Workflow Configuration

Personal configuration for MinEmacs, Bash, Zsh, and other Linux stuff

Abdelhak Bougouffa*

January 1, 2025

Contents

*(rot13 "nobhtbhssn@srqbencebwrpg.bet")

1 This repository

This repository (abougouffa/dotfiles) contains my configuration files for **Bash**, **Zsh**, **MinEmacs**, **Neovim**, **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...).

1.1 Notice

This is the main configuration file, it contains the literal personal configuration for MinEmacs, and I use it to generate some other Linux configuration files (Bash configuration, define aliases, environment variables, user tools, Git configuration...).

2 Deploying the dotfiles

To deploy these dotfiles, I'm using the simple GNU Stow. The configuration files (including the ones tangled from the `literate-config.org` file) are stored in seperate directories. For example: Bash and Zsh configuration files are stored in `shell/`, GDB configuration files in `gdb/`, Emacs/MinEmacs configuraiton files in `emacs/`, and so on.

To tangle the `literate-config.org` file from the command line, you can run:

```
make tangle
```

To deploy a configuration using GNU Stow, it is as simple as running the following commands:


```
git clone https://github.com/abougouffa/dotfiles.git
cd dotfiles

# To install Emacs configuration
stow -t $HOME emacs

# To install GDB configuration
stow -t $HOME gdb
```

Or, you can run it with `make`:

```
cd dotfiles
make emacs  # runs "stow -t $HOME emacs"
make gdb    # runs "stow -t $HOME gdb"
```

3 Introduction

I've been using Linux exclusively since 2009, **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 had become a real mess. As a new Emacs user, I didn't understand in the beginning how to optimize my configuration and how to do things correctly, which resulted in a configuration that takes more than 30s to load.

I discovered then Spacemacs, which made things much easier, but it was a little slow and it was hiding a lot of Emacs features behind some *simplistic* configuration options, which isn't perfect for someone who loves writing code. Just after, I found the awesome Doom Emacs, which helped me progress and learn more about Emacs and Elisp, but at some point, I faced multiple problems with Doom, so I started my own configuration framework, MinEmacs.

MinEmacs is ~~intended to be~~ a minimal configuration framework. In the beginning, I planned to use only a small set of necessary packages (hence, the *Min* in *MinEmacs*). However, it is starting to grow to respond to my daily needs.

This block will be tangled in the file `bootstrap.sh`, which will be used as a script to conditionally install and configure some system features.

```
#!/bin/env bash

# NOTE: This file is generated from "config-literate.org".
# This shell script has been generated from the litterate Org configuration.
# It helps installing required tools (for Arch/Manjaro Linux) and tweak some
# system settings
```

4 MinEmacs

4.1 MinEmacs user configuration files

4.1.1 Early configuration (`early-config.el`)

```
;;; early-config.el -*- coding: utf-8-unix; lexical-binding: t; -*-

;; NOTE: This file is generated from "config-literate.org".

;; MinEmacs specific stuff
(unless minemacs-verbose-p
  (setq minemacs-msg-level 3))
```

4.1.2 Modules (`modules.el`)

```
;;; modules.el -*- coding: utf-8-unix; lexical-binding: t; -*-

;; NOTE: This file is generated from "config-literate.org".

;; This file can be used to override `minemacs-modules'

(setq
;;; MinEmacs modules
 minemacs-modules (minemacs-modules) ; Enable all available modules

;;; MinEmacs disabled packages (included in one of the modules)
 minemacs-disabled-packages nil)
```

4.1.3 User configuration (`config.el`)

```
;;; config.el -*- coding: utf-8-unix; lexical-binding: t; -*-

;; NOTE: This file is generated from "config-literate.org".
```

4.2 General Emacs settings

4.2.1 User information

```
;; Personal info
(setq user-full-name "Abdelhak Bougouffa"
      user-mail-address (rot13 "nobhtbhssn@srqbencebwrpg.bet"))
```

4.2.2 Crypto stuff

Crypto as in *Cryptography* not *Cryptocurrency*!

```
(setq-default
 epa-file-encrypt-to '("F808A020A3E1AC37")) ; Encrypt files to my self by default
```

4.2.3 Bidirectional settings

This combo should speedup opening files, for right-to-left languages like Arabic, you might need to disable this or set it in a per-buffer basis.

```
(setq-default
 ;; Better support for files with long lines
 bidi-paragraph-direction 'left-to-right
 ;; Speeds redisplay, may break paranthesis rendering for bidirectional files
 bidi-inhibit-bpa t)
```

4.2.4 Directories

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

(setq org-directory (if (file-directory-p "~/Dropbox/Org/") "~/Dropbox/Org/" "~/Documents/Org/")
      source-directory "~/Softwares/aur/emacs-git/src/emacs-git/")
```

4.2.5 Misc

```
(setq +binary-hexl-enable t
      browse-url-chromium-program (or (executable-find "brave")
                                      (executable-find "chromium")
                                      (executable-find "chromium-browser"))
      browse-url-chrome-program browse-url-chromium-program)
```

4.2.6 Awqat

Awqat (Arabic: meaning *Times*), is a package to calculate Islamic prayer times based on the current location.

```
(when (featurep 'me-lifestyle)
  (+with-delayed-1!
   ;; Calendar settings (from `solar')
   (setq calendar-latitude 48.86
         calendar-longitude 2.35
         calendar-location-name "Paris, FR"
         calendar-time-display-form '(24-hours ":" minutes))

   (awqat-display-prayer-time-mode 1)
   (awqat-set-preset-french-muslims)))
```

4.2.7 Projects

Automatically scan for projects under `+project-scan-dir-paths` at startup:

```
(+with-delayed-1!
 (setq +project-scan-dir-paths
       '("~/Research/papers/"
         "~/Research/workspace/"
         "~/Research/workspace-no/"
         "~/Research/workspace-no/ez-wheel/swd-starter-kit-repo/"
         "~/Projects/foss/packages/"
         "~/Projects/foss/repos/"))

 (+shutup!
  (+project-scan-for-projects)))
```

4.3 Package configuration

4.3.1 User interface

Theme & font

```
(setq minemacs-theme 'doom-one-light)
```

4.3.2 Natural languages

Offline dictionaries for `lexic` and `sdcv` The `lexic` package needs `sdcv` to be installed, it needs also some *StarDict* dictionaries to be installed, see `M-x lexic-dictionary-help`. You can download some dictionaries from:

- Dict.org archive
- Big English dictionaries
- French dictionaries

Jinx - Enchanted Spell Checker

```
(with-eval-after-load 'jinx
  (setq-default jinx-languages "en_US fr"))
```

4.4 Applications

4.4.1 News feed (elfeed)

Set RSS news feeds

```
(with-eval-after-load 'elfeed
  (setq elfeed-feeds
        '(("https://arxiv.org/rss/cs.RO" robotics academic)
          ("https://interstices.info/feed" science academic)
          ("https://spectrum.ieee.org/rss/robotics/fulltext" robotics academic)
          ("https://spectrum.ieee.org/rss/aerospace/fulltext" academic aerospace)
          ("https://spectrum.ieee.org/rss/computing/fulltext" academic computing)
          ("https://spectrum.ieee.org/rss/blog/automaton/fulltext" academic automation robotics)
          ("https://www.technologyreview.com/feed" tech science)
          ("https://interrupt.memfault.com/feed.xml" embedded prog rust c cpp)
          ("https://itsfoss.com/feed" linux foss)
          ("https://lwn.net/headlines/rss" linux foss)
          ("https://linuxhandbook.com/feed" linux foss)
          ("https://www.omgubuntu.co.uk/feed" linux foss)
          ("https://this-week-in-rust.org/rss.xml" rust prog)
          ("https://planet.emacslife.com/atom.xml" emacs prog foss)
          ("https://developers.redhat.com/blog/feed" linux foss))))
```

4.4.2 Email (mu4e)

Configuring `mu4e` as email client needs three parts:

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

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 plain text file.

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

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

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

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

```
machine smtp.googlemail.com login username@gmail.com port 587 password YOUR_P455w0rd
```

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 `~/.mbsyncrc` file should look like this:

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

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

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

# SECTION (Maildir Stores)
MaildirStore work-local      # Local storage (create directories with mkdir -p ~/Maildir/<ACCOUNT-NAME>)
Path ~/Maildir/work/         # The local store path
Inbox ~/Maildir/work/Inbox   # Location of the INBOX
SubFolders Verbatim          # Download all sub-folders
# END OF SECTION

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

```
# ============================================================================

IMAPAccount gmail
Host imap.gmail.com
User user@gmail.com
PassCmd "gpg2 -q --for-your-eyes-only --no-tty --logger-file /dev/null --batch -d ~/.authinfo.gpg | awk '/machine
↪  smtp.domain.tld/ {print $NF}'"
AuthMechs LOGIN
SSLType IMAPS
CertificateFile /etc/ssl/certs/ca-certificates.crt

IMAPStore gmail-remote
Account gmail

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

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

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

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

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

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

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

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

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

```
Channel gmail-inbox
Channel gmail-sent
Channel gmail-trash
Channel gmail-drafts
Channel gmail-all
Channel gmail-starred
Channel gmail-spam
# END OF SECTION
```

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

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

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

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

```
# Set default values for all following accounts.
defaults
auth                    on
tls                     on
tls_starttls            on
tls_trust_file          /etc/ssl/certs/ca-certificates.crt
logfile                 ~/.msmtp.log

# Gmail
account                 gmail
auth                    plain
host                    smtp.googlemail.com
port                    587
from                    username@gmail.com
user                    username
passwordeval            "gpg -q --for-your-eyes-only --no-tty --logger-file /dev/null --batch -d ~/.authinfo.gpg |
↪  awk '/machine smtp.googlemail.com login .*@gmail.com/ {print $NF}'"
add_missing_date_header on

## Gmail - aliases
account                 alias-account : gmail
from                    alias@mail.com

account                 other-alias : gmail
from                    other.alias@address.org

# Work
account                 work
auth                    on
host                    smtp.domaine.tld
port                    587
from                    username@domaine.tld
user                    username
passwordeval            "gpg -q --for-your-eyes-only --no-tty --logger-file /dev/null --batch -d ~/.authinfo.gpg |
↪  awk '/machine smtp.domaine.tld/ {print $NF}'"
tls_nocertcheck # ignore TLS certificate errors
```

Mail client and indexer (`mu` and `mu4e`) I configure my email accounts in a private file in `private/mu4e-accounts.el`, which will be loaded after this common part:

```
;; To disable auto starting mu4e in background
(setq +mu4e-auto-start nil)

(with-eval-after-load 'mu4e
  ;; Custom files
  (setq mail-personal-alias-file (concat minemacs-config-dir "private/mail-aliases.mailrc")
        mu4e-icalendar-diary-file (concat org-directory "icalendar-diary.org"))

  ;; Add a unified inbox shortcut
  (add-to-list
   'mu4e-bookmarks
   '(:name "Unified inbox" :query "maildir:/.*inbox/" :key ?i) t)

  ;; Add shortcut to view spam messages
  (add-to-list
   'mu4e-bookmarks
   '(:name "Spams" :query "maildir:/.*\\(spam\\|junk\\).*/" :key ?s) t)

  ;; The `+mu4e-extras-ignore-spams-query' function is defined in
  ;; `me-mu4e-extras'.
  (with-eval-after-load 'me-mu4e-extras
    ;; Add shortcut to view yesterday's messages
    (add-to-list
     'mu4e-bookmarks
     `(:name "Yesterday's messages" :query ,(+mu4e-extras-ignore-spams-query "date:1d..today") :key ?y) t))

  ;; Load my accounts
  (+load minemacs-config-dir "private/mu4e-accounts.el")
  (+load minemacs-config-dir "private/mu4e-extra-commands.el"))
```

The content of `private/mu4e-accounts.el` is something like:

```
;; An address to add automatically as BCC for all sent messages
(setq +mu4e-auto-bcc-address "Auto-BCC <some.address@host.tld>")

;; Register multiple accounts using the `+mu4e-register-account' helper
(+mu4e-register-account
 "Work account"
 "my-work"
 `(;; The main Email address for the account
   (user-mail-address     . "some.name@my-work.tld")
   ;; The folders, as configured in your "~/.mbsyncrc" file
   (mu4e-sent-folder      . "/my-work/Sent")
   (mu4e-drafts-folder    . "/my-work/Drafts")
   (mu4e-trash-folder     . "/my-work/Trash")
   (mu4e-refile-folder    . "/my-work/Archive")
   ;; SMTP server settings (when using the "msmtp" based configuration, these
   ;; settings aren't required)
   (smtpmail-smtp-server  . "smtps.my-work.tld")
   (smtpmail-smtp-service . 587)
   ;; If your Email have alises, put them to this list
   (+mu4e-account-aliases . ("other.address@mywork.tld"))
   ;; When `org-msg' is used, you can define templates for greeting and signature
   (org-msg-greeting-fmt  . "Hello%s,")
   ;; You can use the `+org-msg-make-signature' helper to generate a signature
   (org-msg-signature     . ,(+org-msg-make-signature
                              "Best regards," ; Closing phrase
                              "Firstname"
                              "Lastname"
                              ;; The signatures, multiple lines
                              "/Cool R&D Engineer/"
                              "=+33 01 23 45 67 89="))
   (message-signature     . ,(+org-msg-make-signature
                              "Regards,"
```

```
                              "Firstname"
                              "Lastname"
                              "/Cool R&D Engineer/"
                              "=+33 01 23 45 67 89="))

;; Another account
(+mu4e-register-account
 "Personal account"
 "personal"
 `((user-mail-address     . "me@personal-mail.tld")
   (mu4e-sent-folder      . "/personal/Sent")
   (mu4e-drafts-folder    . "/personal/Drafts")
   (mu4e-trash-folder     . "/personal/Trash")
   (mu4e-refile-folder    . "/personal/Archives")
   (smtpmail-smtp-server  . "smtp.personal-mail.tld")
   (smtpmail-smtp-service . 587)
   (+mu4e-account-aliases . ("me.me@personal-mail.tld"
                             "myname@perso.tld"))
   (org-msg-greeting-fmt  . "Hi%s,")
   (org-msg-signature     . ,(+org-msg-make-signature
                              "Regards,"
                              "My first name"
                              "My last name"))))
```

4.4.3 Calendar

```
(with-eval-after-load 'calfw-ical
  (+load minemacs-config-dir "private/calfw-sources.el"))
```

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

4.4.4 EMPV

```
(with-eval-after-load 'empv
  (setq
   ;; Set the radio channels, you can get streams from https://www.radio-browser.info
   empv-radio-channels
   '(("El-Bahdja FM"          . "http://webradio.tda.dz:8001/ElBahdja_64K.mp3")
     ("El-Chaabia"            . "https://radio-dzair.net/proxy/chaabia?mp=/stream")
     ("Quran Radio"           . "http://stream.radiojar.com/0tpy1h0kxtzuv")
     ("Algeria International" . "https://webradio.tda.dz/Internationale_64K.mp3")
     ("JOW Radio"             . "https://str0.creacast.com/jowradio")
     ("France Iter"           . "http://direct.franceinter.fr/live/franceinter-hifi.aac")
     ("France Info"           . "http://direct.franceinfo.fr/live/franceinfo-hifi.aac")
     ("France Culture"        . "http://icecast.radiofrance.fr/franceculture-hifi.aac")
     ("France Musique"        . "http://icecast.radiofrance.fr/francemusique-hifi.aac")
     ("FIP"                   . "http://icecast.radiofrance.fr/fip-hifi.aac")
     ("Beur FM"               . "http://broadcast.infomaniak.ch/beurfm-high.aac")
     ("Skyrock"               . "http://icecast.skyrock.net/s/natio_mp3_128k"))))
```

4.5 Programming

4.5.1 Tramp

```
(with-eval-after-load 'tramp
  (setq
   ;; Do not use a separate history file for tramp sessions (buggy!)
   tramp-histfile-override nil
   ;; Use Bash as a default remote shell
   tramp-default-remote-shell "/bin/bash"
   ;; Use bash for encoding and decoding commands on the local host
   tramp-encoding-shell "/bin/bash"))
```

4.5.2 Robot Operating System (ROS)

```
(with-eval-after-load 'ros
  (setq ros-workspaces
        (list
         (ros-dump-workspace
          :tramp-prefix "/docker:ros@ros-machine:"
          :workspace "~/ros_ws"
          :extends '("/opt/ros/noetic/"))
         (ros-dump-workspace
          :tramp-prefix "/docker:ros@ros-machine:"
          :workspace "~/ros2_ws"
          :extends '("/opt/ros/foxy/")))))
```

4.6 Office

4.6.1 Org mode

Org mode tweaks

```
(with-eval-after-load 'org
  (setq
   ;; Let's put our Org files here
   org-directory "~/Dropbox/Org/"
   ;; Do not ask before tangling
   org-confirm-babel-evaluate nil
   ;; The last level which is still exported as a headline
   org-export-headline-levels 5
   ;; Default file for notes (for org-capture)
   org-default-notes-file (concat org-directory "inbox.org")
   +org-inbox-file (concat org-directory "inbox.org")
   +org-projects-file (concat org-directory "projects.org")
   ;; Custom todo keyword faces
   org-todo-keyword-faces
   '(("IDEA" . (:foreground "goldenrod" :weight bold))
     ("NEXT" . (:foreground "IndianRed1" :weight bold))
     ("STRT" . (:foreground "OrangeRed" :weight bold))
     ("WAIT" . (:foreground "coral" :weight bold))
     ("KILL" . (:foreground "DarkGreen" :weight bold))
     ("PROJ" . (:foreground "LimeGreen" :weight bold))
     ("HOLD" . (:foreground "orange" :weight bold)))
   ;; Custom org-capture templates, see: https://orgmode.org/manual/Capture.html
   org-capture-templates
   `(("t" "Todo" entry (file+headline ,+org-inbox-file "Inbox")
      "* TODO %?\n%i\n%a")
     ("i" "Idea" entry (file+headline ,+org-inbox-file "Ideas")
      "* IDEA %?\n%T\n%i\n%a")
     ("p" "Project note" entry (file ,+org-projects-file)
      "* %?\n%T\n%i\n%a")
     ("n" "Note" entry (file+headline ,+org-inbox-file "Notes")
      "* NOTE %?\n%T\n%i\n%a")))

  (setq org-tag-persistent-alist
        '((:startgroup . nil)
          ("home"      . ?h)
          ("research"  . ?r)
          ("work"      . ?w)
          (:endgroup   . nil)
          (:startgroup . nil)
          ("tool"      . ?o)
          ("dev"       . ?d)
          ("report"    . ?p)
          (:endgroup   . nil)
          (:startgroup . nil)
```

```
        ("easy"      . ?e)
        ("medium"    . ?m)
        ("hard"      . ?a)
        (:endgroup   . nil)
        ("urgent"    . ?u)
        ("key"       . ?k)
        ("bonus"     . ?b)
        ("ignore"    . ?i)
        ("noexport"  . ?x)))

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

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

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

(setq org-agenda-files (list +org-inbox-file
                             +org-projects-file
                             (concat org-directory "gcal-agenda.org"))
      org-agenda-deadline-faces
      '((1.001 . error)
        (1.000 . org-warning)
        (0.500 . org-upcoming-deadline)
        (0.000 . org-upcoming-distant-deadline))
      org-list-demote-modify-bullet
      '(("+"  . "-")
        ("-"  . "+")
        ("*"  . "+")
        ("1." . "a.")))

;; Needs to make a src_latex{\textsc{text}}?, with this hack you can write [[latex:textsc][Some text]].
(+shutup!
 (org-add-link-type
  "latex" nil
  (lambda (path desc format)
    (cond
     ((eq format 'html)
      (format "<span class=\"%s\">%s</span>" path desc))
     ((eq format 'latex)
      (format "{\\%s{%s}}" path desc))))))

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

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

Org + Hugo

```
(with-eval-after-load 'ox-hugo
  (setq org-hugo-auto-set-lastmod t))
```

Org + LaTeX

```
(with-eval-after-load 'ox-latex
  (add-to-list 'org-latex-packages-alist '("svgnames" "xcolor")))
```

Denote

```
(setq denote-directory "~/Dropbox/Org/notes/")

(with-eval-after-load 'recentf
  (add-to-list 'recentf-exclude denote-directory))
```

Bibliography

Org-cite

```
(with-eval-after-load 'oc
  (setq org-cite-csl-styles-dir +biblio-styles-path
        org-cite-global-bibliography (ensure-list +biblio-libraries-path)))
```

Citar

```
(with-eval-after-load 'citar
  (setq citar-library-paths (ensure-list +biblio-storage-path)
        citar-notes-paths (ensure-list +biblio-notes-path)
        citar-bibliography (ensure-list +biblio-libraries-path)))
```

5 System configuration

5.1 Mime types

5.1.1 Org mode files

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

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

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

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

Then set Emacs as the default editor:

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

5.1.2 Registering `org-protocol://`

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

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

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

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

5.1.3 Configuring Chrome/Brave

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

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

if [[ "$INSTALL_CONFIRM" == "Y" ]]
then
  sudo mkdir -p /etc/opt/chrome/policies/managed/

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

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

Then add a bookmarklet in your browser with this code:

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

6 Command line tools

6.1 Emacs command-line wrapper

A wrapper around `emacsclient`, adapted from tecosaur's Emacs configuration.

- 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`).

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

args=()

while :; do
    case "$1" in
    -t | -nw | --tty)
        force_tty=true
        shift
        ;;
    -w | --wait)
        force_wait=true
        shift
        ;;
    -m | --mode)
        stdin_mode=" ($2-mode)"
        shift 2
        ;;
    -h | --help)
        echo -e "\033[1mUsage: e [-t] [-m MODE] [OPTIONS] FILE [-]\033[0m

Emacs client convenience wrapper.

\033[1mOptions:\033[0m
\033[0;34m-h, --help\033[0m            Show this message
\033[0;34m-t, -nw, --tty\033[0m        Force terminal mode
\033[0;34m-w, --wait\033[0m            Don't supply \033[0;34m--no-wait\033[0m to graphical emacsclient
\033[0;34m-\033[0m                     Take \033[0;33mstdin\033[0m (when last argument)
\033[0;34m-m MODE, --mode MODE\033[0m  Mode to open \033[0;33mstdin\033[0m with

Run \033[0;32memacsclient --help\033[0m to see help for the emacsclient."
        exit 0
        ;;
    --*=*)
        set -- "$@" "${1%%=*}" "${1#*=}"
        shift
        ;;
    *)
        if [ "$#" = 0 ]; then
            break
        fi
        args+=("$1")
        shift
        ;;
    esac
done

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

if [ -z "$DISPLAY" ] || $force_tty; then
    # detect terminals with sneaky 24-bit support
    if { [ "$COLORTERM" = truecolor ] || [ "$COLORTERM" = 24bit ]; } &&
```

```
        [ "$(tput colors 2>/dev/null)" -lt 257 ]; then
        if echo "$TERM" | grep -q "^\w\+-[0-9]"; then
            termstub="${TERM%%-*}"
        else
            termstub="${TERM#*-}"
        fi
        if infocmp "$termstub-direct" >/dev/null 2>&1; then
            TERM="$termstub-direct"
        else
            TERM="xterm-direct"
        fi # should be fairly safe
    fi
    emacsclient --tty -create-frame --alternate-editor="$ALTERNATE_EDITOR" "${args[@]}"
else
    if ! $force_wait; then
        args+=(--no-wait)
    fi
    emacsclient -create-frame --alternate-editor="$ALTERNATE_EDITOR" "${args[@]}"
fi
```

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

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

# Aliases to run emacs+magit
alias magit='e --eval "(progn (magit-status) (delete-other-windows))"'
alias magitt='e -t --eval "(progn (magit-status) (delete-other-windows))"'

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

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

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

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

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

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

6.2 AppImage

Install/update the `appimageupdatetool.AppImage` tool:

```
update_appimageupdatetool () {
  TOOL_NAME=appimageupdatetool
  MACHINE_ARCH=$(uname -m)
  APPIMAGE_UPDATE_TOOL_PATH="$HOME/.local/bin/${TOOL_NAME}"

  ↪  APPIMAGE_UPDATE_TOOL_URL="https://github.com/AppImage/AppImageUpdate/releases/download/continuous/${TOOL_NAME}-${MACHINE_ARC

  if [ -f "${APPIMAGE_UPDATE_TOOL_PATH}" ] && "$APPIMAGE_UPDATE_TOOL_PATH" -j "${APPIMAGE_UPDATE_TOOL_PATH}"
  ↪  2&>/dev/null; then
    echo "${TOOL_NAME} already up to date"
  else
```

```
    if [ -f "${APPIMAGE_UPDATE_TOOL_PATH}" ]; then
      echo "Update available, downloading latest ${MACHINE_ARCH} version to ${APPIMAGE_UPDATE_TOOL_PATH}"
      mv "${APPIMAGE_UPDATE_TOOL_PATH}" "${APPIMAGE_UPDATE_TOOL_PATH}.backup"
    else
      echo "${TOOL_NAME} not found, downloading latest ${MACHINE_ARCH} version to ${APPIMAGE_UPDATE_TOOL_PATH}"
    fi
    wget -O "${APPIMAGE_UPDATE_TOOL_PATH}" "${APPIMAGE_UPDATE_TOOL_URL}" && # 2&>/dev/null
        echo "Downloaded ${TOOL_NAME}-${MACHINE_ARCH}.AppImage" &&
        [ -f "${APPIMAGE_UPDATE_TOOL_PATH}.backup" ] &&
        rm "${APPIMAGE_UPDATE_TOOL_PATH}.backup"
    chmod a+x "${APPIMAGE_UPDATE_TOOL_PATH}"
  fi
}

update_appimageupdatetool
```

7 Oh-my-Zsh

7.1 Installation

Follow the installation procedure from `https://github.com/ohmyzsh/ohmyzsh`.

```
if [ ! -d "$HOME/.oh-my-zsh" ]; then
  unset INSTALL_CONFIRM
  read -p "Do you want install Oh-my-Zsh [Y | N]: " INSTALL_CONFIRM
  if [[ "$INSTALL_CONFIRM" =~ "^[Yy]$" ]]; then
    sh -c "$(curl -fsSL https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
  fi
fi
```

7.2 Path

Path to your oh-my-zsh installation.

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

# When logging via Tramp it will look for patterns to detect if a shell is
# present. Fancy shell prompts aren't taken into account.
if [[ $TERM == "dumb" ]]; then
    unsetopt zle
    PS1='$ '
    return
fi

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

7.3 Include extra configuration file

```
# Source extra commands
source "$HOME/.shell_extras"
```

7.4 Themes and customization

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

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

ZSH_THEME="typewritten/typewritten"

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

7.5 Behavior

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

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

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

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

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

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

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

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

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

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

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

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

7.6 Plugins

```
# Would you like to use another custom folder than $ZSH/custom?
ZSH_CUSTOM=$HOME/.config/my_ohmyzsh_customizations

# Which plugins would you like to load?
# Standard plugins can be found in $ZSH/plugins/
# Custom plugins may be added to $ZSH_CUSTOM/plugins/
# Example format: plugins=(rails git textmate ruby lighthouse)
# Add wisely, as too many plugins slow down shell startup.
plugins=(
  zsh-autosuggestions
  zsh-navigation-tools
  zsh-interactive-cd
  archlinux
  ssh-agent
  sudo
  docker
  systemd
  tmux
  python
  pip
  rust
  repo
  cp
  rsync
  z
)
```

7.7 Bootstrap Oh-my-Zsh

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

7.8 Aliases

```
# Aliases
alias zshconfig="vim $HOME/.zshrc"
alias ohmyzsh="ranger $ZSH"
```

8 Oh-my-Bash

8.1 Installation

Follow the installation procedure from `https://github.com/ohmybash/oh-my-bash`.

```
if [ ! -d "$HOME/.oh-my-bash" ]; then
  unset INSTALL_CONFIRM
  read -p "Do you want install Oh-my-Bash [Y | N]: " INSTALL_CONFIRM
  if [[ "$INSTALL_CONFIRM" =~ "^[Yy]$" ]]; then
    bash -c "$(curl -fsSL https://raw.githubusercontent.com/ohmybash/oh-my-bash/master/tools/install.sh)"
  fi
fi
```

8.2 Path

Path to your oh-my-bash installation.

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

# When logging via Tramp it will look for patterns to detect if a shell is
# present. Fancy shell prompts aren't taken into account.
if [[ $TERM == "dumb" ]]; then
    unsetopt zle
    PS1='$ '
    return
fi

# Enable the subsequent settings only in interactive sessions
case $- in
   *i*) ;;
     *) return;;
esac

# Path to your oh-my-bash installation.
export OSH="$HOME/.oh-my-bash"
```

8.3 Include extra configuration file

```
# Source extra commands
source "$HOME/.shell_extras"
```

8.4 Themes and customization

Set name of the theme to load. Optionally, if you set this to `"random"` it'll load a random theme each time that oh-my-bash is loaded.

```
OSH_THEME="font"
```

8.5 Behavior

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

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

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

# Uncomment the following line to change how often to auto-update (in days).
# export UPDATE_OSH_DAYS=13

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

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

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

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

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

```
# DISABLE_UNTRACKED_FILES_DIRTY="true"

# Uncomment the following line if you don't want the repository to be considered dirty
# if there are untracked files.
# SCM_GIT_DISABLE_UNTRACKED_DIRTY="true"

# Uncomment the following line if you want to completely ignore the presence
# of untracked files in the repository.
# SCM_GIT_IGNORE_UNTRACKED="true"

# Uncomment the following line if you want to change the command execution time
# stamp shown in the history command output.  One of the following values can
# be used to specify the timestamp format.
# * 'mm/dd/yyyy'     # mm/dd/yyyy + time
# * 'dd.mm.yyyy'     # dd.mm.yyyy + time
# * 'yyyy-mm-dd'     # yyyy-mm-dd + time
# * '[mm/dd/yyyy]'   # [mm/dd/yyyy] + [time] with colors
# * '[dd.mm.yyyy]'   # [dd.mm.yyyy] + [time] with colors
# * '[yyyy-mm-dd]'   # [yyyy-mm-dd] + [time] with colors
# If not set, the default value is 'yyyy-mm-dd'.
# HIST_STAMPS='yyyy-mm-dd'

# Uncomment the following line if you do not want OMB to overwrite the existing
# aliases by the default OMB aliases defined in lib/*.sh
# OMB_DEFAULT_ALIASES="check"

# Would you like to use another custom folder than $OSH/custom?
# OSH_CUSTOM=/path/to/new-custom-folder

# To disable the uses of "sudo" by oh-my-bash, please set "false" to
# this variable.  The default behavior for the empty value is "true".
OMB_USE_SUDO=true

# To enable/disable display of Python virtualenv and condaenv
# OMB_PROMPT_SHOW_PYTHON_VENV=true  # enable
# OMB_PROMPT_SHOW_PYTHON_VENV=false # disable
```

8.6 Plugins

```
# Which completions would you like to load? (completions can be found in ~/.oh-my-bash/completions/*)
# Custom completions may be added to ~/.oh-my-bash/custom/completions/
# Example format: completions=(ssh git bundler gem pip pip3)
# Add wisely, as too many completions slow down shell startup.
completions=(
  system
  git
  composer
  ssh
  nvm
  npm
  virtualbox
  conda
  makefile
  pip
  pip3
  tmux
)

# Which aliases would you like to load? (aliases can be found in ~/.oh-my-bash/aliases/*)
# Custom aliases may be added to ~/.oh-my-bash/custom/aliases/
# Example format: aliases=(vagrant composer git-avh)
# Add wisely, as too many aliases slow down shell startup.
aliases=(
```

```
  general
)

# Which plugins would you like to load? (plugins can be found in ~/.oh-my-bash/plugins/*)
# Custom plugins may be added to ~/.oh-my-bash/custom/plugins/
# Example format: plugins=(rails git textmate ruby lighthouse)
# Add wisely, as too many plugins slow down shell startup.
plugins=(
  git
  sudo
  bu
  pyenv
  colored-man-pages
)
```

8.7 Bootstrap Oh-my-Bash

```
source "$OSH/oh-my-bash.sh"
```

9 Git

9.1 Defaults

```
[pull]
  rebase = true

[init]
  defaultBranch = main # there is no master!

[commit]
  gpgsign = true

[format]
  signoff = true

[color]
  ui = auto

[filter "lfs"]
  clean = git-lfs clean -- %f
  smudge = git-lfs smudge -- %f
  process = git-lfs filter-process
  required = true

# Store passwords in-memory, ask every 24h (default 900s)
[credential]
  helper = cache --timeout 86400

[url "ssh://git@github.com/"]
  pushInsteadOf = https://github.com/
```

9.2 Git diffs

Based on this gist and this article.

```
*.tex                          diff=tex
*.bib                          diff=bibtex
*.{c,h,c++,h++,cc,hh,cpp,hpp} diff=cpp
*.m                            diff=matlab
*.py                           diff=python
```

```
*.rb                        diff=ruby
*.php                       diff=php
*.pl                        diff=perl
*.{html,xhtml}              diff=html
*.f                         diff=fortran
*.{el,lisp,scm,clj,cljc}    diff=lisp
*.r                         diff=rlang
*.texi*                     diff=texinfo
*.org                       diff=org
*.rs                        diff=rust

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

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

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

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

Then adding some regular expressions for it to `~/.config/git/config`:

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

[diff "lisp"]
# https://protesilaos.com/codelog/2021-01-26-git-diff-hunk-elisp-org
  xfuncname = "^(((;;;+ )|\\(|([ \t]+\\(((cl-|el-patch-)?def(un|var|macro|method|custom)|gb/))).*)$"

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

[diff "texinfo"]
# Taken from:
↪  git.savannah.gnu.org/gitweb/?p=coreutils.git;a=blob;f=.gitattributes;h=c3b2926c78c939d94358cc63d051a70d38cfea5d;hb=HEAD
  xfuncname = "^@node[ \t][ \t]*\\([^,][^,]*\\)"

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

# ====== BINARY FORMATS ======
```

And some tools to view diffs on binary files:

```
[diff "pdf"]
  binary = true
  textconv = sh -c 'pdftotext -layout "$0" -enc UTF-8 -nopgbrk -q -'
  cachetextconv = true
```

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

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

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

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

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

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

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

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

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

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

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

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

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

9.3 Extensions & aliases

9.3.1 `git prune-files`

This tool is taken from `https://gist.github.com/nottrobin/5758221`. It is attributed to David Underhill. It is a script to permanently delete files/folders from a Git repository. To use it, cd to your repository's root, then run the script with a list of paths you want to delete, e.g., `git-prune-files path1 path2`

```
#!/usr/bin/env bash
set -o errexit

if [ $# -eq 0 ]; then
    echo "Usage: git prune-files <path1> [<path2> ...]" 1>&2
    exit 0
fi

# make sure we're at the root of git repo
if [ ! -d .git ]; then
    echo "Error: must run this script from the root of a git repository" 1>&2
    exit 1
fi

# remove all paths passed as arguments from the history of the repo
files=$@
git filter-branch --index-filter "git rm -rf --cached --ignore-unmatch $files" HEAD

# remove the temporary history git-filter-branch otherwise leaves behind for a long time
rm -rf .git/refs/original/ && git reflog expire --all && git gc --aggressive --prune
```

Or, add an alias to the command to your Git configuration:

```
[alias]
  prune-files = "git-prune-files"
```

9.3.2 `git update-commiter`

WIP

```
#!/usr/bin/env bash

if [[ ! $# -eq 2 ]]; then
    echo "Usage: git replace-email old@email.local new@email.com <new-name>"
fi

OLD_EMAIL="$1"
NEW_EMAIL="$2"
```

```
git filter-branch --env-filter '
OLD_EMAIL="oldEmail@xxx-MacBook-Pro.local"
CORRECT_NAME="yourName"
CORRECT_EMAIL="yourEmail"

if [ "$GIT_COMMITTER_EMAIL" = "$OLD_EMAIL" ]
then
    export GIT_COMMITTER_NAME="$CORRECT_NAME"
    export GIT_COMMITTER_EMAIL="$CORRECT_EMAIL"
fi
if [ "$GIT_AUTHOR_EMAIL" = "$OLD_EMAIL" ]
then
    export GIT_AUTHOR_NAME="$CORRECT_NAME"
    export GIT_AUTHOR_EMAIL="$CORRECT_EMAIL"
fi
' --tag-name-filter cat -- --branches --tags
```

9.3.3 `git ignore`

Let's have `git ignore <lang>` to automatically generate `.gitignore` patterns for a specific language.

```
[alias]
  ignore = "!gi() { curl -sL https://www.gitignore.io/api/$@ ;}; gi"
```

9.4 Apache Tika App wrapper

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

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

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

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

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

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

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

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

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

  command -v pandoc >/dev/null || echo "Cannot check, pandoc is missing!"; return

  # Get the lastest version
  TIKA_VERSION=$(
    curl -s "${TIKA_BASE_URL}" | # Get the page
    pandoc -f html -t plain | # Convert HTML page to plain text.
    awk '/([0-9]+\.)+[0-1]\// {print substr($1, 0, length($1)-1)}' | # Get the versions directories (pattern:
    ↪  X.X.X/)
```

```
    sort -rV | # Sort versions, the newest first
    head -n 1 # Get the first (newest) version
  )

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

  echo "Lastest version is ${TIKA_VERSION}"

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

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

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

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

update_apache_tika
```

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

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

10 Shell configuration

10.1 Private/Machine-specific configuration

Load private configuration when found:

```
[[ -f "$HOME/.shell_private" ]] && source "$HOME/.shell_private"
```

10.2 Misc

```
# User configuration
# export MANPATH="/usr/local/man:$MANPATH"

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

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

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

# ssh
# export SSH_KEY_PATH="~/.ssh/rsa_id"

# Set personal aliases, overriding those provided by oh-my-bash libs,
# plugins, and themes. Aliases can be placed here, though oh-my-bash
# users are encouraged to define aliases within the OSH_CUSTOM folder.
# For a full list of active aliases, run `alias`.
#
# Example aliases
# alias bashconfig="mate ~/.bashrc"
# alias ohmybash="mate ~/.oh-my-bash"
```

10.3 `pbcopy` and `pbpaste`

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

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

10.4 `netpaste`

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

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

10.5 Sudo GUI!

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

```
# To run GUI apps from terminal with root permissions
if command -v xhost &> /dev/null; then
```

```
  alias gsuon='xhost si:localuser:root'
  alias gsuoff='xhost -si:localuser:root'
fi
```

10.6 Neovim

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

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

10.7 ESP-IDF

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

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

10.8 CLI wttrin client

Define an alias to get weather information for my city:

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

10.9 Minicom

Enable Meta key and colors in `minicom`:

```
export MINICOM='--metakey --color=on'
```

10.10 Rust

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

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

10.11 Go

I don't like Go creating stuff in my home directory, lets move the Go workspace by setting the `GOPATH` environment variable.

```
# The default is $HOME/go
export GOPATH="$HOME/Projects/go"
```

10.12 Clang-format

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.

```
[[ -d "/opt/clang-format-static" ]] && export PATH="$PATH:/opt/clang-format-static"
```

10.13 CMake

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

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

10.14 Node

Set NPM installation path to local:

```
if ! command -v nvm >/dev/null; then
  unset INSTALL_CONFIRM
  read -p "Do you want install nvm [Y | N]: " INSTALL_CONFIRM
  if [[ "$INSTALL_CONFIRM" =~ "^[Yy]$" ]]; then
    curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh | bash
  fi
fi
```

```
export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"  # This loads nvm bash_completion
```

10.15 Python

Install and setup `pyenv`:

```
if ! command -v pyenv &>/dev/null; then
  unset INSTALL_CONFIRM
  read -p "Do you want install pyenv [Y | N]: " INSTALL_CONFIRM
  if [[ "$INSTALL_CONFIRM" =~ "^[Yy]$" ]]; then
    curl https://pyenv.run | bash
  fi
fi
```

```
export PYENV_ROOT="$HOME/.pyenv"
export PATH="$PYENV_ROOT/bin:$PATH"
command -v pyenv &>/dev/null && eval "$(pyenv init -)"
```

10.16 tmux

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

```
# If not running inside Emacs (via vterm/eshell...)
if [[ -z "${INSIDE_EMACS}" && ( "${DISPLAY}" || "${SSH_TTY}" ) ]]; then
  if command -v tmux &> /dev/null && [[ -z "${TMUX}" ]]; then
    tmux attach -t default || tmux new -s default
  fi
fi
```

10.17 FZF

For Bash, tangled in `~/.bashrc`:

```
command -v fzf &> /dev/null && eval "$(eval fzf --bash)"
```

For Zsh, tangled in `~/.zshrc`:

```
command -v fzf &> /dev/null && eval "$(eval fzf --zsh)"
```

10.18 Nix & Guix

I'm currently experimenting both Nix and Guix as package managers to see which one fits my needs. I'm planning to move from Manjaro/Arch Linux to NixOS or Guix. Guix seems more robust, less hacky than NixOS and uses Guile Scheme for almost every aspect (the Nix language is definitely less powerful and non general purpose as Guile). However, the strict "libre" policy of Guix bothers me a bit, as well as the number of available packages for Guix compared to those available for Nix.

10.18.1 Nix

```
# if ! command -v nix &> /dev/null; then
#   unset INSTALL_CONFIRM
#   read -p "Do you want install Nix for all users? [Y | N]: " INSTALL_CONFIRM

#   if [[ "$INSTALL_CONFIRM" == "Y" ]]; then
#     sh <(curl -L https://nixos.org/nix/install) --daemon
#   else
#     read -p "Do you want install Nix for the current user only? [Y | N]: " INSTALL_CONFIRM

#     if [[ "$INSTALL_CONFIRM" == "Y" ]]; then
#       sh <(curl -L https://nixos.org/nix/install) --no-daemon
#     fi
#   fi
# fi
```

10.18.2 Guix

Create an installer in the `bootstrap.sh` script:

```
# if ! command -v guix &>/dev/null; then
#   unset INSTALL_CONFIRM
#   read -p "Do you want install guix [Y | N]: " INSTALL_CONFIRM
#   if [[ "$INSTALL_CONFIRM" =~ "^[Yy]$" ]]; then
#     curl -sfL https://git.savannah.gnu.org/cgit/guix.git/plain/etc/guix-install.sh | sudo bash
#   fi
# fi
```

For Bash, tangled in `~/.bashrc`:

```
# # Automatically added by the Guix install script.
# if [ -n "$GUIX_ENVIRONMENT" ]; then
#     if [[ $PS1 =~ (.*)"\\$" ]]; then
#         PS1="${BASH_REMATCH[1]} [env]\\\$ "
#     fi
# fi
```

For Zsh, tangled in `~/.zshrc`:

```
# # Automatically added by the Guix install script.
# if [ -n "$GUIX_ENVIRONMENT" ]; then
```

```
#     if [[ $PS1 =~ (.*)"\\$" ]]; then
#         PS1="${BASH_REMATCH[1]} [env]\\\$ "
#     fi
# fi
```

10.19 Jujutsu

Add auto-completion for `jj`. For Bash, tangled in `~/.bashrc`:

```
if command -v jj &> /dev/null; then
    source <(jj util completion bash)
fi
```

For Zsh, tangled in `~/.zshrc`:

```
autoload -Uz compinit
compinit

if command -v jj &> /dev/null; then
    source <(jj util completion zsh)
fi
```

10.20 Direnv

Create an installer in the `bootstrap.sh` script:

```
if ! command -v direnv &>/dev/null; then
  unset INSTALL_CONFIRM
  read -p "Do you want install direnv [Y | N]: " INSTALL_CONFIRM
  if [[ "$INSTALL_CONFIRM" =~ "^[Yy]$" ]]; then
    curl -sfL https://direnv.net/install.sh | bash
  fi
fi
```

For Bash, tangled in `~/.bashrc`:

```
command -v direnv &>/dev/null && eval "$(direnv hook bash)"
```

For Zsh, tangled in `~/.zshrc`:

```
command -v direnv &>/dev/null && eval "$(direnv hook zsh)"
```

10.21 Other stuff

Define some environment variables.

```
export PATH="${PATH}:${HOME}/.local/bin"
```

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

```
[[ -f "$HOME/.bitwarden-session" ]] && source "$HOME/.bitwarden-session"
```

11 Rust format

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

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

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

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

# Format strings
format_strings = true

# Increase the max line width
max_width = 120

# Merge nested imports
merge_imports = true

# Enum and Struct alignement
enum_discrim_align_threshold = 20
struct_field_align_threshold = 20

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

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

12 eCryptfs

12.1 Unlock and mount script

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

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

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

WRAPPED_PASSPHRASE_FILE="$HOME/.ecryptfs/wrapped-passphrase"
```

```
MOUNT_PASSPHRASE_SIG_FILE="$HOME/.ecryptfs/$PRIVATE_DIR.sig"

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

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

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

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

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

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

xdg-open "$HOME/Private"
exit 0
```

12.2 Desktop integration

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

13 GDB

13.1 Early init

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

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

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

13.2 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 standard output.

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

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

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

# Set pretty print
set print pretty on

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

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

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

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

define nn
  next
  refresh
end
```

14 GnuPG

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

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

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

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

15 OCR This

This creates a script named `ocrthis` that can be bound to OS shortcut. When triggered:

- it shows a selection tool to take a partial screenshot,
- it runs `tesseract` to extract the text,
- and then, it copies it to clipboard.

```
#!/bin/bash

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

16 Slack

This script is called at system startup.

```
#!/bin/bash

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

if [[ "$WEEK_DAY" != "6" ]] && [[ "$WEEK_DAY" != "7" ]] && [[ "$HOUR" -gt 7 ]] && [[ "$HOUR" -lt 18 ]]; then
  $SLACK -u %U
else
  echo "It is not work time!"
fi
```

17 Arch Linux packages

Here, we install Arch packages:

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

```
}

PKGS_LIST=(
    # System tools
    git repo ripgrep fd gnupg fzf the_silver_searcher xsel xorg-xhost
    neovim ecryptfs-utils libvterm binutils
    # Fonts
    ttf-martian-mono ttf-ttf-ibm-plex ttf-fira-code ttf-roboto-mono ttf-overpass
    ttf-lato ttf-input ttf-cascadia-code ttf-jetbrains-mono
    ttf-fantasque-sans-mono ttc-iosevka ttc-iosevka-slab ttc-iosevka-curly
    ttc-iosevka-curly-slab
    # Programming tools
    ccls cppcheck clang gcc gcc-m2 gcc-rust gdb lldb valgrind rr openocd vls vlang
    rustup semgrep-bin pylyzer-git zig
    # Lisp/Scheme
    sbcl cmucl clisp chez-scheme mit-scheme chibi-scheme chicken gcl
    # Math
    maxima octave scilab-bin graphviz jupyterlab jupyter-notebook r
    # Media
    mpc mpv mpd vlc yt-dlp poppler ffmpegthumbnailer mediainfo imagemagick
    # Email
    mu isync msmtp
    # Documents
    djvulibre catdoc unrtf perl-image-exiftool wkhtmltopdf pandoc hugo inkscape
    imagemagick
    # Natural languages
    aspell aspell-en aspell-fr aspell-ar grammalecte language-tool ltex-ls-bin sdcv
    # Apps
    brave zotero
)

if command -v pacman >/dev/null; then
    unset INSTALL_CONFIRM
    read -p "Do you want to Arch Linux packages? [Y | N]: " INSTALL_CONFIRM
    if [[ "$INSTALL_CONFIRM" =~ "^[Yy]$" ]]; then
        for PKG in "${PKGS_LIST[@]}"; do
            check_and_install_pkg "$PKG"
        done
    fi
else
    echo "Not on Arch Linux or Manjaro"
fi
```

18 KDE Plasma

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

```
export PLASMA_USE_QT_SCALING=1
```