :icon ("inbox" :set "octicon" :color "yellow")
89                    :extra "")
90                   ("Task with deadline" :keys "d"
91                    :icon ("timer" :set "material" :color "orange" :v-adjust -0.1)
92                    :extra "\nDEADLINE: %^{Deadline:}t")
93                   ("Scheduled Task" :keys "s"
94                    :icon ("calendar" :set "octicon" :color "orange")
95                    :extra "\nSCHEDULED: %^{Start time:}t"))))
96      ("Project" :keys "p"
97       :icon ("repo" :set "octicon" :color "silver")

```

```

100      :prepend t
101      :type entry
102      :headline "Inbox"
103      :template ("* %{{time-or-todo}} %?"
104                "%i"
105                "%a")
106
107      :file ""
108      :custom (:time-or-todo "")
109      :children ((("Project-local todo" :keys "t"
110                  :icon ("checklist" :set "octicon" :color "green")
111                  :time-or-todo "TODO"
112                  :file +org-capture-project-todo-file)
113                  ("Project-local note" :keys "n"
114                  :icon ("sticky-note" :set "faicon" :color "yellow")
115                  :time-or-todo "%U"
116                  :file +org-capture-project-notes-file)
117                  ("Project-local changelog" :keys "c"
118                  :icon ("list" :set "faicon" :color "blue")
119                  :time-or-todo "%U"
120                  :heading "Unreleased"
121                  :file +org-capture-project-changelog-file)))
122      ("Centralised project templates"
123       :keys "o"
124       :type entry
125       :prepend t
126       :template ("* %{{time-or-todo}} %?"
127                 "%i"
128                 "%a")
129       :children ((("Project todo"
130                   :keys "t"
131                   :prepend nil
132                   :time-or-todo "TODO"
133                   :heading "Tasks"
134                   :file +org-capture-central-project-todo-file)
135                   ("Project note"
136                   :keys "n"
137                   :time-or-todo "%U"
138                   :heading "Notes"
139                   :file +org-capture-central-project-notes-file)
140                   ("Project changelog"
141                   :keys "c"
142                   :time-or-todo "%U"
143                   :heading "Unreleased"
144                   :file +org-capture-central-project-changelog-file))))))
145
146 (set-org-capture-templates)
147 (unless (display-graphic-p)
148   (add-hook 'server-after-make-frame-hook
149     (defun org-capture-reinitialise-hook ()
150       (when (display-graphic-p)
151         (set-org-capture-templates)
152         (remove-hook 'server-after-make-frame-hook
153           #'org-capture-reinitialise-hook))))))

```

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

```

1 (defun org-capture-select-template-prettier (&optional keys)
2   "Select a capture template, in a prettier way than default
3   Lisp programs can force the template by setting KEYS to a string."
4   (let ((org-capture-templates
5         (or (org-contextualize-keys
6              (org-capture-upgrade-templates org-capture-templates)
7              org-capture-templates-contexts)
8             '(("t" "Task" entry (file+headline "" "Tasks")
9               "* TODO %?\n %u\n %a")))))
10     (if keys
11         (or (assoc keys org-capture-templates)
12             (error "No capture template referred to by \"%s\" keys" keys))
13         (org-mks org-capture-templates

```

```

14         "Select a capture template\n
15         "Template key: "
16         `(("q" ,(concat (all-the-icons-octicon "stop" :face 'all-the-icons-red :v-adjust 0.01)
17         ↪ "\tAbort")))))))
18 (advice-add 'org-capture-select-template :override #'org-capture-select-template-prettier)
19
20 (defun org-mks-pretty (table title &optional prompt specials)
21   "Select a member of an alist with multiple keys. Prettified.
22
23   TABLE is the alist which should contain entries where the car is a string.
24   There should be two types of entries.
25
26   1. prefix descriptions like (\"a\" \"Description\")
27     This indicates that `a' is a prefix key for multi-letter selection, and
28     that there are entries following with keys like \"ab\", \"ax\"...
29
30   2. Select-able members must have more than two elements, with the first
31     being the string of keys that lead to selecting it, and the second a
32     short description string of the item.
33
34   The command will then make a temporary buffer listing all entries
35   that can be selected with a single key, and all the single key
36   prefixes. When you press the key for a single-letter entry, it is selected.
37   When you press a prefix key, the commands (and maybe further prefixes)
38   under this key will be shown and offered for selection.
39
40   TITLE will be placed over the selection in the temporary buffer,
41   PROMPT will be used when prompting for a key. SPECIALS is an
42   alist with (\"key\" \"description\") entries. When one of these
43   is selected, only the bare key is returned."
44   (save-window-excursion
45     (let ((inhibit-quit t)
46           (buffer (org-switch-to-buffer-other-window "*Org Select*"))
47           (prompt (or prompt "Select: "))
48           (case-fold-search
49            current)
50           (unwind-protect
51            (catch 'exit
52              (while t
53                (setq-local evil-normal-state-cursor (list nil))
54                (erase-buffer)
55                (insert title "\n\n")
56                (let ((des-keys nil)
57                      (allowed-keys '("\C-g"))
58                      (tab-alternatives '("\s" "\t" "\r"))
59                      (cursor-type nil))
60                  ;; Populate allowed keys and descriptions keys
61                  ;; available with CURRENT selector.
62                  (let ((re (format "\\~%s\\(.\\)\\'"
63                                     (if current (regexp-quote current) "")))
64                      (prefix (if current (concat current " ") "")))
65                    (dolist (entry table)
66                      (pcase entry
67                        ;; Description.
68                        `((, (and key (pred (string-match re))) ,desc)
69                          (let ((k (match-string 1 key)))
70                            (push k des-keys)
71                            ;; Keys ending in tab, space or RET are equivalent.
72                            (if (member k tab-alternatives)
73                                (push "\t" allowed-keys)
74                                (push k allowed-keys))
75                            (insert (propertize prefix 'face 'font-lock-comment-face) (propertize k 'face 'bold)
76                                ↪ (propertize ">" 'face 'font-lock-comment-face) " " desc "..." "\n"))
77                          ;; Usable entry.
78                          `((, (and key (pred (string-match re))) ,desc . ,_)
79                            (let ((k (match-string 1 key)))
80                              (insert (propertize prefix 'face 'font-lock-comment-face) (propertize k 'face 'bold) "
81                                ↪ " desc "\n")
82                              (push k allowed-keys)))
83                            ( nil))))))

```

```

81      ;; Insert special entries, if any.
82      (when specials
83        (insert "                \n")
84        (pcase-dolist `((,key ,description) specials)
85          (insert (format "%s %s\n" (propertize key 'face '(bold all-the-icons-red)) description))
86          (push key allowed-keys)))
87      ;; Display UI and let user select an entry or
88      ;; a sublevel prefix.
89      (goto-char (point-min))
90      (unless (pos-visible-in-window-p (point-max))
91        (org-fit-window-to-buffer))
92      (let ((pressed (org--mks-read-key allowed-keys
93                                     prompt
94                                     (not (pos-visible-in-window-p (1- (point-max)))))))
95        (setq current (concat current pressed))
96        (cond
97          ((equal pressed "\C-g") (user-error "Abort"))
98          ;; Selection is a prefix: open a new menu.
99          ((member pressed des-keys)
100           ;; Selection matches an association: return it.
101           ((let ((entry (assoc current table)))
102              (and entry (throw 'exit entry))))
103           ;; Selection matches a special entry: return the
104           ;; selection prefix.
105           ((assoc current specials) (throw 'exit current))
106           (t (error "No entry available"))))))
107      (when buffer (kill-buffer buffer))))))
108 (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.

```

1 (setf (alist-get 'height +org-capture-frame-parameters) 15)
2 ;; (alist-get 'name +org-capture-frame-parameters) " Capture" ;; ATM hardcoded in other places, so changing
3 ↪ breaks stuff
4 (setf +org-capture-fn
5       (lambda ()
6         (interactive)
7         (set-window-parameter nil 'mode-line-format 'none)
8         (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

```

1 (defun +yas/org-src-header-p ()
2   "Determine whether `point' is within a src-block header or header-args."
3   (pcase (org-element-type (org-element-context))
4     ('src-block (< (point) ; before code part of the src-block
5                  (save-excursion (goto-char (org-element-property :begin (org-element-context)))
6                                (forward-line 1)
7                                (point))))
8     ('inline-src-block (< (point) ; before code part of the inline-src-block
9                          (save-excursion (goto-char (org-element-property :begin (org-element-context)))
10                                          (search-forward "]"")
11                                          (point))))
12     ('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.

```

1 (defun +yas/org-prompt-header-arg (arg question values)
2   "Prompt the user to set ARG header property to one of VALUES with QUESTION.
3   The default value is identified and indicated. If either default is selected,
4   or no selection is made: nil is returned."
5   (let* ((src-block-p (not (looking-back "^#\\+property: [ \\t]+header-args:.*" (line-beginning-position))))
6         (default
7          (or
8           (cdr (assoc arg
9                     (if src-block-p
7                     (nth 2 (org-babel-get-src-block-info t))
8                     (org-babel-merge-params
9                      org-babel-default-header-args
10                      (let ((lang-headers
11                           (intern (concat "org-babel-default-header-args:"
12                                           (+yas/org-src-lang))))
13                          (when (boundp lang-headers) (eval lang-headers t))))))
14                      ""))
15                      default-value)
16          (setq values (mapcar
17                      (lambda (value)
18                        (if (string-match-p (regexp-quote value) default)
19                          (setq default-value
20                                (concat value " "
21                                          (propertize "(default)" 'face 'font-lock-doc-face)))
22                          value))
23                      values))
24         (let ((selection (consult--read question values :default default-value)))
25           (unless (or (string-match-p "(default)$" selection)
26                       (string= "" selection))
27             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`).

```

1 (defun +yas/org-src-lang ()
2   "Try to find the current language of the src/header at `point'.
3   Return nil otherwise."
4   (let ((context (org-element-context)))
5     (pcase (org-element-type context)
6       ('src-block (org-element-property :language context))
7       ('inline-src-block (org-element-property :language context))
8       ('keyword (when (string-match "^header-args:\\\\([~ ]+\\\\)" (org-element-property :value context))
9                    (match-string 1 (org-element-property :value context))))))
10
11 (defun +yas/org-last-src-lang ()
12   "Return the language of the last src-block, if it exists."
13   (save-excursion

```

```

14 (beginning-of-line)
15 (when (re-search-backward "[ \t]*#\\+begin_src" nil t)
16   (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 occurrences" 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

```

1 (cl-defmacro +lsp-org-babel-enable (lang)
2   "Support LANG in org source code block."
3   ;; (setq centaur-lsp 'lsp-mode)
4   (cl-check-type lang stringp)
5   (let* ((edit-pre (intern (format "org-babel-edit-pre:%s" lang)))
6          (intern-pre (intern (format "lsp--%s" (symbol-name edit-pre)))))
7     `(progn
8       (defun ,intern-pre (info)
9         (let ((file-name (->> info caddr (alist-get :file))))
10          (unless file-name
11            (setq file-name (make-temp-file "babel-lsp-"))
12            (setq buffer-file-name file-name)
13            (lsp-deferred)))
14          (put ',intern-pre 'function-documentation
15              (format "Enable lsp-mode in the buffer of org source block (%s)."
16                    (upcase ,lang)))
17          (if (fboundp ',edit-pre)
18              (advice-add ',edit-pre :after ',intern-pre)
19              (progn
20                (defun ,edit-pre (info)
21                  (,intern-pre info))
22                (put ',edit-pre 'function-documentation
23                    (format "Prepare local buffer environment for org source block (%s)."
24                          (upcase ,lang)))))))
25
26 (defvar +org-babel-lang-list
27   '("go" "python" "ipython" "bash" "sh"))
28
29 (dolist (lang +org-babel-lang-list)
30   (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 L<sup>A</sup>T<sub>E</sub>X, taken from “Export subfigures to L<sup>A</sup>T<sub>E</sub>X (and HTML)”.

```

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

```

Example of usage:

```

#+caption: Lorem ipsum 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
```

**L<sup>A</sup>T<sub>E</sub>X inline markup** Needs to make a `?`, with this hack you can write `[[latex:textsc][Some text]]`.

```
1 (org-add-link-type
2   "latex" nil
3   (lambda (path desc format)
4     (cond
5       ((eq format 'html)
6        (format "<span class=\"%s\">%s</span>" path desc))
7       ((eq format 'latex)
8        (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:

```
1 (custom-set-faces!
2   '(org-document-title :height 1.2))
3
4 (custom-set-faces!
5   '(outline-1 :weight extra-bold :height 1.25)
6   '(outline-2 :weight bold :height 1.15)
7   '(outline-3 :weight bold :height 1.12)
8   '(outline-4 :weight semi-bold :height 1.09)
9   '(outline-5 :weight semi-bold :height 1.06)
10  '(outline-6 :weight semi-bold :height 1.03)
11  '(outline-8 :weight semi-bold)
12  '(outline-9 :weight semi-bold))
```

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

```
1 (setq org-agenda-deadline-faces
2   '((1.001 . error)
3     (1.000 . org-warning)
4     (0.500 . org-upcoming-deadline)
5     (0.000 . org-upcoming-distant-deadline)))
```

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

```
1 (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.

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

## Inline blocks

```

1 (use-package! org-modern
2   :hook (org-mode . org-modern-mode)
3   :config
4     (setq org-modern-star '(" " " " " " " " " " " " " " " " " ")
5       org-modern-table-vertical 5
6       org-modern-table-horizontal 2
7       org-modern-list '((43 . " ") (45 . "-") (42 . "."))
8       org-modern-footnote (cons nil (cadr org-script-display))
9       org-modern-priority t
10      org-modern-block t
11      org-modern-block-fringe nil
12      org-modern-horizontal-rule t
13      org-modern-keyword
14        '(
15          ((t . t)
16            ("title" . " ")
17            ("subtitle" . " ")
18            ("author" . " ")
19            ("email" . "@")
20            ("date" . " ")
21            ("lastmod" . " ")
22            ("property" . " ")
23            ("options" . " ")
24            ("startup" . " ")
25            ("macro" . " ")
26            ("bind" . #(" " 0 1 (display (raise -0.1))))
27            ("bibliography" . " ")
28            ("print_bibliography" . #(" " 0 1 (display (raise -0.1))))
29            ("cite_export" . " ")
30            ("print_glossary" . #(" " 0 1 (display (raise -0.1))))
31            ("glossary_sources" . #(" " 0 1 (display (raise -0.14))))
32            ("export_file_name" . " ")
33            ("include" . " ")
34            ("setupfile" . " ")
35            ("html_head" . " ")
36            ("html" . " ")
37            ("latex_class" . " ")
38            ("latex_class_options" . #(" " 1 2 (display (raise -0.14))))
39            ("latex_header" . " ")
40            ("latex_header_extra" . " ")
41            ("latex" . " ")
42            ("beamer_theme" . " ")
43            ("beamer_color_theme" . #(" " 1 2 (display (raise -0.12))))
44            ("beamer_font_theme" . " ")
45            ("beamer_header" . " ")
46            ("beamer" . " ))

```

```

46      ("attr_latex"      . " ")
47      ("attr_html"      . " ")
48      ("attr_org"       . " ")
49      ("name"           . " ")
50      ("header"         . ">")
51      ("caption"        . " ")
52      ("RESULTS"        . " ")
53      ("language"       . " ")
54      ("hugo_base_dir"   . " ")
55      ("latex_compiler" . " ")
56      ("results"        . " ")
57      ("filetags"       . "#")
58      ("created"        . " ")
59      ("export_select_tags" . " ")
60      ("export_exclude_tags" . " "))
61
62  ;; Change faces
63  (custom-set-faces! '(org-modern-tag :inherit (region org-modern-label)))
64  (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`.

```

1  (when (modulep! :ui ligatures)
2    (defadvice! +org-init-appearance-h--no-ligatures-a ()
3      :after #' +org-init-appearance-h
4      (set-ligatures! 'org-mode
5        :name nil
6        :src_block nil
7        :src_block_end nil
8        :quote nil
9        :quote_end nil)))

```

We'll bind this to 0 on the `org-mode` localleader, and manually apply a PR recognising the pgtk window system.

```

1  (use-package! org-ol-tree
2    :commands org-ol-tree
3    :config
4    (setq org-ol-tree-ui-icon-set
5      (if (and (display-graphic-p)
6        (fboundp 'all-the-icons-material))
7        'all-the-icons
8        'unicode))
9    (org-ol-tree-ui--update-icon-set))
10
11  (map! :localleader
12    :map org-mode-map
13    :desc "Outline" "0" #'org-ol-tree)

```

```

1  (defvar +org-responsive-image-percentage 0.4)
2  (defvar +org-responsive-image-width-limits '(400 . 700)) ;; '(min-width . max-width)
3
4  (defun +org--responsive-image-h ()
5    (when (eq major-mode 'org-mode)
6      (setq org-image-actual-width
7        (max (car +org-responsive-image-width-limits)
8          (min (cdr +org-responsive-image-width-limits)
9            (truncate (* (window-pixel-width) +org-responsive-image-percentage)))))))
10
11  (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

```
1 (setq org-list-demote-modify-bullet
2   '(("+" . "-")
3     ("-" . "+")
4     ("*" . "+")
5     ("1." . "a.")))
```

```
1 ;; Org styling, hide markup etc.
2 (setq org-hide-emphasis-markers t
3       org-pretty-entities t
4       org-ellipsis " "
5       org-hide-leading-stars t)
6 ;; org-priority-highest ?A
7 ;; org-priority-lowest ?E
8 ;; org-priority-faces
9 ;; ' (?A . 'all-the-icons-red)
10 ;;   (?B . 'all-the-icons-orange)
11 ;;   (?C . 'all-the-icons-yellow)
12 ;;   (?D . 'all-the-icons-green)
13 ;;   (?E . 'all-the-icons-blue)))
```

## Symbols

### L<sup>A</sup>T<sub>E</sub>X fragments

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

```
1 (setq org-highlight-latex-and-related '(native script entities))
2
3 (require 'org-src)
4 (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.

```
1 (setq org-format-latex-options
2   (plist-put org-format-latex-options :background "Transparent"))
3
4 ;; Can be dvipng, dvisvgm, imagemagick
5 (setq org-preview-latex-default-process 'dvisvgm)
6
7 ;; Define a function to set the format latex scale (to be reused in hooks)
8 (defun +org-format-latex-set-scale (scale)
9   (setq org-format-latex-options (plist-put org-format-latex-options :scale scale)))
10
11 ;; Set the default scale
12 (+org-format-latex-set-scale 1.4)
13
14 ;; Increase scale in Zen mode
15 (when (modulep! :ui zen)
16   (add-hook! 'writeroom-mode-enable-hook (+org-format-latex-set-scale 2.0))
17   (add-hook! 'writeroom-mode-disable-hook (+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.**

```

1 (defun +parse-the-fun (str)
2   "Parse the LaTeX environment STR.
3   Return an AST with newlines counts in each level."
4   (let (ast)
5     (with-temp-buffer
6       (insert str)
7       (goto-char (point-min))
8       (while (re-search-forward
9         (rx "\\\"
10          (group (or "\\\" \"begin\" \"end\" \"nonnumber\"))
11          (zero-or-one "{" (group (zero-or-more not-newline)) "}"))
12         nil t)
13       (let ((cmd (match-string 1))
14             (env (match-string 2)))
15         (cond ((string= cmd "begin")
16               (push (list :env (intern env)) ast))
17               ((string= cmd "\\\"
18                 (let ((curr (pop ast)))
19                   (push (plist-put curr :newline (1+ (or (plist-get curr :newline) 0))) ast)))
20               ((string= cmd "nonnumber")
21                 (let ((curr (pop ast)))
22                   (push (plist-put curr :nonnumber (1+ (or (plist-get curr :nonnumber) 0))) ast)))
23               ((string= cmd "end")
24                 (let ((child (pop ast))
25                       (parent (pop ast)))
26                   (push (plist-put parent :childs (cons child (plist-get parent :childs))) ast))))))
27       (plist-get (car ast) :childs)))
28
29 (defun +scimax-org-renumber-environment (orig-func &rest args)
30   "A function to inject numbers in LaTeX fragment previews."
31   (let ((results '())
32         (counter -1))
33     (setq results
34           (cl-loop for (begin . env) in
35                     (org-element-map (org-element-parse-buffer) 'latex-environment
36                                       (lambda (env)
37                                         (cons
38                                           (org-element-property :begin env)
39                                           (org-element-property :value env))))
40                     collect
41                     (cond
42                      ((and (string-match "\\\"\\begin{equation}" env)
43                          (not (string-match "\\\"\\tag{" env)))
44                       (cl-incf counter)
45                       (cons begin counter))
46                      ((string-match "\\\"\\begin{align}" env)
47                       (cl-incf counter)
48                       (let ((p (car (+parse-the-fun env))))
49                         ;; Parse the `env`, count new lines in the align env as equations, unless
50                         (cl-incf counter (- (or (plist-get p :newline) 0)
51                                              (or (plist-get p :nonnumber) 0))))
52                       (cons begin counter))
53                      (t
54                       (cons begin nil)))))
55     (when-let ((number (cdr (assoc (point) results))))
56       (setf (car args)
57             (concat
58               (format "\\setcounter{equation}{%s}\\n" number)
59               (car args))))
60     (apply orig-func args))
61
62 (defun +scimax-toggle-latex-equation-numbering (&optional enable)

```



```

63 "Toggle whether LaTeX fragments are numbered."
64 (interactive)
65 (if (or enable (not (get '+scimax-org-renumber-environment 'enabled)))
66     (progn
67       (advice-add 'org-create-formula-image :around #' +scimax-org-renumber-environment)
68       (put '+scimax-org-renumber-environment 'enabled t)
69       (message "LaTeX numbering enabled."))
70     (advice-remove 'org-create-formula-image #' +scimax-org-renumber-environment)
71     (put '+scimax-org-renumber-environment 'enabled nil)
72     (message "LaTeX numbering disabled.")))
73
74 (defun +scimax-org-inject-latex-fragment (orig-func &rest args)
75   "Advice function to inject latex code before and/or after the equation in a latex fragment.
76   You can use this to set \\mathversion{bold} for example to make
77   it bolder. The way it works is by defining
78   :latex-fragment-pre-body and/or :latex-fragment-post-body in the
79   variable `org-format-latex-options'. These strings will then be
80   injected before and after the code for the fragment before it is
81   made into an image."
82   (setf (car args)
83         (concat
84           (or (plist-get org-format-latex-options :latex-fragment-pre-body) "")
85           (car args)
86           (or (plist-get org-format-latex-options :latex-fragment-post-body) "")))
87   (apply orig-func args))
88
89 (defun +scimax-toggle-inject-latex ()
90   "Toggle whether you can insert latex in fragments."
91   (interactive)
92   (if (not (get '+scimax-org-inject-latex-fragment 'enabled))
93       (progn
94         (advice-add 'org-create-formula-image :around #' +scimax-org-inject-latex-fragment)
95         (put '+scimax-org-inject-latex-fragment 'enabled t)
96         (message "Inject latex enabled"))
97       (advice-remove 'org-create-formula-image #' +scimax-org-inject-latex-fragment)
98       (put '+scimax-org-inject-latex-fragment 'enabled nil)
99       (message "Inject latex disabled")))
100
101 ;; Enable renumbering by default
102 (+scimax-toggle-latex-equation-numbering t)

```

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

```

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

```

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

```

1 (after! org-plot
2   (defun org-plot/generate-theme (_type)
3     "Use the current Doom theme colours to generate a GnuPlot preamble."
4     (format "
5 fgt = \"textcolor rgb '%s'\" # foreground text
6 fgat = \"textcolor rgb '%s'\" # foreground alt text
7 fgl = \"linecolor rgb '%s'\" # foreground line
8 fgat = \"linecolor rgb '%s'\" # foreground alt line
9
10 # foreground colors
11 set border lc rgb '%s'
12 # change text colors of tics
13 set xtics @fgt
14 set ytics @fgt
15 # change text colors of labels
16 set title @fgt

```

```

17 set xlabel @fgt
18 set ylabel @fgt
19 # change a text color of key
20 set key @fgt
21
22 # line styles
23 set linetype 1 lw 2 lc rgb '%s' # red
24 set linetype 2 lw 2 lc rgb '%s' # blue
25 set linetype 3 lw 2 lc rgb '%s' # green
26 set linetype 4 lw 2 lc rgb '%s' # magenta
27 set linetype 5 lw 2 lc rgb '%s' # orange
28 set linetype 6 lw 2 lc rgb '%s' # yellow
29 set linetype 7 lw 2 lc rgb '%s' # teal
30 set linetype 8 lw 2 lc rgb '%s' # violet
31
32 # palette
33 set palette maxcolors 8
34 set palette defined ( 0 '%s',\
35 1 '%s',\
36 2 '%s',\
37 3 '%s',\
38 4 '%s',\
39 5 '%s',\
40 6 '%s',\
41 7 '%s' )
42 "
43         (doom-color 'fg)
44         (doom-color 'fg-alt)
45         (doom-color 'fg)
46         (doom-color 'fg-alt)
47         (doom-color 'fg)
48         ;; colours
49         (doom-color 'red)
50         (doom-color 'blue)
51         (doom-color 'green)
52         (doom-color 'magenta)
53         (doom-color 'orange)
54         (doom-color 'yellow)
55         (doom-color 'teal)
56         (doom-color 'violet)
57         ;; duplicated
58         (doom-color 'red)
59         (doom-color 'blue)
60         (doom-color 'green)
61         (doom-color 'magenta)
62         (doom-color 'orange)
63         (doom-color 'yellow)
64         (doom-color 'teal)
65         (doom-color 'violet)))
66
67 (defun org-plot/gnuplot-term-properties (_type)
68   (format "background rgb '%s' size 1050,650"
69     (doom-color 'bg)))
70
71 (setq org-plot/gnuplot-script-preamble #'org-plot/generate-theme
72       org-plot/gnuplot-term-extra #'org-plot/gnuplot-term-properties))

```

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

```

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

```

### 9.2.5 Bibliography

```

1 (setq bibtex-completion-bibliography +my/biblio-libraries-list
2       bibtex-completion-library-path +my/biblio-storage-list
3       bibtex-completion-notes-path +my/biblio-notes-path
4       bibtex-completion-notes-template-multiple-files "*" ${author-or-editor}, ${title}, ${journal}, (${year})
↪ :${type}=: \n\nSee [[cite:&${key=}]]\n"
5       bibtex-completion-additional-search-fields '(keywords)
6       bibtex-completion-display-formats
7       '( (article . "${has-pdf=:1}${has-note=:1} ${year:4} ${author:36} ${title:*} ${journal:40}")
8         (inbook . "${has-pdf=:1}${has-note=:1} ${year:4} ${author:36} ${title:*} Chapter
↪ ${chapter:32}")
9         (incollection . "${has-pdf=:1}${has-note=:1} ${year:4} ${author:36} ${title:*} ${booktitle:40}")
10        (inproceedings . "${has-pdf=:1}${has-note=:1} ${year:4} ${author:36} ${title:*} ${booktitle:40}")
11        (t . "${has-pdf=:1}${has-note=:1} ${year:4} ${author:36} ${title:*}")
12        bibtex-completion-pdf-open-function
13        (lambda (fpath)
14          (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.

```

1 (use-package! org-bib
2   :commands (org-bib-mode))

```

```

1 (after! oc
2   (setq org-cite-csl-styles-dir +my/biblio-styles-path)
3   ;; org-cite-global-bibliography +my/biblio-libraries-list)
4
5   (defun +org-ref-to-org-cite ()
6     "Simple conversion of org-ref citations to org-cite syntax."
7     (interactive)
8     (save-excursion
9       (goto-char (point-min))
10      (while (re-search-forward "\\[cite\\(.\\*\\):\\([~]*\\)\\]" nil t)
11        (let* ((old (substring (match-string 0) 1 (1- (length (match-string 0)))))
12              (new (s-replace "&" "@" old)))
13          (message "Replaced citation %s with %s" old new)
14          (replace-match new))))))

```

## Org-cite

```

1 (after! citar
2   (setq citar-library-paths +my/biblio-storage-list
3       citar-notes-paths (list +my/biblio-notes-path)
4       citar-bibliography +my/biblio-libraries-list
5       citar-symbol-separator " ")
6
7   (when (display-graphic-p)
8     (setq citar-symbols
9       `((file ,(all-the-icons-octicon "file-pdf" :face 'error) . " ")
10         (note ,(all-the-icons-octicon "file-text" :face 'warning) . " ")
11         (link ,(all-the-icons-octicon "link-external" :face 'org-link) . " "))))))
12
13 (use-package! citar-org-roam
14   :after citar org-roam
15   :no-require

```

```

16 :config (citar-org-roam-mode)
17 :init
18 ;; Modified form: https://jethrokuan.github.io/org-roam-guide/
19 (defun +org-roam-node-from-cite (entry-key)
20   (interactive (list (citar-select-ref)))
21   (let ((title (citar-format--entry
22                 "${author editor} (${date urldate}) :: ${title}"
23                 (citar-get-entry entry-key))))
24     (org-roam-capture- :templates
25                       '(("r" "reference" plain
26                           "%?"
27                           :if-new (file+head "references/${citekey}.org"
28                                               ":properties:
29 :roam_refs: [cite:@${citekey}]
30 :end:
31 #+title: ${title}\n")
32                           :immediate-finish t
33                           :unnarrowed t))
34       :info (list :citekey entry-key)
35       :node (org-roam-node-create :title title)
36       :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.

```

1 (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:`).

```

1 (require 'ox-extra)
2 (ox-extras-activate '(ignore-headlines))

```

```

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

```

## L<sup>A</sup>T<sub>E</sub>X export

```

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

```

## Compiling

```

1 ;; 'svg' package depends on inkscape, imagemagick and ghostscript
2 (when (+all (mapcar 'executable-find '("inkscape" "magick" "gs"))))
3   (add-to-list 'org-latex-packages-alist '("" "svg")))
4
5 (add-to-list 'org-latex-packages-alist '("svgnames" "xcolor"))
6 ;; (add-to-list 'org-latex-packages-alist '("" "fontspec")) ;; for xelatex
7 ;; (add-to-list 'org-latex-packages-alist '("utf8" "inputenc"))

```

## Org L<sup>A</sup>T<sub>E</sub>X 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.

```

1  ;; Should be configured per document, as a local variable
2  ;; (setq org-latex-listings 'minted)
3  ;; (add-to-list 'org-latex-packages-alist '(" " "minted"))
4
5  ;; Default `minted` options, can be overwritten in file/dir locals
6  (setq org-latex-minted-options
7    '(("frame"          "lines")
8      ("fontsize"       "\\footnotesize")
9      ("tabsize"        "2")
10     ("breaklines"      "true")
11     ("breakanywhere"   "true") ;; break anywhere, no just on spaces
12     ("style"           "default")
13     ("bgcolor"         "GhostWhite")
14     ("linenos"         "true")))
15
16  ;; Link some org-mode blocks languages to lexers supported by minted
17  ;; via (pygmentize), you can see supported lexers by running this command
18  ;; in a terminal: `pygmentize -L lexers`
19  (dolist (pair '((ipython    "python")
20                  (jupyter    "python")
21                  (scheme     "scheme")
22                  (lisp-data   "lisp")
23                  (conf-unix   "unixconfig")
24                  (conf-space  "unixconfig")
25                  (authinfo    "unixconfig")
26                  (gdb-script  "unixconfig")
27                  (conf-toml   "yaml")
28                  (conf        "ini")
29                  (gitconfig   "ini")
30                  (systemd     "ini"))))
31    (unless (member pair org-latex-minted-langs)
32      (add-to-list 'org-latex-minted-langs pair)))

```

```

1  (after! ox-latex
2    (add-to-list
3      'org-latex-classes
4      '("scr-article"
5        "\\documentclass{scrartcl}"
6        ("\\section{%s}" . "\\section*{%s}")
7        ("\\subsection{%s}" . "\\subsection*{%s}")
8        ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
9        ("\\paragraph{%s}" . "\\paragraph*{%s}")
10       ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
11
12  (add-to-list
13    'org-latex-classes
14    '("lettre"
15      "\\documentclass{lettre}"
16      ("\\section{%s}" . "\\section*{%s}")
17      ("\\subsection{%s}" . "\\subsection*{%s}")
18      ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
19      ("\\paragraph{%s}" . "\\paragraph*{%s}")
20      ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
21
22  (add-to-list
23    'org-latex-classes
24    '("blank"
25      "[NO-DEFAULT-PACKAGES] \\n[NO-PACKAGES] \\n[EXTRA]"
26      ("\\section{%s}" . "\\section*{%s}"))

```

```

27  ("\\subsection{%s}" . "\\subsection*{%s}")
28  ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
29  ("\\paragraph{%s}" . "\\paragraph*{%s}")
30  ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))
31
32  (add-to-list
33   'org-latex-classes
34   '("IEEEtran"
35     "\\documentclass{IEEEtran}"
36     ("\\section{%s}" . "\\section*{%s}")
37     ("\\subsection{%s}" . "\\subsection*{%s}")
38     ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
39     ("\\paragraph{%s}" . "\\paragraph*{%s}")
40     ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
41
42  (add-to-list
43   'org-latex-classes
44   '("ieeeconf"
45     "\\documentclass{ieeeconf}"
46     ("\\section{%s}" . "\\section*{%s}")
47     ("\\subsection{%s}" . "\\subsection*{%s}")
48     ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
49     ("\\paragraph{%s}" . "\\paragraph*{%s}")
50     ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
51
52  (add-to-list
53   'org-latex-classes
54   '("sagej"
55     "\\documentclass{sagej}"
56     ("\\section{%s}" . "\\section*{%s}")
57     ("\\subsection{%s}" . "\\subsection*{%s}")
58     ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
59     ("\\paragraph{%s}" . "\\paragraph*{%s}")
60     ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
61
62  (add-to-list
63   'org-latex-classes
64   '("thesis"
65     "\\documentclass[11pt]{book}"
66     ("\\chapter{%s}" . "\\chapter*{%s}")
67     ("\\section{%s}" . "\\section*{%s}")
68     ("\\subsection{%s}" . "\\subsection*{%s}")
69     ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
70     ("\\paragraph{%s}" . "\\paragraph*{%s}"))))
71
72  (add-to-list
73   'org-latex-classes
74   '("thesis-fr"
75     "\\documentclass[french,12pt,a4paper]{book}"
76     ("\\chapter{%s}" . "\\chapter*{%s}")
77     ("\\section{%s}" . "\\section*{%s}")
78     ("\\subsection{%s}" . "\\subsection*{%s}")
79     ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
80     ("\\paragraph{%s}" . "\\paragraph*{%s}"))))
81
82  (setq org-latex-default-class "article")
83
84  ;; org-latex-tables-booktabs t
85  ;; 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:

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.

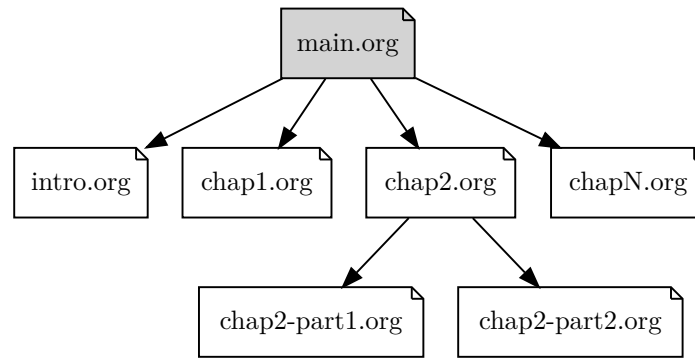


Figure 1: Example of a multi-files document structure

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.

```

1 (defvar +org-export-to-pdf-main-file nil
2   "The main (entry point) Org file for a multi-files document.")
3
4 (advice-add
5   'org-latex-export-to-pdf :around
6   (lambda (orig-fn &rest orig-args)
7     (message
8       "PDF exported to: %s."
9       (let ((main-file (or (bound-and-true-p +org-export-to-pdf-main-file) "main.org")))
10         (if (file-exists-p (expand-file-name main-file))
11             (with-current-buffer (find-file-noselect main-file)
12               (apply orig-fn orig-args))
13             (apply orig-fn orig-args))))))

```

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

```

1 (setq time-stamp-active t
2       time-stamp-start "#\\+lastmod: [ \\t]*"
3       time-stamp-end "$"
4       time-stamp-format "%04Y-%02m-%02d")
5
6 (add-hook 'before-save-hook 'time-stamp nil)
7 (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.

```

1 (after! text-mode
2   (add-hook! 'text-mode-hook
3     (unless (derived-mode-p 'org-mode)
4       ;; Apply ANSI color codes
5       (with-silent-modifications
6         (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”.

```

1 (use-package! academic-phrases
2   :commands (academic-phrases
3             academic-phrases-by-section))

```

### 9.3.3 French apostrophes

```

1 (defun +helper--in-buffer-replace (old new)
2   "Replace OLD with NEW in the current buffer."
3   (save-excursion
4     (goto-char (point-min))
5     (let ((case-fold-search nil)
6         (cnt 0))
7       (while (re-search-forward old nil t)
8         (replace-match new)
9         (setq cnt (1+ cnt)))
10      cnt)))
11
12 (defun +helper-clear-frenchy-ponctuations ()
13   "Replace french apostrophes (') by regular quotes (')."
14   (interactive)
15   (let ((chars '("(" " " . "")) ("'" " " . "")))
16     (cnt 0))
17   (dolist (pair chars)
18     (setq cnt (+ cnt (+helper--in-buffer-replace (car pair) (cdr pair)))))
19   (message "Replaced %d matche(s)." cnt)))

```

### 9.3.4 Yanking multi-lines paragraphs

```

1 (defun +helper-paragraphized-yank ()
2   "Copy, then remove newlines and Org styling (/ *_~)."
3   (interactive)
4   (copy-region-as-kill nil nil t)
5   (with-temp-buffer
6     (yank)
7     ;; Remove newlines, and Org styling (/ *_~)
8     (goto-char (point-min))
9     (let ((case-fold-search nil))
10      (while (re-search-forward "[\n/*_~]" nil t)
11        (replace-match (if (s-matches-p (match-string 0) "\n") " " "" t)))
12      (kill-region (point-min) (point-max))))
13
14 (map! :localleader
15       :map (org-mode-map markdown-mode-map latex-mode-map text-mode-map)
16       :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`.

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

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

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

Then set Emacs as the default editor:

```
1 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.

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

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

```
1 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.

```
1 read -p "Do you want to set Chrome/Brave to show the 'Always open ...' checkbox, to be used with the
2 ↩ 'org-protocol://' registration? [Y | N]: " INSTALL_CONFIRM
3
4 if [[ "$INSTALL_CONFIRM" == "Y" ]]
5 then
6   sudo mkdir -p /etc/opt/chrome/policies/managed/
7
8   sudo tee /etc/opt/chrome/policies/managed/external_protocol_dialog.json > /dev/null <<'EOF'
9   {
10     "ExternalProtocolDialogShowAlwaysOpenCheckbox": true
11   }
12 EOF
```

```

12 sudo chmod 644 /etc/opt/chrome/policies/managed/external_protocol_dialog.json
13
14 fi

```

Then add a bookmarklet in your browser with this code:

```

1 javascript:location.href =
2   'org-protocol://roam-ref?template=r&ref='
3   + encodeURIComponent(location.href)
4   + '&title='
5   + encodeURIComponent(document.title)
6   + '&body='
7   + encodeURIComponent(window.getSelection())

```

## 10.2 Git

### 10.2.1 Git diffs

Based on this gist and this article.

```

1 *.tex diff=tex
2 *.bib diff=bibtex
3 *.{c,h,c++,h++,cc,hh,cpp,hpp} diff=cpp
4 *.m diff=matlab
5 *.py diff=python
6 *.rb diff=ruby
7 *.php diff=php
8 *.pl diff=perl
9 *.{html,xhtml} diff=html
10 *.f diff=fortran
11 *.{el,lisp,scm} diff=lisp
12 *.r diff=rstats
13 *.texi* diff=texinfo
14 *.org diff=org
15 *.rs diff=rust
16
17 *.odt diff=odt
18 *.odp diff=libreoffice
19 *.ods diff=libreoffice
20 *.doc diff=doc
21 *.xls diff=xls
22 *.ppt diff=ppt
23 *.docx diff=docx
24 *.xlsx diff=xlsx
25 *.pptx diff=pptx
26 *.rtf diff=rtf
27
28 *.{png,jpg,jpeg,gif} diff=exif
29
30 *.pdf diff=pdf
31 *.djvu diff=djvu
32 *.epub diff=pandoc
33 *.chm diff=tika
34 *.mhtml? diff=tika
35
36 *.{class,jar} diff=tika
37 *.{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.

```

1 # ===== TEXT FORMATS =====
2 [diff "org"]
3   xfuncname = "^(\\"*+ +.*)"

```

```

4
5 [diff "lisp"]
6     xfuncname = "^((\\(\\.\\)*))$"
7
8 [diff "rstats"]
9     xfuncname = "^([a-zA-Z.]< <- function.\\)*$"
10
11 [diff "texinfo"]
12 # from http://git.savannah.gnu.org/gitweb/?p=coreutils.git;a=blob;f=.gitattributes;h=c3b2926c78c939d94358cc63d05
↵ 1a70d38cfea5d;hb=HEAD
13     xfuncname = "@node[ \\t][ \\t]*\\([\\^,][\\^,]*\\)"
14
15 [diff "rust"]
16     xfuncname = "[ \\t]*(pub|)[ \\t]*((fn|struct|enum|impl|trait|mod)[^;]*)$"
17
18 # ===== BINARY FORMATS =====
19 [diff "pdf"]
20     binary = true
21 # textconv = pdfinfo
22 # textconv = sh -c 'pdftotext "$@" -' # sudo apt install pdftotext
23 textconv = sh -c 'pdftotext -layout "$@" -enc UTF-8 -nopgbrk -q -'
24     cachetextconv = true
25
26 [diff "djvu"]
27     binary = true
28 # textconv = pdfinfo
29 textconv = djvutxt # yay -S djvulibre
30     cachetextconv = true
31
32 [diff "odt"]
33     textconv = odt2txt
34 # textconv = pandoc --standalone --from=odt --to=plain
35     binary = true
36     cachetextconv = true
37
38 [diff "doc"]
39 # textconv = wvText
40 textconv = catdoc # yay -S catdoc
41     binary = true
42     cachetextconv = true
43
44 [diff "xls"]
45 # textconv = in2csv
46 # textconv = xlscat -a UTF-8
47 # textconv = soffice --headless --convert-to csv
48 textconv = xls2csv # yay -S catdoc
49     binary = true
50     cachetextconv = true
51
52 [diff "ppt"]
53 textconv = catppt # yay -S catdoc
54     binary = true
55     cachetextconv = true
56
57 [diff "docx"]
58 textconv = pandoc --standalone --from=docx --to=plain
59 # textconv = sh -c 'docx2txt.pl "$@" -'
60     binary = true
61     cachetextconv = true
62
63 [diff "xlsx"]
64 textconv = xlsx2csv # pip install xlsx2csv
65 # textconv = in2csv
66 # textconv = soffice --headless --convert-to csv
67     binary = true
68     cachetextconv = true
69
70 [diff "pptx"]
71 # pip install --user pptx2md (currently not working with Python 3.10)
72 # textconv = sh -c 'pptx2md --disable_image --disable_umf -i "$@" -o ~/.cache/git/presentation.md >/dev/null &&
↵ cat ~/.cache/git/presentation.md'

```

```

73 # Alternative hack, convert PPTX to PPT, then use the catppt tool
74 textconv = sh -c 'soffice --headless --convert-to ppt --outdir /tmp "$@" && TMP_FILENAME=$(basename -- "$@")
↪ && catppt "/tmp/${TMP_FILENAME%.*}.ppt"'
75 binary = true
76 cachetextconv = true
77
78 [diff "rtf"]
79 textconv = unrtf --text # yay -S unrtf
80 binary = true
81 cachetextconv = true
82
83 [diff "epub"]
84 textconv = pandoc --standalone --from=epub --to=plain
85 binary = true
86 cachetextconv = true
87
88 [diff "tika"]
89 textconv = tika --config=~/.local/share/tika/tika-conf.xml --text
90 binary = true
91 cachetextconv = true
92
93 [diff "libreoffice"]
94 textconv = soffice --cat
95 binary = true
96 cachetextconv = true
97
98 [diff "exif"]
99 binary = true
100 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`:

```

1 #!/bin/sh
2 APACHE_TIKA_JAR="$HOME/.local/share/tika/tika-app.jar"
3
4 if [ -f "${APACHE_TIKA_JAR}" ]
5 then
6     exec java -Dfile.encoding=UTF-8 -jar "${APACHE_TIKA_JAR}" "$@" 2>/dev/null
7 else
8     echo "JAR file not found at ${APACHE_TIKA_JAR}"
9 fi

```

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

```

1 update_apache_tika () {
2     TIKA_JAR_PATH="$HOME/.local/share/tika"
3
4     if [ ! -d "${TIKA_JAR_PATH}" ]
5     then
6         mkdir -p "${TIKA_JAR_PATH}"
7     fi
8
9     TIKA_BASE_URL=https://archive.apache.org/dist/tika/
10    TIKA_JAR_LINK="${TIKA_JAR_PATH}/tika-app.jar"
11
12    echo -n "Checking for new Apache Tika App version... "
13
14    # Get the latest version
15    TIKA_VERSION=$(
16        curl -s "${TIKA_BASE_URL}" | # Get the page

```

```

17  pandoc -f html -t plain | # Convert HTML page to plain text.
18  awk '/([0-9]+\.)+[0-1]\\/{print substr($1, 0, length($1)-1)}' | # Get the versions directories (pattern:
    ↪ X.X.X/)
19  sort -rV | # Sort versions, the newest first
20  head -n 1 # Get the first (newest) version
21  )
22
23  if [ -z "${TIKA_VERSION}" ]
24  then
25      echo "Failed, check your internet connection."
26      exit 1
27  fi
28
29  echo "Lastest version is ${TIKA_VERSION}"
30
31  TIKA_JAR="${TIKA_JAR_PATH}/tika-app-${TIKA_VERSION}.jar"
32  TIKA_JAR_URL="${TIKA_BASE_URL}/${TIKA_VERSION}/tika-app-${TIKA_VERSION}.jar"
33
34  if [ ! -f "${TIKA_JAR}" ]
35  then
36      echo "New version available!"
37      read -p "Do you want to download Apache Tika App v${TIKA_VERSION}? [Y | N]: " INSTALL_CONFIRM
38      if [[ "$INSTALL_CONFIRM" == "Y" ]]
39      then
40          curl -o "${TIKA_JAR}" "${TIKA_JAR_URL}" && echo "Apache Tika App v${TIKA_VERSION} downloaded successfully"
41          fi
42      else
43          echo "Apache Tika App is up-to-date, version ${TIKA_VERSION} already downloaded to '${TIKA_JAR}'"
44      fi
45
46      # Check the existance of the symbolic link
47      if [ -L "${TIKA_JAR_LINK}" ]
48      then
49          unlink "${TIKA_JAR_LINK}"
50      fi
51
52      # Create a symbolic link to the installed version
53      ln -s "${TIKA_JAR}" "${TIKA_JAR_LINK}"
54  }
55
56  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).

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <properties>
3    <parsers>
4      <parser class="org.apache.tika.parser.DefaultParser">
5        <parser-exclude class="org.apache.tika.parser.ocr.TesseractOCRParser"/>
6      </parser>
7    </parsers>
8  </properties>

```

## 10.3 Emacs' Systemd daemon

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

```

1  [Unit]
2  Description=Emacs server daemon
3  Documentation=info:emacs man:emacs(1) https://gnu.org/software/emacs/
4
5  [Service]

```

```

6 Type=forking
7 ExecStart=sh -c 'emacs --daemon && emacsclient -c --eval "(delete-frame)'"
8 ExecStop=emacsclient --no-wait --eval "(progn (setq kill-emacs-hook nil) (kill-emacs))"
9 Restart=on-failure
10
11 [Install]
12 WantedBy=default.target

```

Which is then enabled by:

```

1 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.

```

1 [Desktop Entry]
2 Name=Emacs (Client)
3 GenericName=Text Editor
4 Comment=A flexible platform for end-user applications
5 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-shellscrip;text/x-c;text/x-c++;
6 Exec=emacsclient -create-frame --frame-parameters="(fullscreen . maximized)"
  ↪ --alternate-editor="/usr/bin/emacs" --no-wait %F
7 Icon=emacs
8 Type=Application
9 Terminal=false
10 Categories=TextEditor;Utility;
11 StartupWMClass=Emacs
12 Keywords=Text;Editor;
13 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.

```

1 #!/usr/bin/env bash
2 force_tty=false
3 force_wait=false
4 stdin_mode=""
5
6 args=()

```

```

7
8 usage () {
9     echo -e "Usage: e [-t] [-m MODE] [OPTIONS] FILE [-]"
10
11     Emacs client convenience wrapper.
12
13     Options:
14     -h, --help           Show this message
15     -t, -nw, --tty       Force terminal mode
16     -w, --wait           Don't supply --no-wait to graphical emacsclient
17     -                     Take stdin (when last argument)
18     -m MODE, --mode MODE Mode to open stdin with
19     -mm, --maximized     Start Emacs client in maximized window
20
21     Run emacsclient --help to see help for the emacsclient."
22 }
23
24 while :
25 do
26     case "$1" in
27         -t | -nw | --tty)
28             force_tty=true
29             shift ;;
30         -w | --wait)
31             force_wait=true
32             shift ;;
33         -m | --mode)
34             stdin_mode=" ($2-mode)"
35             shift 2 ;;
36         -mm | --maximized)
37             args+=("--frame-parameters='(fullscreen . maximized)")
38             shift ;;
39         -h | --help)
40             usage
41             exit 0 ;;
42         --*=*)
43             set -- "$@" "${1%*=}" "${1#*=}"
44             shift ;;
45         *)
46             [ "$#" = 0 ] && break
47             args+=("$1")
48             shift ;;
49         esac
50     done
51
52     if [ ! "${#args[*]}" = 0 ] && [ "${args[-1]}" = "-" ]
53     then
54         unset 'args[-1]'
55         TMP="$(mktemp /tmp/emacsstdin-XXX)"
56         cat > "$TMP"
57         args+=((--eval "(let ((b (generate-new-buffer \"*stdin*\"))) (switch-to-buffer b) (insert-file-contents
58             ↪ \"$TMP\") (delete-file \"$TMP\")${stdin_mode}))")
59     fi
60
61     if [ -z "$DISPLAY" ] || $force_tty
62     then
63         # detect terminals with sneaky 24-bit support
64         if { [ "$COLORTERM" = truecolor ] || [ "$COLORTERM" = 24bit ]; } \
65             && [ "$(tput colors 2>/dev/null)" -lt 257 ]
66         then
67             if echo "$TERM" | grep -q "^\\w\\+-[0-9]"
68             then
69                 termstub="${TERM%*-}"
70             else
71                 termstub="${TERM#*-}"
72             fi
73
74             if infocmp "$termstub-direct" >/dev/null 2>&1
75             then
76                 TERM="$termstub-direct"

```

```

76     else
77         TERM="xterm-direct"
78     fi # should be fairly safe
79 fi
80
81 emacsclient --tty -create-frame --alternate-editor="/usr/bin/emacs" "${args[@]}"
82 else
83     if ! $force_wait
84     then
85         args+=(--no-wait)
86     fi
87
88     emacsclient -create-frame --alternate-editor="/usr/bin/emacs" "${args[@]}"
89 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.

```

1  # Aliases to run emacs+magit
2  alias magit='e --eval "(progn (magit-status) (delete-other-windows))"'
3  alias magitt='e -t --eval "(progn (magit-status) (delete-other-windows))"'
4
5  # Aliases to run emacs+mu4e
6  alias emu='e --eval "(progn (=mu4e) (delete-other-windows))"'
7  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`

```

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

```

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

```

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

```

```

1  export EDITOR="$HOME/.local/bin/et"
2  # export VISUAL="$HOME/.local/bin/ev"

```

## 10.5 AppImage

Install/update the `appimageupdatetool.AppImage` tool:

```

1  update_appimageupdatetool () {
2      TOOL_NAME=appimageupdatetool
3      MACHINE_ARCH=$(uname -m)
4      APPIMAGE_UPDATE_TOOL_PATH="$HOME/.local/bin/${TOOL_NAME}"
5      APPIMAGE_UPDATE_TOOL_URL="https://github.com/AppImage/AppImageUpdate/releases/download/continuous/${TOOL_NAME}
6      ↪ -${MACHINE_ARCH}.AppImage"
7
8      if [ -f "${APPIMAGE_UPDATE_TOOL_PATH}" ] && "${APPIMAGE_UPDATE_TOOL_PATH}" -j "${APPIMAGE_UPDATE_TOOL_PATH}"
9      ↪ 2&>/dev/null
10     then
11         echo "${TOOL_NAME} already up to date"
12     else
13         if [ -f "${APPIMAGE_UPDATE_TOOL_PATH}" ]
14         then
15             echo "Update available, downloading latest ${MACHINE_ARCH} version to ${APPIMAGE_UPDATE_TOOL_PATH}"
16             mv "${APPIMAGE_UPDATE_TOOL_PATH}" "${APPIMAGE_UPDATE_TOOL_PATH}.backup"

```



```

15     else
16         echo "${TOOL_NAME} not found, downloading latest ${MACHINE_ARCH} version to ${APPIMAGE_UPDATE_TOOL_PATH}"
17     fi
18     wget -O "${APPIMAGE_UPDATE_TOOL_PATH}" "${APPIMAGE_UPDATE_TOOL_URL}" && # 2&>/dev/null
19     echo "Downloaded ${TOOL_NAME}-${MACHINE_ARCH}.AppImage" &&
20     [ -f "${APPIMAGE_UPDATE_TOOL_PATH}.backup" ] &&
21     rm "${APPIMAGE_UPDATE_TOOL_PATH}.backup"
22     chmod a+x "${APPIMAGE_UPDATE_TOOL_PATH}"
23 fi
24 }
25
26 update_appimageupdatetool;

```

## 10.6 Oh-my-Zsh

### 10.6.1 Path

Path to your oh-my-zsh installation.

```

1 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](https://github.com/ohmyzsh/ohmyzsh/wiki/Themes).

```

1 # Typewritten customizations
2 TYPEWRITTEN_RELATIVE_PATH="adaptive"
3 TYPEWRITTEN_CURSOR="underscore"
4
5 ZSH_THEME="typewritten/typewritten"
6
7 # Set list of themes to pick from when loading at random
8 # Setting this variable when ZSH_THEME=random will cause zsh to load
9 # a theme from this variable instead of looking in $ZSH/themes/
10 # If set to an empty array, this variable will have no effect.
11 # ZSH_THEME_RANDOM_CANDIDATES=( "robbyrussell" "agnoster" )

```

### 10.6.3 Behavior

```

1 # Uncomment the following line to use case-sensitive completion.
2 # CASE_SENSITIVE="true"
3
4 # Uncomment the following line to use hyphen-insensitive completion.
5 # Case-sensitive completion must be off. _ and - will be interchangeable.
6 # HYPHEN_INSENSITIVE="true"
7
8 # Uncomment the following line to disable bi-weekly auto-update checks.
9 # DISABLE_AUTO_UPDATE="true"
10
11 # Uncomment the following line to automatically update without prompting.
12 DISABLE_UPDATE_PROMPT="true"
13
14 # Uncomment the following line to change how often to auto-update (in days).
15 export UPDATE_ZSH_DAYS=3
16
17 # Uncomment the following line if pasting URLs and other text is messed up.
18 # DISABLE_MAGIC_FUNCTIONS="true"
19

```

```

20 # Uncomment the following line to disable colors in ls.
21 # DISABLE_LS_COLORS="true"
22
23 # Uncomment the following line to disable auto-setting terminal title.
24 # DISABLE_AUTO_TITLE="true"
25
26 # Uncomment the following line to enable command auto-correction.
27 # ENABLE_CORRECTION="true"
28
29 # Uncomment the following line to display red dots whilst waiting for completion.
30 # COMPLETION_WAITING_DOTS="true"
31
32 # Uncomment the following line if you want to disable marking untracked files
33 # under VCS as dirty. This makes repository status check for large repositories
34 # much, much faster.
35 # 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.

```

1 # Define aliases to 'pbcopy' and 'pbpaste'
2 if command -v xclip &> /dev/null
3 then
4     # Define aliases using xclip
5     alias pbcopy='xclip -selection clipboard'
6     alias pbpaste='xclip -selection clipboard -o'
7 elif command -v xsel &> /dev/null
8 then
9     # Define aliases using xsel
10    alias pbcopy='xsel --clipboard --input'
11    alias pbpaste='xsel --clipboard --output'
12 fi

```

### 10.7.2 netpaste

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

```

1 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**.

```

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

```

### 10.7.4 Neovim

Use Neovim instead of VIM to provide **vi** and **vim** commands.

```

1 # NeoVim
2 if command -v nvim &> /dev/null
3 then
4     alias vim="nvim"
5     alias vi="nvim"
6 fi

```

### 10.7.5 ESP-IDF

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

```

1 if [ -d "$HOME/Softwares/src/esp-idf/" ]
2 then
3     alias esp-prepare-env='source $HOME/Softwares/src/esp-idf/export.sh'
4     alias esp-update='echo "Updating ESP-IDF framework..." && cd $HOME/src/esp-idf && git pull --all && echo
5     ↪ "Updated successfully"'
6 else
7     alias esp-prepare-env='echo "esp-idf repo not found. You can clone the esp-idf repo using git clone
8     ↪ https://github.com/espressif/esp-idf.git"'
9     alias esp-update=esp-prepare-env
10 fi

```

### 10.7.6 CLI wttr.in client

Define an alias to get weather information for my city:

```

1 export WTTRIN_CITY=Orsay
2
3 alias wttrin='curl wttr.in/$WTTRIN_CITY'
4 alias wttrin2='curl v2.wttr.in/$WTTRIN_CITY'

```

### 10.7.7 Minicom

Enable Meta key and colors in minicom:

```

1 export MINICOM='-m -c on'

```

### 10.7.8 Rust

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

```

1 export RUST_SRC_PATH=$HOME/.rustup/toolchains/stable-x86_64-unknown-linux-gnu/lib/rustlib/src/rust/src/
2 export PATH=$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

```

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

```

### 10.7.10 CMake

Add my manually installed libraries to CMake and PATH.

```

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

```

### 10.7.11 Node

Set NPM installation path to local:

```

1 NPM_PACKAGES="${HOME}/.npm-packages"
2
3 # Export NPM bin path
4 export PATH="$PATH:$NPM_PACKAGES/bin"
5
6 # Preserve MANPATH if you already defined it somewhere in your config.
7 # Otherwise, fall back to `manpath` so we can inherit from `/etc/manpath`.
8 export MANPATH="${MANPATH-$(manpath)}:$NPM_PACKAGES/share/man"
9
10 # Tell Node about these packages
11 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`:

```

1 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:

```

1 # If not running inside Emacs (via vterm/eshell...)
2 if [ -z $INSIDE_EMACS ]
3 then
4     if command -v tmux && /dev/null && [ -z "$TMUX" ]
5     then
6         tmux attach -t default || tmux new -s default
7     fi
8 fi

```

### 10.7.13 Other stuff

```

1 # You may need to manually set your language environment
2 # export LANG=en_US.UTF-8
3
4 # Preferred editor for local and remote sessions
5 # if [[ -n $SSH_CONNECTION ]]; then
6 #     export EDITOR='vim'
7 # else
8 #     export EDITOR='mvim'
9 # fi
10
11 # Compilation flags
12 # export ARCHFLAGS="-arch x86_64"
13
14 # FZF
15 [ -f ~/.fzf.zsh ] && source ~/.fzf.zsh
16
17 # OPAM configuration
18 [[ ! -r $HOME/.opam/opam-init/init.zsh ]] || source $HOME/.opam/opam-init/init.zsh > /dev/null 2> /dev/null
19
20 # Add ~/.config/emacs/bin to path (for DOOM Emacs stuff)
21 export PATH=$PATH:$HOME/.config/emacs/bin
22
23 export TEXMFHOME=$HOME/.texmf

```

Define some environment variables.

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

Load my bitwarden-cli session, exported to BW\_SESSION.

```
1 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:

```
1 # Rust edition 2018
2 edition = "2018"
3
4 # Use Unix style newlines, with 2 spaces tabulation.
5 newline_style = "Unix"
6 tab_spaces = 2
7 hard_tabs = false
8
9 # Make one line functions in a single line
10 fn_single_line = true
11
12 # Format strings
13 format_strings = true
14
15 # Increase the max line width
16 max_width = 120
17
18 # Merge nested imports
19 merge_imports = true
20
21 # Enum and Struct alignment
22 enum_discrim_align_threshold = 20
23 struct_field_align_threshold = 20
24
25 # Reorder impl items: type > const > macros > methods.
26 reorder_impl_items = true
27
28 # Comments and documentation formating
29 wrap_comments = true
30 normalize_comments = true
31 normalize_doc_attributes = true
32 format_code_in_doc_comments = true
33 report_fixme = "Always"
34 todo = "Always"
```

## 10.9 eCryptfs

### 10.9.1 Unlock and mount script

```
1 #!/bin/sh -e
2 # This script mounts a user's confidential private folder
3 #
4 # Original by Michael Halcrow, IBM
5 # Extracted to a stand-alone script by Dustin Kirkland <kirkland@ubuntu.com>
6 # Modified by: Abdelhak Bougouffa <abougouffa@fedoraproject.org>
7 #
8 # This script:
9 # * interactively prompts for a user's wrapping passphrase (defaults to their
```

```

10 # login passphrase)
11 # * checks it for validity
12 # * unwraps a users mount passphrase with their supplied wrapping passphrase
13 # * inserts the mount passphrase into the keyring
14 # * and mounts a user's encrypted private folder
15
16 PRIVATE_DIR="Private"
17 PW_ATTEMPTS=3
18 MESSAGE=`gettext "Enter your login passphrase:"`
19
20 if [ -f $HOME/.ecryptfs/wrapping-independent ]
21 then
22     # use a wrapping passphrase different from the login passphrase
23     MESSAGE=`gettext "Enter your wrapping passphrase:"`
24 fi
25
26 WRAPPED_PASSPHRASE_FILE="$HOME/.ecryptfs/wrapped-passphrase"
27 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" - |
48             ↪ ecryptfs-add-passphrase -
49             then
50                 break
51             else
52                 zenity --error --title "eCryptfs" --text "Error: Your passphrase is incorrect"
53                 tries=$((tries + 1))
54                 continue
55             fi
56         else
57             if printf "%s\0" "$LOGINPASS" | ecryptfs-insert-wrapped-passphrase-into-keyring
58             ↪ "$WRAPPED_PASSPHRASE_FILE" -
59             then
60                 break
61             else
62                 zenity --error --title "eCryptfs" --text "Error: Your passphrase is incorrect"
63                 tries=$((tries + 1))
64                 continue
65             fi
66         fi
67     done
68
69     if [ $tries -ge $PW_ATTEMPTS ]
70     then
71         zenity --error --title "eCryptfs" --text "Too many incorrect password attempts, exiting"
72         exit 1
73     fi
74
75     /sbin/mount.ecryptfs_private
76 else
77     zenity --error --title "eCryptfs" --text "Encrypted private directory is not setup properly"
78     exit 1
79 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
6 Exec=/home/hacko/.ecryptfs/ecryptfs-mount-private-gui
7 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`.

```

1 # GDB early init file
2 # Abdelhak Bougouffa (c) 2022
3
4 # Disable showing the initial message
5 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.

```

1 # GDB init file
2 # Abdelhak Bougouffa (c) 2022
3
4 # Save history
5 set history save on
6 set history filename ~/.gdb_history
7 set history remove-duplicates 2048
8
9 # When debugging my apps, debug information of system libraries
10 # aren't that important
11 set debuginfod enabled off
12
13 # Set pretty print
14 set print pretty on
15
16 # I hate stepping into system libraries when I'm debugging my
17 # crappy stuff, so lets add system headers to skipped files
18 skip pending on
19 python
20 import os
21
22 # Add paths here, they will be explored recursively

```



```

23 LIB_PATHS = ["/usr/include" "/usr/local/include"]
24
25 for lib_path in LIB_PATHS:
26     for root, dirs, files in os.walk(lib_path):
27         for file in files:
28             cmd = f"skip file {os.path.join(root, file)}"
29             gdb.execute(cmd, True, to_string=True)
30     end
31     skip enable
32
33     skip pending on
34     guile
35     <<gdb-init-guile>>
36     end
37     skip enable
38
39     # This fixes the annoying ncurses TUI glitches and saves typing C-l each time to refresh the screen
40     define cc
41         continue
42         refresh
43     end
44
45     define nn
46         next
47         refresh
48     end

```

## 10.11 GnuPG

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

```

1 # Do not ask me about entered passwords for 24h (during the same session)
2 default-cache-ttl 86400
3 max-cache-ttl 86400
4
5 # As I'm using KDE, use Qt based pinentry tool instead of default GTK+
6 pinentry-program /usr/bin/pinentry-qt
7
8 # Allow pinentry in Emacs minibuffer (combined with epg-pinentry-mode)
9 allow-loopback-pinentry
10 allow-emacs-pinentry

```

## 10.12 OCR This

```

1 #!/bin/bash
2
3 IMG=$(mktemp -u --suffix=".png")
4 scrot -s "$IMG" -q 100
5 mogrify -modulate 100,0 -resize 400% "$IMG"
6 tesseract "$IMG" -l eng 2> /dev/null | xsel -ib

```

## 10.13 Arch Linux packages

Here, we install Arch packages

```

1 check_and_install_pkg() {
2     PKG_NAME="$1"
3     if ! pacman -Qiq "${PKG_NAME}" &>/dev/null; then
4         echo "Package ${PKG_NAME} is missing, installing it using yay"
5         yay -S "${PKG_NAME}"

```

```

6     fi
7 }
8
9 PKGS_LIST=(
10     git ripgrep fd gnupg fzf the_silver_searcher
11     ttf-ibm-plex ttf-fira-code ttf-roboto-mono ttf-overpass ttf-lato ttf-input
12     ttf-cascadia-code ttf-jetbrains-mono ttf-fantasque-sans-mono
13     ttc-iosevka ttf-iosevka-nerd ttc-iosevka-slab ttc-iosevka-curly
14     ttc-iosevka-curly-slab ttc-iosevka-etoile ttc-iosevka-ss09
15     ccls cppcheck clang gcc gdb lldb valgrind rr openocd
16     sbcl cmucl clisp chez-scheme mit-scheme chibi-scheme chicken
17     vls vlang rustup
18     mu isync msmtp xsel xorg-xhost
19     mpc mpv mpd vlc yt-dlp
20     maxima fricas octave scilab-bin graphviz jupyterlab jupyter-notebook r
21     djvulibre catdoc unrar perl-image-exiftool wkhtmltopdf
22     chezmoi neovim repo ecryptfs-utils
23     pandoc hugo inkscape imagemagick
24     aspell aspell-en aspell-fr aspell-ar grammalecte language-tool ltex-ls-bin
25     libvterm brave zotero bitwarden-cli binutils
26     poppler ffmpegthumbnailer mediainfo imagemagick tar unzip
27 )
28
29 for PKG in "${PKGS_LIST[@]}; do
30     check_and_install_pkg "$PKG"
31 done

```

## 10.14 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.

```

1 export PLASMA_USE_QT_SCALING=1

```