Emacs config

Egidius Mysliwietz

February 10, 2023

Contents

1 Config

```
;;; $DOOMDIR/config.el -*- lexical-binding: t; -*-

;; Place your private configuration here! Remember, you do not need to run
↪  'doom
;; sync' after modifying this file!


;; Some functionality uses this to identify you, e.g. GPG configuration, email
;; clients, file templates and snippets.
(setq user-full-name "Egidius Mysliwietz"
      user-mail-address "egidius@mysliwietz.de"
      auth-sources '("~/.authinfo")
      auth-source-cache-expiry nil)

;; Doom exposes five (optional) variables for controlling fonts in Doom. Here
;; are the three important ones:
;;
;; + `doom-font'
;; + `doom-variable-pitch-font'
;; + `doom-big-font' -- used for `doom-big-font-mode'; use this for
;;   presentations or streaming.
;;
;; They all accept either a font-spec, font string ("Input Mono-12"), or xlfd
;; font string. You generally only need these two:
;; (setq doom-font (font-spec :family "monospace" :size 12 :weight 'semi-light)
;;       doom-variable-pitch-font (font-spec :family "sans" :size 13))

;; There are two ways to load a theme. Both assume the theme is installed and
;; available. You can either set `doom-theme' or manually load a theme with the
;; `load-theme' function. This is the default:
(setq doom-theme 'doom-one)
(custom-set-faces!
  '(doom-modeline-buffer-modified :foreground "orange")
  '(org-cite :foreground "purple")
  '(org-cite-key :foreground "MediumPurple1" :slant italic))


;; If you use `org' and don't want your org files in the default location
↪  below,
;; change `org-directory'. It must be set before org loads!
(setq org-directory "~/org/")
```

```
;; This determines the style of line numbers in effect. If set to `nil', line
;; numbers are disabled. For relative line numbers, set this to `relative'.
(setq display-line-numbers-type t)


;; Here are some additional functions/macros that could help you configure
↪  Doom:
;;
;; - `load!' for loading external *.el files relative to this one
;; - `use-package!' for configuring packages
;; - `after!' for running code after a package has loaded
;; - `add-load-path!' for adding directories to the `load-path', relative to
;;   this file. Emacs searches the `load-path' when you load packages with
;;   `require' or `use-package'.
;; - `map!' for binding new keys
;;
;; To get information about any of these functions/macros, move the cursor over
;; the highlighted symbol at press 'K' (non-evil users must press 'C-c c k').
;; This will open documentation for it, including demos of how they are used.
;;
;; You can also try 'gd' (or 'C-c c d') to jump to their definition and see how
;; they are implemented.

(add-to-list 'load-path "~/.doom.d/modules")

(defmacro -- (var)
  `(when ,var
     (setq ,var (- ,var 1))))

(defmacro ++ (var)
  `(when ,var
     (setq ,var (+ ,var 1))))


;(when (or (not debug-my-config) (> debug-my-config 0)) (dec debug-my-config)
↪  (message "a"))

(setq use-package-verbose t)
(async-bytecomp-package-mode 1)

(defun after-startup (func)
  (unless debug-my-config (add-hook! after-startup-hook
                            (load-module func))))

(mkdir (concat doom-private-dir "modules") t)

;; Variable to determine how many modules are loaded, for debugging
;; Any number is number of modules
;; nil means all
(setq debug-my-config nil)
(defmacro load-module (module)
   `(when (or (not debug-my-config) (> debug-my-config 0)) (-- debug-my-config)
   ↪  (make-thread
   (let ((time (current-time)))
     ;(message (format "Loading %s." func-name))
     (require ,module)
```

```
    (message (format "Loaded %s in %.06f." ,module (float-time (time-since
    ↪  time)))))))

;; Load a module only if dependency could successfully be loaded
(defmacro load-module-if (dependency module)
`(when (require ,dependency nil 'noerror)
   (load-module ,module)))

(load-module 'cl) ; Still a requirement for ivy
(load-module 'exwm-tweaks)
(load-module 'general)
(load-module 'navigation)
(load-module 'shortcuts)
(load-module 'private-config)
(load-module 'search)
(load-module 'read-aloud)
(load-module 'read-single)
(load-module 'pamparam)
(load-module 'beancount-tweaks)
(load-module 'popes)
;(load-module '(require 'bible))
(load-module 'keycast-tweaks)
(load-module 'wttrin)
(load-module 'org-tweaks)
(load-module 'languages)
(when (require 'mu4e nil 'noerror)
  (load-module 'email)
  (load-module 'email-config)
  (load-module 'email-accounts))
(load-module 'latex-tweaks)
; Unsorted
(setq doom-modeline-enable-word-count t)
;(doom/quickload-session)

(defun doom--get-modules (file)
  (unless (file-exists-p file)
    (user-error "%s does not exist" file))
  (with-temp-buffer
    (insert-file-contents file)
    (when (re-search-forward "(doom! " nil t)
      (goto-char (match-beginning 0))
      (cdr (sexp-at-point)))))

(defun doom--put-modules (tmpfile modules)
  (with-temp-file tmpfile
    (delay-mode-hooks (emacs-lisp-mode))
    (insert (replace-regexp-in-string " " "\n" (prin1-to-string modules)))
    (indent-region (point-min) (point-max))))

;;;###autoload
(defun doom/what-has-changed ()
  "Open an ediff session to compare the module list in
     ~/.emacs.d/init.example.el and ~/.doom.d/init.el."
  (interactive)
  (let ((old-modules (doom--get-modules (expand-file-name (concat
  ↪  doom-emacs-dir "templates/init.example.el")))
```

```
        (new-modules (doom--get-modules (expand-file-name "init.el"
        ↪  doom-private-dir)))
        (example-init-el "/tmp/doom-init.example.el")
        (private-init-el "/tmp/doom-private-init.el"))
    (doom--put-modules example-init-el old-modules)
    (doom--put-modules private-init-el new-modules)
    (ediff private-init-el example-init-el)))
```

2 Packages

```
;; -*- no-byte-compile: t; -*-
;;; $DOOMDIR/packages.el

;; To install a package with Doom you must declare them here and run 'doom
↪  sync'
;; on the command line, then restart Emacs for the changes to take effect -- or
;; use 'M-x doom/reload'.


;; To install SOME-PACKAGE from MELPA, ELPA or emacsmirror:
;(package! some-package)

;; To install a package directly from a remote git repo, you must specify a
;; `:recipe'. You'll find documentation on what `:recipe' accepts here:
;; https://github.com/raxod502/straight.el#the-recipe-format
;(package! another-package
;  :recipe (:host github :repo "username/repo"))

;; If the package you are trying to install does not contain a PACKAGENAME.el
;; file, or is located in a subdirectory of the repo, you'll need to specify
;; `:files' in the `:recipe':
;(package! this-package
;  :recipe (:host github :repo "username/repo"
;           :files ("some-file.el" "src/lisp/*.el")))

;; If you'd like to disable a package included with Doom, you can do so here
;; with the `:disable' property:
;(package! builtin-package :disable t)

;; You can override the recipe of a built in package without having to specify
;; all the properties for `:recipe'. These will inherit the rest of its recipe
;; from Doom or MELPA/ELPA/Emacsmirror:
;(package! builtin-package :recipe (:nonrecursive t))
;(package! builtin-package-2 :recipe (:repo "myfork/package"))

;; Specify a `:branch' to install a package from a particular branch or tag.
;; This is required for some packages whose default branch isn't 'master'
↪  (which
;; our package manager can't deal with; see raxod502/straight.el#279)
;(package! builtin-package :recipe (:branch "develop"))

;; Use `:pin' to specify a particular commit to install.
;(package! builtin-package :pin "1a2b3c4d5e")
```

```
;; Doom's packages are pinned to a specific commit and updated from release to
;; release. The `unpin!' macro allows you to unpin single packages...
;(unpin! pinned-package)
;; ...or multiple packages
;(unpin! pinned-package another-pinned-package)
;; ...Or *all* packages (NOT RECOMMENDED; will likely break things)
;(unpin! t)

(package! xelb)
(package! exwm)
(package! winum)
(package! switch-window)
(package! try)
(package! counsel)
(package! ivy)
(package! all-the-icons-ibuffer)
(package! all-the-icons-ivy)
(package! swiper)
(package! ido-vertical-mode)
(package! amx)
(package! diredfl)
(package! all-the-icons-dired)
(package! diredful)
(package! dired-git-info)
(package! async)
(package! dired-quick-sort)
(package! openwith)
(package! lispy)
(package! hydra)
(package! ace-link)
(package! sudo-edit)
(package! beancount :recipe (:host github :repo "beancount/beancount-mode")
  :pin "ea8257881b7e276e8d170d724e3b2e179f25cb77")
(package! rotate)
(package! vlf)
(package! evil-escape :disable t)

; Auto Activating Snippets
(package! aas :recipe (:host github :repo "ymarco/auto-activating-snippets")
  :pin "1699bec4d244a1f62af29fe4eb8b79b6d2fccf7d")

(package! string-inflection :pin "fd7926ac17293e9124b31f706a4e8f38f6a9b855")

(package! info-colors :pin "47ee73cc19b1049eef32c9f3e264ea7ef2aaf8a5")
(package! modus-themes :pin "392ebb115b07f8052d512ec847619387d109edd6")
(package! theme-magic :pin "844c4311bd26ebafd4b6a1d72ddcc65d87f074e3")
(package! keycast :pin "04ba7519f34421c235bac458f0192c130f732f12")
(package! page-break-lines :recipe (:host github :repo
↪  "purcell/page-break-lines"))
(package! spray :pin "74d9dcfa2e8b38f96a43de9ab0eb13364300cb46")
;(package! authinfo-color-mode
;  :recipe (:repo "lisp/authinfo-color-mode"))
(package! systemd :pin "b6ae63a236605b1c5e1069f7d3afe06ae32a7bae")
; Ebooks
(package! calibredb :pin "cb93563d0ec9e0c653210bc574f9546d1e7db437")
(package! nov :pin "b3c7cc28e95fe25ce7b443e5f49e2e45360944a3")
```

```
; Email
(package! org-mime)
(package! org-auto-tangle)
(package! mu4e-alert)
(package! outline-minor-faces)
(package! mu4e-conversation)
(package! elfeed-summary)
(package! org-modern)
(package! good-scroll)
(package! desktop-environment)
(package! emms)
(package! elfeed-tube)
(package! mpv)
(package! elfeed-tube-mpv)
(package! spell-fu)
(package! wallpaper)
```

3 Secrets crypt

—BEGIN PGP MESSAGE—
jA0ECQMCpaa2IHWz/qX/0ncB5MHYMuE7woX9lpKqkvubmgY7wy3/6LCoYMn6XYRY
sY2v9BQxlfFkz+rwgWnsdRYeGLR8Yd8mpHGeGg4jrvC8v5eiOhaqCKpyNiAcM-
pOv age95IpQjrQoIy0CPDtiouDWKNrXrTSQctareltEAkkB+p48HdVkKg== =z0qj
—END PGP MESSAGE—

4 Modules

4.1 Utility functions

```
(load-module 'utility-functions)
```

4.1.1 Code

```
(require 'string-inflection)
(defun string-inflection-title-to-lisp-case-function (title-str)
  "Title String for Something => title-string-for-something"
  (string-inflection-kebab-case-function (s-replace-all '((" " . "-"))
  ↪  title-str)))
(provide 'utility-functions)
```

4.2 Autocorrect

```
(load-module 'auto-correct)
```

```
;;; auto-correct.el -*- lexical-binding: t; -*-

(define-key ctl-x-map "\C-i"
  #'endless/ispell-word-then-abbrev)
```

```
(defun endless/simple-get-word ()
  (car-safe (save-excursion (ispell-get-word nil))))

(defun endless/ispell-word-then-abbrev (p)
  "Call `ispell-word', then create an abbrev for it.
     With prefix P, create local abbrev. Otherwise it will
     be global.
     If there's nothing wrong with the word at point, keep
     looking for a typo until the beginning of buffer. You can
     skip typos you don't want to fix with `SPC', and you can
     abort completely with `C-g'."
  (interactive "P")
  (let (bef aft)
    (save-excursion
      (while (if (setq bef (endless/simple-get-word))
                 ;; Word was corrected or used quit.
                 (if (ispell-word nil 'quiet)
                     nil ; End the loop.
                   ;; Also end if we reach `bob'.
                   (not (bobp)))
               ;; If there's no word at point, keep looking
               ;; until `bob'.
               (not (bobp)))
        (backward-word)
        (backward-char))
      (setq aft (endless/simple-get-word)))
    (if (and aft bef (not (equal aft bef)))
        (let ((aft (downcase aft))
              (bef (downcase bef)))
          (define-abbrev
            (if p local-abbrev-table global-abbrev-table)
            bef aft)
          (message "\"%s\" now expands to \"%s\" %sally"
                   bef aft (if p "loc" "glob")))
      (user-error "No typo at or before point"))))

(setq save-abbrevs 'silently)
(setq-default abbrev-mode t)

(provide 'auto-correct)
```

4.3 Beancount

```
(load-module 'beancount-tweaks)
```

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

(use-package! beancount
  :mode ("\\.beancount\\'" . beancount-mode)
  :init
  (after! all-the-icons
    (add-to-list 'all-the-icons-icon-alist
                 '("\\.beancount\\'" all-the-icons-material "attach_money"
                 ↪  :face all-the-icons-lblue))
    (add-to-list 'all-the-icons-mode-icon-alist
```

```
                '(beancount-mode all-the-icons-material "attach_money" :face
                ↪  all-the-icons-lblue)))
  :config
  (setq beancount-electric-currency t)
  (defun beancount-bal ()
    "Run bean-report bal."
    (interactive)
    (let ((compilation-read-command nil))
      (beancount--run "bean-report"
                      (file-relative-name buffer-file-name) "bal")))
  (map! :map beancount-mode-map
        :n "TAB" #'beancount-align-to-previous-number
        :i "RET" (cmd! (newline-and-indent)
        ↪  (beancount-align-to-previous-number))))

(provide 'beancount-tweaks)
```

4.4 Dired

4.4.1 Dired Inline images

```
(load-module 'dired-inline-images)
```

Code

```
;;; dired-inline-images.el -*- lexical-binding: t; -*-

(defun dired-preview--dired-line-is-previewable ()
  "Return non-nil if line under point is previewable"
  (let* ((fname (dired-get-filename nil))
         (ext (upcase (file-name-extension fname)))
         (allowed-extensions '("PBM" "XBM" "XPM" "GIF" "JPEG" "JPG" "TIFF"
         ↪  "TIF" "PNG" "SVG"))
         (search-fun (apply-partially (lambda (a b) (string= a b)) ext))
         (is-ext-allowed (seq-find search-fun allowed-extensions nil)))
    is-ext-allowed))

(defun dired-preview--readin (filename)
  "Read in the file.
Return a string suitable for insertion in `dired' buffer."
    (let ((preview-image (create-image filename 'imagemagick nil :height 200)))
      (with-temp-buffer
        (insert-image preview-image)
        (insert "\n")
        (buffer-string))))

(defun dired-preview-insert ()          ;; Copied more or less directly from
↪  dired-subtree
  "Insert preview under this file."
  (interactive)
  (when (and (dired-preview--dired-line-is-previewable)
             (not (dired-subtree--is-expanded-p)))
    (let* ((filename (dired-get-filename nil))
           (listing (dired-preview--readin filename))
```

```
        beg end)
    (read-only-mode -1)
    (move-end-of-line 1)
    ;; this is pretty ugly, I'm sure it can be done better
    (save-excursion
      (insert listing)
      (setq end (+ (point) 2)))
    (newline)
    (setq beg (point))
    (let ((inhibit-read-only t))
      (remove-text-properties (1- beg) beg '(dired-filename)))
    (let* ((ov (make-overlay beg end))
           (parent (dired-subtree--get-ov (1- beg)))
           (depth (or (and parent (+ 2 (overlay-get parent
           ↪  'dired-subtree-depth)))
                      2))
           (face (intern (format "dired-subtree-depth-%d-face" depth))))
      (when dired-subtree-use-backgrounds
        (overlay-put ov 'face face))
      ;; refactor this to some function
      (overlay-put ov 'line-prefix
                   (if (stringp dired-subtree-line-prefix)
                       (if (not dired-subtree-use-backgrounds)
                           (apply 'concat (-repeat depth
                           ↪  dired-subtree-line-prefix))
                         (cond
                          ((eq nil dired-subtree-line-prefix-face)
                           (apply 'concat
                                  (-repeat depth dired-subtree-line-prefix)))
                          ((eq 'subtree dired-subtree-line-prefix-face)
                           (concat
                            dired-subtree-line-prefix
                            (propertize
                             (apply 'concat
                                    (-repeat (1- depth)
                                    ↪  dired-subtree-line-prefix))
                             'face face)))
                          ((eq 'parents dired-subtree-line-prefix-face)
                           (concat
                            dired-subtree-line-prefix
                            (apply 'concat
                                   (--map
                                    (propertize dired-subtree-line-prefix
                                                'face
                                                (intern (format
                                                ↪  "dired-subtree-depth-%d-face"
                                                ↪  it)))
                                    (number-sequence 1 (1- depth)))))))))
                     (funcall dired-subtree-line-prefix depth)))
      (overlay-put ov 'dired-subtree-name filename)
      (overlay-put ov 'dired-subtree-parent parent)
      (overlay-put ov 'dired-subtree-depth depth)
      (overlay-put ov 'evaporate t)
      (push ov dired-subtree-overlays))
    (goto-char (- beg 1))
    (dired-move-to-filename)
    (read-only-mode 1)
```

```
      (run-hooks 'dired-subtree-after-insert-hook))))

(defun dired-preview-insert-preview-or-subtree (orig-fun)
  "Call the right insert function for a preview or a subtree"
  (interactive)
  (cond ((dired-subtree--dired-line-is-directory-or-link-p) (apply orig-fun
  ↪  nil))
        ((dired-preview--dired-line-is-previewable) (dired-preview-insert))))

(advice-add 'dired-subtree-insert :around
↪  #'dired-preview-insert-preview-or-subtree)

(provide 'dired-inline-images)
```

4.4.2 Dired Tweaks

```
(load-module 'dired-tweaks)
```

Code

```
;;; dired-tweaks.el -*- lexical-binding: t; -*-

;; Dired
;;; Colourful dired
(use-package! diredfl
  :init (diredfl-global-mode 1))

(use-package! all-the-icons-dired
  :config
  ;(all-the-icons-dired-mode 1)
  (add-hook 'dired-mode-hook 'all-the-icons-dired-mode))

(defun dired-open-file ()
  "In dired, open the file named on this line."
  (interactive)
  (let* ((file (dired-get-filename nil t)))
    (message "Opening %s..." file)
    (call-process "xdg-open" nil 0 nil file)))


(define-minor-mode dired-follow-mode
  "Diplay file at point in dired after a move."
  :lighter " dired-f"
  :global t
  (if dired-follow-mode
      (advice-add 'dired-next-line :after (lambda (arg) (dired-display-file)))
    (advice-remove 'dired-next-line (lambda (arg) (dired-display-file)))))

(setq vc-follow-symlinks t
      dired-listing-switches "-ahlt"
      diredp-toggle-find-file-reuse-dir 1
      image-dired-thumb-size 100
      diredp-image-preview-in-tooltip 100
      dired-auto-revert-buffer t
```

```
      diredp-hide-details-initially-flag nil
      dired-hide-details-mode 0)

(defmacro image-view (direction)
  `(lambda ()
     (interactive)
     (quit-window)
     (let ((pt (point))
           filename)
       (or (ignore-errors
             (catch 'filename
               (while (dired-next-line ,direction)
                 (when (image-type-from-file-name
                        (setq filename (dired-get-filename)))
                   (throw 'filename filename)))))
           (goto-char pt))
       (dired-view-file))))

(eval-after-load "image-mode"
  '(progn
    (define-key image-mode-map "n" (image-view 1))
    (define-key image-mode-map "p" (image-view -1))))

;(use-package dired-k
;  ;; use dired-k as alternative to revert buffer. This will refresh git status
;  :hook (dired-mode . dired-k)
;  :bind (:map dired-mode-map
;              ("g" . dired-k)))

(use-package! diredful
  :config (diredful-mode 1))

; I don't like colors that much, icons already do everything
;(use-package dired-rainbow
;
;  :defer t
;  :config
;  (progn
;    (dired-rainbow-define-chmod directory "#6cb2eb" "d.*")
;    (dired-rainbow-define html "#eb5286" ("css" "less" "sass" "scss" "htm"
↪  "html" "jhtm" "mht" "eml" "mustache" "xhtml"))
;    (dired-rainbow-define xml "#f2d024" ("xml" "xsd" "xsl" "xslt" "wsdl" "bib"
↪  "json" "msg" "pgn" "rss" "yaml" "yml" "rdata"))
;    (dired-rainbow-define document "#9561e2" ("docm" "doc" "docx" "odb" "odt"
↪  "pdb" "pdf" "ps" "rtf" "djvu" "epub" "odp" "ppt" "pptx"))
;    (dired-rainbow-define markdown "#ffed4a" ("org" "etx" "info" "markdown"
↪  "md" "mkd" "nfo" "pod" "rst" "tex" "textfile" "txt"))
;    (dired-rainbow-define database "#6574cd" ("xlsx" "xls" "csv" "accdb" "db"
↪  "mdb" "sqlite" "nc"))
;    (dired-rainbow-define media "#de751f" ("mp3" "mp4" "MP3" "MP4" "avi"
↪  "mpeg" "mpg" "flv" "ogg" "mov" "mid" "midi" "wav" "aiff" "flac"))
;    (dired-rainbow-define image "#f66d9b" ("tiff" "tif" "cdr" "gif" "ico"
↪  "jpeg" "jpg" "png" "psd" "eps" "svg"))
;    (dired-rainbow-define log "#c17d11" ("log"))
;    (dired-rainbow-define shell "#f6993f" ("awk" "bash" "bat" "sed" "sh" "zsh"
↪  "vim"))
```

```
;    (dired-rainbow-define interpreted "#38c172" ("py" "ipynb" "rb" "pl" "t"
↪  "msql" "mysql" "pgsql" "sql" "r" "clj" "cljs" "scala" "js"))
;    (dired-rainbow-define compiled "#4dc0b5" ("asm" "cl" "lisp" "el" "c" "h"
↪  "c++" "h++" "hpp" "hxx" "m" "cc" "cs" "cp" "cpp" "go" "f" "for" "ftn" "f90"
↪  "f95" "f03" "f08" "s" "rs" "hi" "hs" "pyc" ".java"))
;    (dired-rainbow-define executable "#8cc4ff" ("exe" "msi"))
;    (dired-rainbow-define compressed "#51d88a" ("7z" "zip" "bz2" "tgz" "txz"
↪  "gz" "xz" "z" "Z" "jar" "war" "ear" "rar" "sar" "xpi" "apk" "xz" "tar"))
;    (dired-rainbow-define packaged "#faad63" ("deb" "rpm" "apk" "jad" "jar"
↪  "cab" "pak" "pk3" "vdf" "vpk" "bsp"))
;    (dired-rainbow-define encrypted "#ffed4a" ("gpg" "pgp" "asc" "bfe" "enc"
↪  "signature" "sig" "p12" "pem"))
;    (dired-rainbow-define fonts "#6cb2eb" ("afm" "fon" "fnt" "pfb" "pfm" "ttf"
↪  "otf"))
;    (dired-rainbow-define partition "#e3342f" ("dmg" "iso" "bin" "nrg" "qcow"
↪  "toast" "vcd" "vmdk" "bak"))
;    (dired-rainbow-define vc "#0074d9" ("git" "gitignore" "gitattributes"
↪  "gitmodules"))
;    (dired-rainbow-define-chmod executable-unix "#38c172" "-.*x.*")
;    ))

(use-package! dired-git-info
  :config
  (setq dgi-auto-hide-details-p nil)
  (add-hook 'dired-after-readin-hook 'dired-git-info-auto-enable))


(use-package! async
  :init (dired-async-mode 1))

(use-package! dired-quick-sort
  :config
  (dired-quick-sort-setup)
  (setq dired-quick-sort-suppress-setup-warning t))

(use-package! openwith
  :config
  (setq openwith-associations
        (cond
         ((string-equal system-type "darwin")
          '(("\\.\\(dmg\\|doc\\|docs\\|xls\\|xlsx\\)$"
             "open" (file))
            ("\\.\\(mp4\\|mp3\\|webm\\|avi\\|flv\\|mov\\)$"
             "open" ("-a" "VLC" file))))
         ((string-equal system-type "gnu/linux")

          ↪  '(("\\.\\(mp4\\|m4a\\|mp3\\|mkv\\|webm\\|avi\\|flv\\|mov\\|pdf\\|part\\)$"
            "xdg-open" (file))))))
  (openwith-mode t)
  (setq large-file-warning-threshold 3000000000))

(define-key dired-mode-map (kbd "<backspace>") 'dired-up-directory)

;; Docview j and k go forward a line which is weird behaviour in a pdf
;; Paging is prefered to scrolling
(after! doc-view-mode
(define-key doc-view-mode-map (kbd "j") 'doc-view-next-page)
```

```
(define-key doc-view-mode-map (kbd "k") 'doc-view-previous-page)
  )

;; Convert files automatically
(defun dired-convert-file ()
  "Converts pptx or docx files to pdf"
  (interactive)
  (cl-map 'nil '(lambda (file)
                  (let ((ext (file-name-extension file))
                        (base-name-sans-ext (file-name-sans-extension
                        ↪  (file-name-nondirectory file))))
                    (cond
                     ((or (string-equal ext "pptx") (string-equal ext "ppt"))
                      (async-shell-command (format "libreoffice --headless
                      ↪  --invisible --convert-to pdf \"%s\"" file)))
                     ((or (string-equal ext "docx") (string-equal ext "doc")
                     ↪  (string-equal ext "epub") (string-equal ext "tex")
                     ↪  (string-equal "html") (string-equal ext "org")
                     ↪  (string-equal ext "txt"))
                      (async-shell-command (format "pandoc -i \"%s\" -o
                      ↪  \"%s.pdf\"" file base-name-sans-ext)))
                     ((or (string-equal ext "jpg") (string-equal ext "jpeg")
                     ↪  (string-equal ext "png"))
                      (async-shell-command (format "convert \"%s\" -rotate 90
                      ↪  \"%s\"" file file)))
                     ))) (dired-get-marked-files)))

;; Toggle youtube-dl-list if in elfeed-youtube buffer, else perform regular
↪  load
(defun dired-load-or-youtube-toggle ()
  (interactive)
  (cond ((string-equal (buffer-name) "elfeed-youtube")
         (youtube-dl-list))
        ((eq major-mode 'youtube-dl-list-mode) (kill-buffer))
        (t (dired-do-load))))

(map! :map dired-mode-map
      ;:after dired-mode
      ;:n doom-leader-key nil
      :n "c" #'dired-convert-file
      :n "L" #'dired-load-or-youtube-toggle)

(map! :map youtube-dl-list-mode-map
      :n "L" #'dired-load-or-youtube-toggle)

(provide 'dired-tweaks)
```

4.5 Ebooks

```
(load-module 'ebook-tweaks)
```

4.5.1 Code

```
;;; ebook-tweaks.el -*- lexical-binding: t; -*-

(use-package! calibredb
  :commands calibredb
  :config
  (setq calibredb-root-dir "/home/user/sshfs/calibre/"
        calibredb-db-dir (expand-file-name "metadata.db" calibredb-root-dir)
        sql-sqlite-program "sqlite3")

(setq calibredb-id-width 12)
(setq calibredb-format-all-the-icons t)
(setq calibredb-format-icons-in-terminal t)
(setq calibredb-format-character-icons t)
  (map! :map calibredb-show-mode-map
        :ne "?" #'calibredb-entry-dispatch
        :ne "o" #'calibredb-find-file
        :ne "O" #'calibredb-find-file-other-frame
        :ne "V" #'calibredb-open-file-with-default-tool
        :ne "s" #'calibredb-set-metadata-dispatch
        :ne "e" #'calibredb-export-dispatch
        :ne "q" #'calibredb-entry-quit
        :ne "." #'calibredb-open-dired
        :ne [tab] #'calibredb-toggle-view-at-point
        :ne "M-t" #'calibredb-set-metadata--tags
        :ne "M-a" #'calibredb-set-metadata--author_sort
        :ne "M-A" #'calibredb-set-metadata--authors
        :ne "M-T" #'calibredb-set-metadata--title
        :ne "M-c" #'calibredb-set-metadata--comments)
  (map! :map calibredb-search-mode-map
        :ne [mouse-3] #'calibredb-search-mouse
        :ne "RET" #'calibredb-find-file
        :ne "?" #'calibredb-dispatch
        :ne "a" #'calibredb-add
        :ne "A" #'calibredb-add-dir
        :ne "c" #'calibredb-clone
        :ne "d" #'calibredb-remove
        :ne "D" #'calibredb-remove-marked-items
        :ne "j" #'calibredb-next-entry
        :ne "k" #'calibredb-previous-entry
        :ne "l" #'calibredb-virtual-library-list
        :ne "L" #'calibredb-library-list
        :ne "n" #'calibredb-virtual-library-next
        :ne "N" #'calibredb-library-next
        :ne "p" #'calibredb-virtual-library-previous
        :ne "P" #'calibredb-library-previous
        :ne "s" #'calibredb-set-metadata-dispatch
        :ne "S" #'calibredb-switch-library
        :ne "o" #'calibredb-find-file
        :ne "O" #'calibredb-find-file-other-frame
        :ne "v" #'calibredb-view
        :ne "V" #'calibredb-open-file-with-default-tool
        :ne "." #'calibredb-open-dired
        :ne "b" #'calibredb-catalog-bib-dispatch
        :ne "e" #'calibredb-export-dispatch
        :ne "r" #'calibredb-search-refresh-and-clear-filter
```

```
        :ne "R" #'calibredb-search-clear-filter
        :ne "q" #'calibredb-search-quit
        :ne "m" #'calibredb-mark-and-forward
        :ne "f" #'calibredb-toggle-favorite-at-point
        :ne "x" #'calibredb-toggle-archive-at-point
        :ne "h" #'calibredb-toggle-highlight-at-point
        :ne "u" #'calibredb-unmark-and-forward
        :ne "i" #'calibredb-edit-annotation
        :ne "DEL" #'calibredb-unmark-and-backward
        :ne [backtab] #'calibredb-toggle-view
        :ne [tab] #'calibredb-toggle-view-at-point
        :ne "M-n" #'calibredb-show-next-entry
        :ne "M-p" #'calibredb-show-previous-entry
        :ne "/" #'calibredb-search-live-filter
        :ne "M-t" #'calibredb-set-metadata--tags
        :ne "M-a" #'calibredb-set-metadata--author_sort
        :ne "M-A" #'calibredb-set-metadata--authors
        :ne "M-T" #'calibredb-set-metadata--title
        :ne "M-c" #'calibredb-set-metadata--comments))

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

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

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

  (defun +nov-mode-setup ()
    (face-remap-add-relative 'variable-pitch
                             :family "Merriweather"
                             :height 1.4
                             :width 'semi-expanded)
    (face-remap-add-relative 'default :height 0.7)
    (setq-local line-spacing 0.1
                next-screen-context-lines 4
                shr-use-colors nil)
    (require 'visual-fill-column nil t)
    (setq-local visual-fill-column-center-text t
                visual-fill-column-width 81
                nov-text-width 120)
    (visual-fill-column-mode 1)
```

```
    (hl-line-mode -1)

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

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

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

(provide 'ebook-tweaks)
```

4.6 Org-Roam

```
(load-module 'org-roam-tweaks)
```

4.6.1 Requirements

```
(package! org-roam)
(package! org-roam-ui)
(package! org-roam-timestamps)
(package! org-roam-bibtex)
(package! citar-org-roam)
```

4.6.2 Code

```
(setq org-roam-directory "~/dox/sync/org/roam")
(org-roam-db-autosync-mode 1)
(provide 'org-roam-tweaks)
```

4.7 Editing

```
(load-module 'editing)
```

4.7.1 Requirements

#+begin$_{\text{src}}$ emacs-lisp :tangle packages.el

4.7.2 Code

#+end$_{\text{src}}$

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

(defun which-active-modes ()
  "Return which minor modes are enabled in the current buffer."
  (let ((active-modes))
    (mapc (lambda (mode) (condition-case nil
                         (if (and (symbolp mode) (symbol-value mode))
                             (add-to-list 'active-modes mode))
                       (error nil) ))
          minor-mode-list)
    (format "%s" active-modes)))

(defun replace-regexp-entire-buffer (pattern replacement)
  "Perform regular-expression replacement throughout buffer."
  (interactive
   (let ((args (query-replace-read-args "Replace" t)))
     (setcdr (cdr args) nil)    ; remove third value returned from query---args
     args))
  (save-excursion
    (goto-char (point-min))
    (while (re-search-forward pattern nil t)
      (replace-match replacement))))


(setq toggle-auto-fill-boolean nil
      which-key-idle-delay 0.5
      which-key-allow-multiple-replacements t)

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

(after! company
  (setq company-idle-delay 0.5
        company-minimum-prefix-length 2
        company-show-numbers t)
  (add-hook 'evil-normal-state-entry-hook #'company-abort))

(set-company-backend!
  '(text-mode
```

```
    markdown-mode
    gfm-mode)
  '(:seperate
    company-ispell
    company-files
    company-yasnippet))

(set-company-backend! 'ess-r-mode
  '(company-R-args company-R-objects company-dabbrev-code :separate))

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

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

(defun toggle-auto-fill-on ()
  (set-fill-column 100) ;80
  (auto-fill-mode t)
  (setq toggle-auto-fill-boolean t)
  ;(string-match-p "auto-fill-function" (which-active-modes))
  (message "auto-fill-mode on"))


(defun toggle-auto-fill-off ()
  (replace-regexp-entire-buffer "\n" " ")
  (auto-fill-mode nil)
  (setq toggle-auto-fill-boolean nil)
  (message "auto-fill-mode off")
  )

(defun toggle-auto-fill ()
  "Toggle auto fill mode and reset buffer to non-auto-fill."
  (interactive)
  (if toggle-auto-fill-boolean
      (toggle-auto-fill-off)
    (toggle-auto-fill-on)
    ))



(global-set-key (kbd "M-q") 'toggle-auto-fill)

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

(setq yas-triggers-in-field t)

(use-package! string-inflection
  :commands (string-inflection-all-cycle
             string-inflection-toggle
             string-inflection-camelcase
             string-inflection-lower-camelcase
             string-inflection-kebab-case
             string-inflection-underscore
             string-inflection-capital-underscore
             string-inflection-upcase)
```

```
  :init
  (map! :leader :prefix ("c~" . "naming convention")
        :desc "cycle" "~" #'string-inflection-all-cycle
        :desc "toggle" "t" #'string-inflection-toggle
        :desc "CamelCase" "c" #'string-inflection-camelcase
        :desc "downCase" "d" #'string-inflection-lower-camelcase
        :desc "kebab-case" "k" #'string-inflection-kebab-case
        :desc "under_score" "_" #'string-inflection-underscore
        :desc "Upper_Score" "u" #'string-inflection-capital-underscore
        :desc "UP_CASE" "U" #'string-inflection-upcase)
  (after! evil
    (evil-define-operator evil-operator-string-inflection (beg end _type)
      "Define a new evil operator that cycles symbol casing."
      :move-point nil
      (interactive "<R>")
      (string-inflection-all-cycle)
      (setq evil-repeat-info '([?g ?~])))
    (define-key evil-normal-state-map (kbd "g~")
    ↪  'evil-operator-string-inflection)
    (define-key evil-normal-state-map (kbd "<remap> <evil-next-line>")
    ↪  'evil-next-visual-line)
    (define-key evil-normal-state-map (kbd "<remap> <evil-previous-line>")
    ↪  'evil-previous-visual-line)
    (define-key evil-motion-state-map (kbd "<remap> <evil-next-line>")
    ↪  'evil-next-visual-line)
    (define-key evil-motion-state-map (kbd "<remap> <evil-previous-line>")
    ↪  'evil-previous-visual-line)
    ))

(sp-local-pair
 '(org-mode)
 "<<" ">>"
 :actions '(insert))

(use-package! authinfo-color-mode
  :mode ("authinfo.gpg\\'" . authinfo-color-mode)
  :init (advice-add 'authinfo-mode :override #'authinfo-color-mode))

(use-package! systemd
  :defer t)

(setq global-visual-line-mode t
 evil-respect-visual-line-mode t)

(provide 'editing)
```

4.8 Elfeed Tweaks

```
(load-module 'elfeed-tweaks)
```

4.8.1 Requirements

#+begin$_{\text{src}}$ emacs-lisp :tangle packages.el

4.8.2 Code

#+end$_{\text{src}}$

```
;;; elfeed-tweaks.el -*- lexical-binding: t; -*-

(setq rmh-elfeed-org-files (cons (expand-file-name "ext/elfeed/elfeed.org"
↪  doom-private-dir)())
      elfeed-db-directory (expand-file-name "ext/elfeed/db/" doom-private-dir)
      elfeed-thumbnail-dir "/tmp/elfeed-thumbnails/")


(map! :map elfeed-search-mode-map
      :after elfeed-search
                                        ;[remap kill-this-buffer] "q"
                                        ;[remap kill-buffer] "q"
      :n doom-leader-key nil
      :n "q" #'elfeed-save-summary
      :n "e" #'elfeed-update
      :n "r" #'elfeed-search-untag-all-unread
      :n "u" #'elfeed-search-tag-all-unread
      :n "s" #'elfeed-search-live-filter
      :n "RET" #'elfeed-search-show-entry
      :n "p" #'elfeed-show-pdf
      :n "v" #'elfeed-search-youtube-dl
      :n "L" #'youtube-dl-list
      :n "+" #'elfeed-search-tag-all
      :n "-" #'elfeed-search-untag-all
      :n "S" #'elfeed-search-set-filter
      :n "b" #'elfeed-search-browse-url
      :n "t" #'elfeed-search-thumbnail
      :n "y" #'elfeed-search-yank)

(map! :map elfeed-show-mode-map
      :after elfeed-show
                                        ;[remap kill-this-buffer] "q"
                                        ;[remap kill-buffer] "q"
      :n doom-leader-key nil
      :nm "q" #'elfeed-save-close
      :nm "o" #'ace-link-elfeed
      :nm "A" #'elfeed-wget-url
      :nm "RET" #'elfeed-tube-mpv-open
      :nm "n" #'elfeed-show-next
      :nm "N" #'elfeed-show-prev
      :nm "p" #'elfeed-show-pdf
      :nm "v" #'elfeed-show-youtube-dl
      :nm "d" #'elfeed-show-download-enclosure
      :nm "D" #'elfeed-show-download-enclosure
      :nm "L" #'youtube-dl-list
      :nm "+" #'elfeed-show-tag
      :nm "-" #'elfeed-show-untag
      :nm "s" #'elfeed-show-new-live-search
      :nm "y" #'elfeed-show-yank)
(map! :map elfeed-summary-mode-map
      :after elfeed-summary
      :n "L" #'youtube-dl-list
      :n "V" #'open-yt-dl-videos
```

```
      :n "R" #'elfeed-summary-load-update
      :n "C-x C-s" #'elfeed-summary-save
      :n "RET" #'elfeed-summary-action-save-location)

(after! elfeed-search
  (set-evil-initial-state! 'elfeed-search-mode 'normal))
(after! elfeed-show-mode
  (set-evil-initial-state! 'elfeed-show-mode   'normal))

(after! evil-snipe
  (push 'elfeed-show-mode   evil-snipe-disabled-modes)
  (push 'elfeed-search-mode evil-snipe-disabled-modes))

(after! elfeed
  (elfeed-org)
  (use-package! elfeed-link)
  (elfeed-db-load)
  (setq ;elfeed-search-filter "@1-week-ago +unread"
        elfeed-search-filter "@3-days-ago unread"
        flycheck-global-modes '(not . (elfeed-search-mode))
        elfeed-summary--only-unread t
        elfeed-search-print-entry-function '+rss/elfeed-search-print-entry
        elfeed-search-title-min-width 80
        elfeed-show-entry-switch #'pop-to-buffer
        elfeed-show-entry-delete #'+rss/delete-pane
        elfeed-show-refresh-function #'+rss/elfeed-show-refresh--better-style
        shr-max-image-proportion 0.6)

  (add-hook! 'elfeed-show-mode-hook (hide-mode-line-mode 1))
(defun elfeed-eb-garamond ()
  (buffer-face-set '(:family "EB Garamond" :height 120)))

(add-hook! 'elfeed-show-mode-hook 'elfeed-eb-garamond)
  (add-hook! 'elfeed-search-update-hook #'hide-mode-line-mode)

  (defface elfeed-show-title-face '((t (:weight ultrabold :slant italic :height
  ↪  1.5)))
    "title face in elfeed show buffer"
    :group 'elfeed)
  (defface elfeed-show-author-face `((t (:weight light)))
    "title face in elfeed show buffer"
    :group 'elfeed)
  (set-face-attribute 'elfeed-search-title-face nil
                      :foreground 'nil
                      :weight 'light)

  (defadvice! +rss-elfeed-wrap-h-nicer ()
    "Enhances an elfeed entry's readability by wrapping it to a width of
       `fill-column' and centering it with `visual-fill-column-mode'."
    :override #'+rss-elfeed-wrap-h
    (setq-local truncate-lines nil
                shr-width 120
                visual-fill-column-center-text t
                default-text-properties '(line-height 1.1))
    (let ((inhibit-read-only t)
          (inhibit-modification-hooks t))
      (visual-fill-column-mode nil)
```

```
    (setq-local shr-current-font '(:family "Linux Libertine O" :height 1.2))
    (set-buffer-modified-p nil)))

(defun +rss/elfeed-search-print-entry (entry)
  "Print ENTRY to the buffer."
  (let* ((elfeed-goodies/tag-column-width 40)
         (elfeed-goodies/feed-source-column-width 30)
         (title (or (elfeed-meta entry :title) (elfeed-entry-title entry)
         ↪  ""))
         (title-faces (elfeed-search--faces (elfeed-entry-tags entry)))
         (feed (elfeed-entry-feed entry))
         (feed-title
          (when feed
            (or (elfeed-meta feed :title) (elfeed-feed-title feed))))
         (tags (mapcar #'symbol-name (elfeed-entry-tags entry)))
         (tags-str (concat (mapconcat 'identity tags ",")))
         (title-width (- (window-width)
         ↪  elfeed-goodies/feed-source-column-width
                         elfeed-goodies/tag-column-width 4))

         (tag-column (elfeed-format-column
                      tags-str (elfeed-clamp (length tags-str)
                                             elfeed-goodies/tag-column-width
                                             elfeed-goodies/tag-column-width)
                      :left))
         (feed-column (elfeed-format-column
                       feed-title (elfeed-clamp
                       ↪  elfeed-goodies/feed-source-column-width

                                                 ↪  elfeed-goodies/feed-source-column-width

                                                 ↪  elfeed-goodies/feed-source-column-width)
                       :left)))

                                        ;(insert (propertize feed-column 'face
                                        ↪  'elfeed-search-feed-face) " ")
                                        ;(insert (propertize tag-column 'face
                                        ↪  'elfeed-search-tag-face) " ")
    (insert (propertize title 'face title-faces 'kbd-help title))
    (setq-local line-spacing 0.2)))

(defun +rss/elfeed-show-refresh--better-style ()
  "Update the buffer to match the selected entry, using a mail-style."
  (interactive)
  (let* ((inhibit-read-only t)
         (title (elfeed-entry-title elfeed-show-entry))
         (date (seconds-to-time (elfeed-entry-date elfeed-show-entry)))
         (author (elfeed-meta elfeed-show-entry :author))
         (link (elfeed-entry-link elfeed-show-entry))
         (tags (elfeed-entry-tags elfeed-show-entry))
         (tagsstr (mapconcat #'symbol-name tags ", "))
         (nicedate (format-time-string "%a, %e %b %Y %T %Z" date))
         (content (elfeed-deref (elfeed-entry-content elfeed-show-entry)))
         (type (elfeed-entry-content-type elfeed-show-entry))
         (feed (elfeed-entry-feed elfeed-show-entry))
         (feed-title (elfeed-feed-title feed))
         (base (and feed (elfeed-compute-base (elfeed-feed-url feed)))))
```

```
      (erase-buffer)
      (insert "\n")
      (insert (format "%s\n\n" (propertize title 'face
      ↪  'elfeed-show-title-face)))
      (insert (format "%s\t" (propertize feed-title 'face
      ↪  'elfeed-search-feed-face)))
      (when (and author elfeed-show-entry-author)
        (insert (format "%s\n" (propertize author 'face
        ↪  'elfeed-show-author-face))))
      (insert (format "%s\n\n" (propertize nicedate 'face
      ↪  'elfeed-log-date-face)))
      (when tags
        (insert (format "%s\n"
                        (propertize tagsstr 'face 'elfeed-search-tag-face))))
      ;; (insert (propertize "Link: " 'face 'message-header-name))
      ;; (elfeed-insert-link link link)
      ;; (insert "\n")
      (cl-loop for enclosure in (elfeed-entry-enclosures elfeed-show-entry)
               do (insert (propertize "Enclosure: " 'face
               ↪  'message-header-name))
               do (elfeed-insert-link (car enclosure))
               do (insert "\n"))
      (insert "\n")
      (if content
          (if (eq type 'html)
              (elfeed-insert-html content base)
            (insert content))
        (insert (propertize "(empty)\n" 'face 'italic)))
      (goto-char (point-min))))

  (defface elfeed-youtube
    '((t :foreground "purple"))
    "Marks YouTube videos in Elfeed."
    :group 'elfeed)

  (defface elfeed-religion
    '((t :foreground "gold"))
    "Marks YouTube videos in Elfeed."
    :group 'elfeed)

  (defface elfeed-tech
    '((t :foreground "LightSteelBlue4"))
    "Marks Tech videos in Elfeed."
    :group 'elfeed)

  (push '(youtube elfeed-youtube)
        elfeed-search-face-alist)
  (push '(religion elfeed-religion)
        elfeed-search-face-alist)
  (push '(tech elfeed-tech)
        elfeed-search-face-alist)


  )

(after! elfeed-show
  (require 'url)
```

```
(defun elfeed-show-download-enclosure ()
  "Download the enclosure to yt-dlp directory"
  (interactive)
  (let*
      ((url-enclosure (car (elt (elfeed-entry-enclosures elfeed-show-entry)
      ↪  0)))
       (filename (concat elfeed-enclosure-default-dir "/"
       ↪  (elfeed-entry-title elfeed-show-entry) ".mp3")))
    (elfeed--download-enclosure url-enclosure filename)
    (message (format "Downloading %s" filename))))

(defvar elfeed-pdf-dir
  (expand-file-name "pdfs/"
                    (file-name-directory (directory-file-name
                    ↪  elfeed-enclosure-default-dir))))

(defvar elfeed-link-pdfs
  '(("https://www.jstatsoft.org/index.php/jss/article/view/v0\\([^/]+\\)" .
  ↪  "https://www.jstatsoft.org/index.php/jss/article/view/v0\\1/v\\1.pdf")
    ("http://arxiv.org/abs/\\([^/]+\\)" . "https://arxiv.org/pdf/\\1.pdf"))
  "List of alists of the form (REGEX-FOR-LINK . FORM-FOR-PDF)")

(defun elfeed-show-pdf (entry)
  (interactive
   (list (or elfeed-show-entry (elfeed-search-selected :ignore-region))))
  (let ((link (elfeed-entry-link entry))
        (feed-name (plist-get (elfeed-feed-meta (elfeed-entry-feed entry))
        ↪  :title))
        (title (elfeed-entry-title entry))
        (file-view-function
         (lambda (f)
           (when elfeed-show-entry
             (elfeed-kill-buffer))
           (pop-to-buffer (find-file-noselect f))))
        pdf)

    (let ((file (expand-file-name
                 (concat (subst-char-in-string ?/ ?, title) ".pdf")
                 (expand-file-name (subst-char-in-string ?/ ?, feed-name)
                                   elfeed-pdf-dir))))
      (if (file-exists-p file)
          (funcall file-view-function file)
        (dolist (link-pdf elfeed-link-pdfs)
          (when (and (string-match-p (car link-pdf) link)
                     (not pdf))
            (setq pdf (replace-regexp-in-string (car link-pdf) (cdr link-pdf)
            ↪  link))))
        (if (not pdf)
            (message "No associated PDF for entry")
          (message "Fetching %s" pdf)
          (unless (file-exists-p (file-name-directory file))
            (make-directory (file-name-directory file) t))
          (url-copy-file pdf file)
          (funcall file-view-function file))))))

)
```

```
(after! elfeed-summary
  (elfeed-org))

(defun elfeed-summary-save ()
  "Save database"
  (interactive)
  (elfeed-db-save-safe))

(defun elfeed-save-summary ()
  "Save database and go to summary"
  (interactive)
  (elfeed-db-save-safe)
  (kill-this-buffer)
  (elfeed-summary)
  (when (boundp 'elfeed-summary--current-pos)
    (goto-char elfeed-summary--current-pos)))

(defun elfeed-save-close ()
  "Save database and close rss"
  (interactive)
  (elfeed-db-save-safe)
  (+rss/delete-pane))

(defun elfeed-load-summary ()
  "Load database and go to summary"
  (interactive)
  (when (and (functionp 'elfeed-db-load) (not (get-buffer "*elfeed-summary*")))
    (make-thread (elfeed-db-load)))
  (elfeed-summary)
  (when (boundp 'elfeed-summary--current-pos)
    (progn
      (goto-char elfeed-summary--current-pos)
      (recenter-top-bottom))))

(defun elfeed-summary-load-update ()
  "Loads the database again before updating"
  (interactive)
  (elfeed-db-load)
  (message "Refreshing db...")
  (elfeed-update)
  (elfeed-summary-update))

(setq elfeed-summary-settings
      '(
        (group (:title . "Blogs [Security]")
               (:elements
                (query . (and people security))))
        (group (:title . "Blogs [People]")
               (:elements
                (query . (and people (not security)))
                ))
        (group (:title . "Religion")
               (:elements
                (query . religion)))
        (group (:title . "Cooking")
               (:elements
```

```
                (query . cooking)))
        (group (:title . "ASMR")
               (:elements
                (query . asmr)))
        (group (:title . "Crafting")
               (:elements
                (query . crafting)))
        (group (:title . "Entertainment")
               (:elements
                (query . entertainment)))
        (group (:title . "Finances")
               (:elements
                (query . finances)))
        (group (:title . "Foreign Places")
               (:elements
                (query . foreign_places)))
        (group (:title . "Geography")
               (:elements
                (query . geography)))
        (group (:title . "History")
               (:elements
                (query . history)))
        (group (:title . "Language")
               (:elements
                (query . language)))
        (group (:title . "Math")
               (:elements
                (query . music)))
        (group (:title . "Nature")
               (:elements
                (query . nature)))
        (group (:title . "Philosophy")
               (:elements
                (query . philosophy)))
        (group (:title . "Politics")
               (:elements
                (query . politics)))
        (group (:title . "Science")
               (:elements
                (query . science)))
        (group (:title . "SCP")
               (:elements
                (query . scp)))
        (group (:title . "Tech")
               (:elements
                (query . tech)))
        (group (:title . "Podcasts")
               (:elements
                (query . podcast)))
        (group (:title . "Pictures")
               (:elements
                (query . picture)))
        ;; ...
        (group (:title . "Miscellaneous")
               (:elements
                                       ;(group
                                       ; (:title . "Searches")
```

```
                                        ; (:elements
                                        ;  (search
                                        ;   (:filter . "@6-months-ago")
                                        ;   (:title . "Unread"))))
                (group
                 (:title . "Ungrouped")
                 (:elements :misc))))))
(global-set-key (kbd "s-e") 'elfeed-load-summary)

                                        ; Elfeed Youtube

                                        ; External youtube-dl library
(add-to-list 'load-path "~/.doom.d/lisp/youtube-dl-emacs")
(after-startup (require 'youtube-dl))
(setq youtube-dl-directory "~/elfeed-youtube"
      elfeed-enclosure-default-dir youtube-dl-directory
      youtube-dl-temp-directory "/tmp/elfeed-youtube"
      youtube-dl-program "yt-dlp"
      youtube-dl-arguments
      (nconc `("-f" "bestvideo[height<=1080]+bestaudio/best[height<=1080]"
               "--sponsorblock-remove" "default"
               "--prefer-free-formats"
               "--embed-subs"
               "--embed-metadata"
               "--embed-chapters"
               "--ffmpeg-location" "/home/user/.doom.d/ext/bin/"
               "--no-colors")
             youtube-dl-arguments))
                                        ; (setq youtube-dl-arguments nil)

(global-set-key (kbd "s-v") 'open-yt-dl-videos)
(global-set-key (kbd "s-V") 'open-yt-dl-temp-videos)

(defun open-yt-dl-videos ()
  (interactive)
  (find-file youtube-dl-directory))

(defun open-yt-dl-temp-videos ()
  (interactive)
  (find-file youtube-dl-temp-directory))


(cl-defun elfeed-show-youtube-dl (&key slow)
  "Download the current entry with youtube-dl."
  (interactive)
  (if (null (youtube-dl (elfeed-entry-link elfeed-show-entry)
                        :title (elfeed-entry-title elfeed-show-entry)
                        :slow slow))
      (message "Entry is not a YouTube link!")
    (message "Downloading %s" (elfeed-entry-title elfeed-show-entry))))


(cl-defun elfeed-search-youtube-dl (&key slow)
  "Download the current entry with youtube-dl."
  (interactive)
  (let ((entries (elfeed-search-selected)))
    (dolist (entry entries)
```

```
      (if (null (youtube-dl (elfeed-entry-link entry)
                            :title (elfeed-entry-title entry)
                            :slow slow))
          (message "Entry is not a YouTube link!")
        (message "Downloading %s" (elfeed-entry-title entry)))
      (elfeed-untag entry 'unread)
      (elfeed-search-update-entry entry)
      (unless (use-region-p) (forward-line)))))

(defun youtube-dl-list-url ()
  "Return url of item under point."
  (interactive)
  (let* ((n (1- (line-number-at-pos)))
         (item (nth n youtube-dl-items)))
    (when item
      (message (youtube-dl-item-destination item)))))
                                        ; Faces

(defun elfeed-summary-action-save-location (pos &optional event)
  (interactive "@d")
  (setq elfeed-summary--current-pos pos)
  (elfeed-summary--action pos event)
  )



(defun image-tooltip (img-path)
  "Display image at img-path as tooltip"
  (tooltip-mode 1)
  (tooltip-show
    (propertize "Look in minbuffer"
                'display (create-image img-path))))

(defun elfeed-search-thumbnail ()
  "Display the thumbnail of the currently selected video"
  (interactive)
  (mkdir elfeed-thumbnail-dir t)
  (let ((buffer (current-buffer))
        (entries (elfeed-search-selected)))
    (cl-loop for entry in entries
             when (elfeed-entry-link entry)
             do (let ((title (concat elfeed-thumbnail-dir (secure-hash 'sha224
             ↪  (elfeed-entry-title entry)))))
                  (if (file-exists-p (concat title ".jpg"))
                      (image-tooltip (concat title ".jpg"))
                    (youtube-dl-get-video-thumbnail it title (lambda (a)
                    ↪  (image-tooltip (concat title ".jpg"))))))
    (with-current-buffer buffer
      (mapc #'elfeed-search-update-entry entries)
      (unless (or elfeed-search-remain-on-entry (use-region-p))))))

(defun elfeed-wget-url ()
  "Wgets URL at point to elfeed video dir"
  (interactive)
  (let ((url (shr-url-at-point current-prefix-arg)))
    (add-to-list 'display-buffer-alist '("*Async Shell Command*"
    ↪  display-buffer-no-window (nil)))
```

```
    (async-shell-command (concat "wget -O " youtube-dl-directory "/\""
    ↪  (elfeed-entry-title elfeed-show-entry) "\".mp3 " url))))

(defun youtube-dl-move-temp ()
  "Moves content of elfeed video dir to temporary location"
  (interactive)
  (mkdir youtube-dl-temp-directory t)
  (add-to-list 'display-buffer-alist '("*Async Shell Command*"
  ↪  display-buffer-no-window (nil)))
  (async-shell-command (concat "mv " youtube-dl-directory "/* "
  ↪  youtube-dl-temp-directory "/")))

(use-package! elfeed-tube
  :after elfeed
  :demand t
  :config
  ;; (setq elfeed-tube-auto-save-p nil) ; default value
  ;; (setq elfeed-tube-auto-fetch-p t)  ; default value
  (elfeed-tube-setup)

  :bind (:map elfeed-show-mode-map
         ("F" . elfeed-tube-fetch)
         ([remap save-buffer] . elfeed-tube-save)
         :map elfeed-search-mode-map
         ("F" . elfeed-tube-fetch)
         ([remap save-buffer] . elfeed-tube-save)))

(use-package! elfeed-tube-mpv
  :bind
  ("C-c C-f" . elfeed-tube-mpv-follow-mode)
  ("C-c C-w" . elfeed-tube-mpv-where))

(setq elfeed-tube-captions-languages '("en" "de" "la" "english (auto
↪  generated)" "german (auto generated)")
      elfeed-tube-captions-chunk-time 60
      elfeed-tube-thumbnail-size 'large)

(defun elfeed-tube-mpv-open ()
  "Opens selected elfeed tube feed in mpv and activates follow mode"
  (interactive)
  (elfeed-tube-mpv-follow-mode 1)
  (elfeed-tube-mpv (point)))

(add-hook! 'elfeed-show-mode-hook '(lambda () (elfeed-tube-mpv-follow-mode 1)))

(provide 'elfeed-tweaks)
```

4.9 Interface

```
(load-module 'interface)
```

4.9.1 Requirements

4.9.2 Code

General settings

```
(setq-default
 x-stretch-cursor t)
(good-scroll-mode -1)
(setq-default word-wrap t)

(setq undo-limit 80000000                         ; Raise undo-limit to 80Mb
      evil-want-fine-undo t                       ; By default while in insert
      ↪  all changes are one big blob. Be more granular
      truncate-string-ellipsis "…"                ; Unicode ellispis are nicer
      ↪  than "...", and also save /precious/ space
      password-cache-expiry nil                   ; I can trust my computers
      ↪  ... can't I?
      scroll-margin 2)                            ; It's nice to maintain a
      ↪  little margin

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

Font

```
;;; Unicode emojis
(if (>= emacs-major-version 27)
    (set-fontset-font t '(#x1f000 . #x1faff)
                      (font-spec :family "Noto Color Emoji")))
(set-face-attribute
 'default nil :stipple nil :height 120 :width 'normal :inverse-video nil :box
 ↪  nil :strike-through nil :overline nil :underline nil :slant 'normal
 ↪  :weight 'normal :foundry "outline" :family "Source Code Pro for
 ↪  Powerline")
;;;; setting up composition functions for emoji modifiers
;;(dolist (items `(((?  . ?) [".[-]+" 0 font-shape-gstring])
;                 ((?  . ?) [".[ -]*" 0 font-shape-gstring])
;                 (?  ["[#*0-9] " 2 font-shape-gstring])
;                 ;; TODO: I can't make keycap sequences work because I
;                 ;; think they're trying to shape with the wrong font.
;                 ,@(mapcar (lambda (range) (list range [".?[-]?[ ]*?" 0
↪  font-shape-gstring]))
;                           (concatenate 'list "                "
;                                        '((?  . ?) (?  . ?) (?  . ?) (?  . ?)
;                                          (?  . ?) (?  . ?) (?  . ?) (?  . ?)
↪  (?  . ?)
;                                          (?  . ?) (?  . ?) (?  . ?) (?  . ?)
↪  (?  . ?)
;                                          (?  . ?) (?  . ?))) )
;                 (?  [".?[-]?[       -      -  ]*" 0 font-shape-gstring])
;                 ((?  . ?) [".?[-]?[       -      -  ]*" 0
↪  font-shape-gstring])
;                 ,@(mapcar (lambda (str) (list (elt str 0) (vector str 0
↪  'font-shape-gstring)))
```

```
;                          '(" " " " " " " " " " " " " " " " "))))
;  (set-char-table-range
;   composition-function-table
;   (car items)
;   (list (cadr items))))

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

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

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

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

Other

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

(global-set-key (kbd "<f5>") 'revert-buffer)

                                      ;(unless (string-match-p "^Power N/A"
                                      ↪  (battery))   ; On laptops...
                                      ;  (display-battery-mode 1))
                                      ↪  ; it's nice to know how much power
                                      ↪  you have
(global-subword-mode 1)                         ; Iterate through CamelCase
↪  words
(setq battery-mode-line-format "%t")
(display-battery-mode t)
(display-battery-mode -1)



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

```
(set-char-table-range composition-function-table ?f '(["\\(?:ff?[fijlt]\\)" 0
↪  font-shape-gstring]))
(set-char-table-range composition-function-table ?T '(["\\(?:Th\\)" 0
↪  font-shape-gstring]))

(after! centaur-tabs
  (centaur-tabs-mode -1)
  (setq centaur-tabs-height 12
        centaur-tabs-set-icons t
        centaur-tabs-modified-marker "o"
        centaur-tabs-close-button "×"
        centaur-tabs-set-bar 'above
        centaur-tabs-gray-out-icons 'buffer)
  (centaur-tabs-change-fonts "SourceCodePro" 100))

(defun cleanup-after-init ()
  (switch-to-buffer "*scratch*")
  (delete-other-windows)
  (kill-unwanted-buffers)
  )

(defun schedule-cleanup-after-init ()
  (run-at-time "1 sec" nil 'cleanup-after-init))

                                        ;(schedule-cleanup-after-init)

                                        ;(add-hook 'after-init-hook
                                        ↪  'schedule-cleanup-after-init)

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

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

(use-package! page-break-lines
  :commands page-break-lines-mode
  :init
  (autoload 'turn-on-page-break-lines-mode "page-break-lines")
  :config
  (setq page-break-lines-max-width fill-column)
  (map! :prefix "g"
        :desc "Prev page break" :nv "[" #'backward-page
        :desc "Next page break" :nv "]" #'forward-page))


(use-package! theme-magic
  :commands theme-magic-from-emacs
  :config
  (defadvice! theme-magic--auto-extract-16-doom-colors ()
    :override #'theme-magic--auto-extract-16-colors
    (list
     (face-attribute 'default :background)
     (doom-color 'error)
     (doom-color 'success)
     (doom-color 'type)
     (doom-color 'keywords)
```

```
     (doom-color 'constants)
     (doom-color 'functions)
     (face-attribute 'default :foreground)
     (face-attribute 'shadow :foreground)
     (doom-blend 'base8 'error 0.1)
     (doom-blend 'base8 'success 0.1)
     (doom-blend 'base8 'type 0.1)
     (doom-blend 'base8 'keywords 0.1)
     (doom-blend 'base8 'constants 0.1)
     (doom-blend 'base8 'functions 0.1)
     (face-attribute 'default :foreground))))

(run-with-idle-timer 0.1 nil (lambda () (add-hook 'doom-load-theme-hook
↪   'theme-magic-from-emacs)))

                                        ; Modern org mode
(global-org-modern-mode t)

;; Transparent scratch buffer
(defun buffer-empty-p (&optional buffer)
  (= (buffer-size buffer) 0))

(defun frame-trans-on ()
  (interactive)
  (set-frame-parameter (selected-frame) 'alpha '(0 0)))

(defun frame-trans-off ()
  (interactive)
  (set-frame-parameter (selected-frame) 'alpha '(100 100)))

(defun scratch-trans ()
  (setq my-buffer (get-buffer "*scratch*"))
  (cond ((eq my-buffer (window-buffer (selected-window)))
         (if (= (length (window-list)) 1) (frame-trans-on) (frame-trans-off)))
        ((get-buffer-window my-buffer)
         (frame-trans-off))
        (t
         (frame-trans-off))))

(add-hook 'window-configuration-change-hook 'scratch-trans)

;; Async shell commands without popup buffer
(defun async-shell-command-no-window
    (command)
  "Execute async shell command without popup buffer."
  (interactive)
  (let
      ((display-buffer-alist
        (list
         (cons
          "\\*Async Shell Command\\*.*"
          (cons #'display-buffer-no-window nil)))))
    (async-shell-command command)))
(provide 'interface)
```

4.10 EXWM Tweaks

```
(load-module 'exwm-tweaks)
```

4.10.1 Requirements

4.10.2 Code

```
;;; exwm-tweaks.el -*- lexical-binding: t; -*-
(use-package! exwm
  :config
  (setq mouse-autoselect-window t
        focus-follows-mouse t)
  (require 'exwm)
  (require 'exwm-config)
  (exwm-config-default)
  (require 'exwm-randr)

(when (string= (system-name) "astaroth")
  (setq exwm-randr-workspace-output-plist '(1 "DP-2-1" 2 "HDMI-2" 3 "DP-2-2" 4
  ↪  "eDP-1")))
(when (string= (system-name) "jarvis")
  (setq exwm-randr-workspace-output-plist '(1 "DisplayPort-0" 2 "DVI-0" 3
  ↪  "HDMI-0" 4 "eDP-1")))

  (add-hook 'exwm-randr-screen-change-hook
            (lambda ()
              (start-process-shell-command
               "xrandr" nil "xrandr --output eDP-1 --primary --mode 1920x1080
               ↪  --pos 1920x0 --rotate normal --output DP-1 --off --output
               ↪  HDMI-1 --off --output DP-2 --off --output HDMI-2 --mode
               ↪  1920x1080 --pos 0x0 --rotate normal")))

  (exwm-randr-enable)
  (winner-mode t)
  (require 'exwm-systemtray)
  (exwm-systemtray-enable)
  (define-key exwm-mode-map (kbd "C-c") nil)
  (setq exwm-input-simulation-keys
        '(([?\C-b] . [left])
          ([?\C-f] . [right])
          ([?\C-p] . [up])
          ([?\C-n] . [down])
          ([?\C-a] . [home])
          ([?\C-e] . [end])
          ([?\M-a] . [C-a])
          ([?\M-v] . [prior])
          ([?\C-d] . [delete])
          ([?\C-k] . [S-end delete])
          ([?\C-w] . [?\C-x])
          ([?\M-w] . [?\C-c])
          ([?\C-y] . [?\C-v])
          ;; search
          ([?\C-s] . [?\C-f])
          ([?\M-s] . [?\C-s])))
  (when (functionp 'exwm-enable-ido-workaround)
```

```
    (exwm-enable-ido-workaround))
  (with-eval-after-load 'ediff-wind
  (setq ediff-control-frame-parameters
        (cons '(unsplittable . t) ediff-control-frame-parameters)))

  (global-set-key (kbd "C-x C-c") 'save-buffers-kill-emacs)
;  (global-set-key (kbd "C-c m") 'toggle-maximize-buffer)


  (defun fullscreen ()
    (interactive)
    (if (eq major-mode 'exwm-mode)
      (call-interactively 'exwm-layout-toggle-fullscreen)
      (toggle-maximize-buffer)
      ))

;;; Make current buffer float
(defun toggle-float-buffer ()
  (interactive)
  (if (eq major-mode 'exwm-mode)
      (progn
      (call-interactively 'exwm-floating-toggle-floating)
      (call-interactively 'exwm-layout-hide-mode-line)
      )))


;;; Sometimes exwm fails to sets a buffer, so set it to scratch
;;; with a button press
(defun go-to-scratch ()
  (interactive)
  (message "%s" (selected-window))
  (switch-to-buffer "*scratch*"))

(defun go-to-scratch-other ()
  (interactive)
  (switch-to-buffer-other-frame "*scratch*"))

(setq save-temp-location "~/dox/temp-save/")
(defun save-buffer-temp ()
  (interactive)
  (let* ((s (buffer-string))
         (ss (split-string s " "))
         (nl (butlast ss (- (length ss) 5)))

         )
    (set-visited-file-name (concat save-temp-location (mapconcat '(lambda (x)
    ↪  (format "%s" x))  nl " ") ".org"))
    (save-buffer)
    )
  )

  (defun switchmonitor-next ()
    (interactive)
    (shell-command "xdotool mousemove_relative 1920 0"))

  (defun switchmonitor-prev ()
    (interactive)
```

```
    (shell-command "xdotool mousemove_relative -- -1920 0"))


  (setq exwm-workspace-number 9
        exwm-workspace-show-all-buffers t
        exwm-layout-show-all-buffers t
        exwm-manage-force-tiling t)
  (setq exwm-input-global-keys
      `(([?\s-f] . fullscreen)
        ([?\s-F] . toggle-maximize-buffer)
        ([?\s-g] . toggle-float-buffer)
        ([?\s-q] . kill-curr-buffer)
        ([?\s-n] . switchmonitor-next)
        ([?\s-p] . switchmonitor-prev)
        ;((kbd "s-<return>") . switchmonitor-prev)
        ,@(mapcar (lambda (i)
                    `(,(kbd (format "s-%d" i)) .
                      (lambda ()
                        (interactive)
                        (exwm-workspace-switch-create ,i))))
                  (number-sequence 0 9))))
  (add-hook 'exwm-manage-finish-hook
          (lambda ()
            (if (and exwm-class-name
                     (string= exwm-class-name "St"))
              (progn
                (exwm-input-release-keyboard))
              (progn))
            (exwm-layout-hide-mode-line)))

(setq exwm-input-prefix-keys
'(?\C-x ?\C-u ?\C-h ?\M-x ?\M-` ?\M-& ?\M-:))


(global-set-key (kbd "s-<f4>") 'go-to-scratch)
(global-set-key (kbd "s-S-<f4>") 'save-buffer-temp)
(require 'exwm-edit)
(defun ag-exwm/on-exwm-edit-compose ()
  (funcall 'org-mode))
(add-hook 'exwm-edit-compose-hook 'ag-exwm/on-exwm-edit-compose)


(add-hook 'exwm-update-title-hook
          (lambda ()
              (exwm-workspace-rename-buffer exwm-title))))

(setq exwm-manage-configurations
      '(((or (string-equal exwm-class-name "Nm-applet")
             (string-equal exwm-class-name "Surf")
             (string-equal exwm-class-name "Steam")
             (not (message exwm-class-name)))
          floating t
          floating-mode-line nil
;          width 0.4
;          height 0.4
          )
        ((equal exwm-window-type xcb:Atom:_NET_WM_WINDOW_TYPE_DIALOG)
```

```
         floating t
         floating-mode-line nil)
        ))

(defun exwm-floating--set-floating (id)
  "Make window ID floating."
  (let ((window (get-buffer-window (exwm--id->buffer id))))
    (when window
      ;; Hide the non-floating X window first.
      (set-window-buffer window (other-buffer nil t))))
  (let* ((original-frame (buffer-local-value 'exwm--frame
                                             (exwm--id->buffer id)))
         ;; Create new frame
         (frame (with-current-buffer
                    (or (get-buffer "*scratch*")
                        (progn
                          (set-buffer-major-mode
                           (get-buffer-create "*scratch*"))
                          (get-buffer "*scratch*")))
                  (make-frame
                   `((minibuffer . ,(minibuffer-window exwm--frame))
                     (left . ,(* window-min-width -10000))
                     (top . ,(* window-min-height -10000))
                     (width . ,window-min-width)
                     (height . ,window-min-height)
                     (unsplittable . t))))) ;and fix the size later
         (outer-id (string-to-number (frame-parameter frame 'outer-window-id)))
         (window-id (string-to-number (frame-parameter frame 'window-id)))
         (frame-container (xcb:generate-id exwm--connection))
         (window (frame-first-window frame)) ;and it's the only window
         (x (slot-value exwm--geometry 'x))
         (y (slot-value exwm--geometry 'y))
         (width (slot-value exwm--geometry 'width))
         (height (slot-value exwm--geometry 'height)))
    ;; Force drawing menu-bar & tool-bar.
    (redisplay t)
    (exwm-workspace--update-offsets)
    (exwm--log "Floating geometry (original): %dx%d%+d%+d" width height x y)
    ;; Save frame parameters.
    (set-frame-parameter frame 'exwm-outer-id outer-id)
    (set-frame-parameter frame 'exwm-id window-id)
    (set-frame-parameter frame 'exwm-container frame-container)
    (set-frame-parameter frame 'alpha 10)
    ;; Fix illegal parameters
    ;; FIXME: check normal hints restrictions
    (let* ((workarea (elt exwm-workspace--workareas
                          (exwm-workspace--position original-frame)))
           (x* (aref workarea 0))
           (y* (aref workarea 1))
           (width* (aref workarea 2))
           (height* (aref workarea 3)))
      ;; Center floating windows
      (when (and (or (= x 0) (= x x*))
                 (or (= y 0) (= y y*)))
        (let ((buffer (exwm--id->buffer exwm-transient-for))
              window edges)
          (when (and buffer (setq window (get-buffer-window buffer)))
```

```
          (setq edges (window-inside-absolute-pixel-edges window))
          (unless (and (<= width (- (elt edges 2) (elt edges 0)))
                       (<= height (- (elt edges 3) (elt edges 1))))
            (setq edges nil)))
        (if edges
            ;; Put at the center of leading window
            (setq x (+ x* (/ (- (elt edges 2) (elt edges 0) width) 2))
                  y (+ y* (/ (- (elt edges 3) (elt edges 1) height) 2)))
          ;; Put at the center of screen
          (setq x (/ (- width* width) 2)
                y (/ (- height* height) 2)))))
    (if (> width width*)
        ;; Too wide
        (progn (setq x x*
                     width width*))
      ;; Invalid width
      (when (= 0 width) (setq width (/ width* 2)))
      ;; Make sure at least half of the window is visible
      (unless (< x* (+ x (/ width 2)) (+ x* width*))
        (setq x (+ x* (/ (- width* width) 2)))))
    (if (> height height*)
        ;; Too tall
        (setq y y*
              height height*)
      ;; Invalid height
      (when (= 0 height) (setq height (/ height* 2)))
      ;; Make sure at least half of the window is visible
      (unless (< y* (+ y (/ height 2)) (+ y* height*))
        (setq y (+ y* (/ (- height* height) 2)))))
    ;; The geometry can be overridden by user options.
    (let ((x** (plist-get exwm--configurations 'x))
          (y** (plist-get exwm--configurations 'y))
          (width** (plist-get exwm--configurations 'width))
          (height** (plist-get exwm--configurations 'height)))
      (if (integerp x**)
          (setq x (+ x* x**))
        (when (and (floatp x**)
                   (>= 1 x** 0))
          (setq x (+ x* (round (* x** width*))))))
      (if (integerp y**)
          (setq y (+ y* y**))
        (when (and (floatp y**)
                   (>= 1 y** 0))
          (setq y (+ y* (round (* y** height*))))))
      (if (integerp width**)
          (setq width width**)
        (when (and (floatp width**)
                   (> 1 width** 0))
          (setq width (max 1 (round (* width** width*))))))
      (if (integerp height**)
          (setq height height**)
        (when (and (floatp height**)
                   (> 1 height** 0))
          (setq height (max 1 (round (* height** height*)))))))
    (exwm--set-geometry id x y nil nil)
    (xcb:flush exwm--connection)
    (exwm--log "Floating geometry (corrected): %dx%d%+d%+d" width height x y)
```

```
    ;; Fit frame to client
    ;; It seems we have to make the frame invisible in order to resize it
    ;; timely.
    ;; The frame will be made visible by `select-frame-set-input-focus'.
    (make-frame-invisible frame)
    (let* ((edges (window-inside-pixel-edges window))
           (frame-width (+ width (- (frame-pixel-width frame)
                                    (- (elt edges 2) (elt edges 0)))))
           (frame-height (+ height (- (frame-pixel-height frame)
                                      (- (elt edges 3) (elt edges 1)))
                            ;; Use `frame-outer-height' in the future.
                            exwm-workspace--frame-y-offset))
           (floating-mode-line (plist-get exwm--configurations
                                          'floating-mode-line))
           (floating-header-line (plist-get exwm--configurations
                                            'floating-header-line))
           (border-pixel (exwm--color->pixel exwm-floating-border-color)))
      (if floating-mode-line
          (setq exwm--mode-line-format (or exwm--mode-line-format
                                           mode-line-format)
                mode-line-format floating-mode-line)
        (if (and (not (plist-member exwm--configurations 'floating-mode-line))
                 exwm--mwm-hints-decorations)
            (when exwm--mode-line-format
              (setq mode-line-format exwm--mode-line-format))
          ;; The mode-line need to be hidden in floating mode.
          (setq frame-height (- frame-height (window-mode-line-height
                                              (frame-root-window frame)))
                exwm--mode-line-format (or exwm--mode-line-format
                                           mode-line-format)
                mode-line-format nil)))
      (if floating-header-line
          (setq header-line-format floating-header-line)
        (if (and (not (plist-member exwm--configurations
                                    'floating-header-line))
                 exwm--mwm-hints-decorations)
            (setq header-line-format nil)
          ;; The header-line need to be hidden in floating mode.
          (setq frame-height (- frame-height (window-header-line-height
                                              (frame-root-window frame)))
                header-line-format nil)))
      (set-frame-size frame frame-width frame-height t)
      ;; Create the frame container as the parent of the frame.
      (xcb:+request exwm--connection
          (make-instance 'xcb:CreateWindow
                         :depth 0
                         :wid frame-container
                         :parent exwm--root
                         :x x
                         :y (- y exwm-workspace--window-y-offset)
                         :width width
                         :height height
                         :border-width
                         (with-current-buffer (exwm--id->buffer id)
                           (let ((border-witdh (plist-get exwm--configurations
                                                          'border-width)))
                             (if (and (integerp border-witdh)
```

```
                                  (>= border-witdh 0))
                             border-witdh
                           exwm-floating-border-width)))
                    :class xcb:WindowClass:InputOutput
                    :visual 0
                    :value-mask (logior xcb:CW:BackPixmap
                                        (if border-pixel
                                            xcb:CW:BorderPixel 0)
                                        xcb:CW:OverrideRedirect)
                    :background-pixmap xcb:BackPixmap:ParentRelative
                    :border-pixel border-pixel
                    :override-redirect 1))
  (xcb:+request exwm--connection
      (make-instance 'xcb:ewmh:set-_NET_WM_NAME
                     :window frame-container
                     :data
                     (format "EXWM floating frame container for 0x%x" id)))
  ;; Map it.
  (xcb:+request exwm--connection
      (make-instance 'xcb:MapWindow :window frame-container))
  ;; Put the X window right above this frame container.
  (xcb:+request exwm--connection
      (make-instance 'xcb:ConfigureWindow
                     :window id
                     :value-mask (logior xcb:ConfigWindow:Sibling
                                         xcb:ConfigWindow:StackMode)
                     :sibling frame-container
                     :stack-mode xcb:StackMode:Above)))
;; Reparent this frame to its container.
(xcb:+request exwm--connection
    (make-instance 'xcb:ReparentWindow
                   :window outer-id :parent frame-container :x 0 :y 0))
(exwm-floating--set-allowed-actions id nil)
(xcb:flush exwm--connection)
;; Set window/buffer
(with-current-buffer (exwm--id->buffer id)
  (setq window-size-fixed exwm--fixed-size
        exwm--floating-frame frame)
  ;; Do the refresh manually.
  (remove-hook 'window-configuration-change-hook #'exwm-layout--refresh)
  (set-window-buffer window (current-buffer)) ;this changes current buffer
  (add-hook 'window-configuration-change-hook #'exwm-layout--refresh)
  (set-window-dedicated-p window t)
  (exwm-layout--show id window))
(with-current-buffer (exwm--id->buffer id)
  (if (exwm-layout--iconic-state-p id)
      ;; Hide iconic floating X windows.
      (exwm-floating-hide)
    (with-selected-frame exwm--frame
      (exwm-layout--refresh)))
  (select-frame-set-input-focus frame))
;; FIXME: Strangely, the Emacs frame can move itself at this point
;;        when there are left/top struts set.  Force resetting its
;;        position seems working, but it'd better to figure out why.
;; FIXME: This also happens in another case (#220) where the cause is
;;        still unclear.
(exwm--set-geometry outer-id 0 0 nil nil)
```

```
    (xcb:flush exwm--connection))
  (with-current-buffer (exwm--id->buffer id)
    (run-hooks 'exwm-floating-setup-hook))
  ;; Redraw the frame.
  (redisplay t))

;; Additional commands that should also work in exwm
(exwm-input-set-key (kbd "s-<return>") (lambda () (interactive) (+vterm/toggle
↪  nil)))
(exwm-input-set-key (kbd "s-e") (lambda () (interactive)
↪  (elfeed-load-summary)))
(exwm-input-set-key (kbd "s-v") (lambda () (interactive) (open-yt-dl-videos)))
(exwm-input-set-key (kbd "s-r") (lambda () (interactive) (progn
  (+vterm/here t)
  (vterm-send-string "cd /home/user/dox/install/rosarium && cargo run\n")
)))
(exwm-input-set-key (kbd "s-<f4>") (lambda () (interactive) (go-to-scratch)))
(exwm-input-set-key (kbd "s-<left>") (lambda () (interactive) (winner-undo)))
(exwm-input-set-key (kbd "s-<right>") (lambda () (interactive) (winner-undo)))
(exwm-input-set-key (kbd "s-a") (lambda () (interactive) (org-agenda-list)))
(exwm-input-set-key (kbd "s-m") (lambda () (interactive) (mu4e--goto-inbox)))
(exwm-input--update-global-prefix-keys)

;; Wallpaper
(defmacro ifdirexists (dir &rest actions)
  "Execute functions taking dir as an argument if dir exists"
  `(when (file-exists-p ,dir)
     ((lambda (dir)
       ,@actions) ,dir)))

(ifdirexists "/home/user/dox/wallpapers"
             (setq wallpaper-cycle-directory dir)
             (wallpaper-set-wallpaper))

(provide 'exwm-tweaks)
```

4.11 General

```
(load-module 'general)
```

4.11.1 Requirements

4.11.2 Code

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

(setq mouse-autoselect-window t
      focus-follows-mouse t)

;; Disable backup files
(setq make-backup-files nil)
(setq auto-save-default nil)

;; Delete selection when pasting
(delete-selection-mode 1)
```

```
;; Save the session
(desktop-save-mode 1)
(setq desktop-restore-eager 10)
;; Save last visited place in files
(setq-default save-place t)
(setq save-place-file "~/.emacs.d/etc/saveplace")

(provide 'general)
```

4.12 Navigation

```
(load-module 'navigation)
```

4.12.1 Requirements

4.12.2 Code

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

;; Kill minibuffer when loosing focus
(defun stop-using-minibuffer ()
  "kill the minibuffer"
  (when (and (>= (recursion-depth) 1) (active-minibuffer-window))
    (abort-recursive-edit)))

(add-hook 'mouse-leave-buffer-hook 'stop-using-minibuffer)
(setq doom-fallback-buffer-name "  Doom"
      +doom-dashboard-name "  Doom")

(map! :map +doom-dashboard-mode-map
      :ne "f" #'find-file
      :ne "r" #'consult-recent-file
      :ne "p" #'doom/open-private-config
      :ne "c" (cmd! (find-file (expand-file-name "config.org"
      ↪  doom-private-dir)))
      :ne "." (cmd! (doom-project-find-file "~/.config/")) ; . for dotfiles
      :ne "b" #'+vertico/switch-workspace-buffer
      :ne "B" #'consult-buffer
      :ne "q" #'save-buffers-kill-terminal)

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

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

```
;           (unless (string= "-" project-name)
;             (format (if (buffer-modified-p)  "  %s" "   %s")
↪  project-name))))))

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

(defun ignore-dired-buffers-ivy (str)
  "Return non-nil if STR names a Dired buffer.
      This function is intended for use with `ivy-ignore-buffers'."
  (let ((buf (get-buffer str)))
    (and buf (eq (buffer-local-value 'major-mode buf) 'dired-mode))))

(defun ignore-help-buffers-ivy (str)
  "Return non-nil if STR names a help buffer (buffers starting and ending with
      *)
      This function is intended for use with `ivy-ignore-buffers'."
  (and
   (s-starts-with-p "*" str)
   (s-ends-with-p "*" str)))

(defun ignore-unwanted-buffers-ivy (str)
  "Return non-nil if STR names a Dired buffer.
      This function is intended for use with `ivy-ignore-buffers'."
  (or
   (string-equal "elfeed.org" str)
   (member str (map 'list 'file-name-nondirectory org-agenda-files))
   ))

(with-eval-after-load 'ivy
  (progn
  (add-to-list 'ivy-ignore-buffers #'ignore-dired-buffers-ivy)
  (add-to-list 'ivy-ignore-buffers #'ignore-help-buffers-ivy)
  (add-to-list 'ivy-ignore-buffers #'ignore-unwanted-buffers-ivy)
  ))

;;; Switch window
(use-package! switch-window
  :config
  (setq switch-window-multiple-frames nil)
  (setq switch-window-input-style 'minibuffer)
  (setq switch-window-increase 4)
  (setq switch-window-threshold 2)
  (setq switch-window-shortcut-style 'qwerty)
  (setq switch-window-qwerty-shortcuts
        '("j" "k" "l" "a" "s" "d" "f")) ; ö does not work without pressing RET
  :bind
  ([remap other-window] . switch-window))

;;; Temporarily maximize current buffer
(defun toggle-maximize-buffer () "Maximize buffer"
       (interactive)
       (if (= 1 (length (window-list)))
           (jump-to-register '_)
         (progn
           (window-configuration-to-register '_)
           (delete-other-windows))))
```

```
(defun transparent-buffer-advice
  (orig-fun &rest args)
  (shell-command "transset -p 1") ; 0.3
  (let
      ((res
        (apply orig-fun args)))
    (shell-command "transset -p 1")
    res))


;;; kill current buffer
(defun kill-curr-buffer ()
  (interactive)
  (if (not (string-equal (buffer-name (current-buffer)) "*scratch*"))
      (kill-buffer (current-buffer))
    (bury-buffer)
    (switch-to-buffer "*scratch*")
  ))

;;; move to start and end of buffer
(global-set-key (kbd "M-n") 'end-of-buffer)
(global-set-key (kbd "M-p") 'beginning-of-buffer)

;; Kill all buffers
(defun close-all-buffers ()
  (interactive)
  (mapc 'kill-buffer (buffer-list)))
(global-set-key (kbd "C-x C-k k") 'close-all-buffers)

;; Kill unwanted buffers
(defun kill-if-unwanted (buffer)
  (let ((b (buffer-name buffer))
        (bfn (buffer-file-name buffer))
        (bmm (buffer-local-value 'major-mode buffer))
        (unwanted-buffers '(
                            "*Messages*"
                            "*Backtrace*"
                            "*Help*"
                            "*Warnings*"
                            "*Compile-Log*"
                            "*elfeed-log*"
                            "*system-packages*"
                            "*Async Shell Command*"
                            "*Flycheck errors*"
                            "*Flycheck error messages*"
                            "*Flymake log*"
                            "*Calendar*"
                            "*XELB-DEBUG*"
                            "*Read-Aloud Log*"
                            "*elfeed-search*"
                            "elfeed.org"
                            )))
    (when (or (member b unwanted-buffers)
          (member bfn (mapcar 'expand-file-name org-agenda-files))
          (eq 'dired-mode bmm)
          (string-match "^\*tramp.*\*$" b)
```

```
          (string-match "\.png$" b)
          (string-match "\.jpg$" b)
          (string-match "\.jpeg$" b)
          (string-match "\.gif$" b)
          (string-match "\.log$" b)
          (string-match "^_region_.tex$" b)
          (string-match "^\*helpful .*\*" b)
          (string-match "- Thunar" b)
          (string-match "^magit" b)
          (string-match "^\*.*\*$" b))
      (kill-buffer b))))

(defun kill-unwanted-buffers ()
  (interactive)
  (mapc 'kill-if-unwanted (buffer-list)))

(global-set-key (kbd "C-x k") 'kill-unwanted-buffers)

;;; Window splitting
(defun split-and-follow-horizontally ()
  (interactive)
  (split-window-below)
  (balance-windows)
  (other-window 1))
(global-set-key (kbd "C-x 2") 'split-and-follow-horizontally)

(defun split-and-follow-vertically ()
  (interactive)
  (split-window-right)
  (balance-windows)
  (other-window 1))
(global-set-key (kbd "C-x 3") 'split-and-follow-vertically)

(defun kill-and-balance ()
  (interactive)
  (delete-window)
  (balance-windows))
(global-set-key (kbd "C-x 0") 'kill-and-balance)



;;; Subword moving
(global-subword-mode 1)

;;; Cycle though tabs
;(global-set-key (kbd "C-<tab>") 'next-buffer)
;(global-set-key (kbd "<C-iso-lefttab>") 'previous-buffer)

;;; Winum mode for easy moving through windows
(use-package! winum
  :config
  (setq winum-auto-setup-mode-line nil)
  (winum-mode t)
  :bind (
         ("s-=" . winum-select-window-0)
         ("s-!" . winum-select-window-1)
         ("s-\"" . winum-select-window-2)
```

```
        ("s-§" . winum-select-window-3)
        ("s-$" . winum-select-window-4)
        ("s-%" . winum-select-window-5)
        ("s-&" . winum-select-window-6)
        ("s-/" . winum-select-window-7)
        ("s-(" . winum-select-window-8)
        ("s-)" . winum-select-window-9)
        ("s-°" . winum-select-window-by-number))
  )

(provide 'navigation)
```

4.13 Shortcuts

```
(load-module 'shortcuts)
```

4.13.1 Requirements

4.13.2 Code

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

;;;; Copy-whole-line
(defun copy-whole-line ()
  (interactive)
  (save-excursion
    (kill-ring-save (point-at-bol) (point-at-eol))))

(global-set-key (kbd "C-c w l") 'copy-whole-line)

;;;; Copy-line-above and copy-line-below (and paste)
(defun copy-line-above ()
  (interactive)
  (save-excursion
    (evil-previous-visual-line)
    (copy-whole-line)
    (evil-next-visual-line)
    (evil-paste-after 1)))

(global-set-key (kbd "C-c l a") 'copy-line-above)

(defun copy-line-below ()
  (interactive)
  (save-excursion
    (evil-next-visual-line)
    (copy-whole-line)
    (evil-previous-visual-line)
    (evil-paste-after 1)))

(global-set-key (kbd "C-c l b") 'copy-line-below)

;;;; Duplicate line
(defun duplicate-line ()
  (interactive)
  (save-excursion
```

```
    (evil-open-below 1)
    (copy-line-above))
  (evil-next-visual-line)
  (evil-normal-state)
  (evil-forward-char))

(global-set-key (kbd "C-c l l") 'duplicate-line)

;;; Kill word improved
;;; normal kill-word kills forward, but not whole word. This fixes that
(defun kill-whole-word ()
  (interactive)
  (backward-word)
  (kill-word 1))
(global-set-key (kbd "C-c k w") 'kill-whole-word)

;;; File shortcuts
;; Note taken on [2018-08-03 Fri 18:19]
(global-unset-key (kbd "C-c z"))

(defadvice goto-line (after unfold-tree activate)
  (when (outline-invisible-p)
    (save-excursion
      (outline-previous-visible-heading 1)
      (org-fold-show-subtree))))


(defun agenda-today ()
  (interactive)
  (goto-line (string-to-number (shell-command-to-string
  ↪  "~/.scripts/agendatoday")))
  (org-reveal 1))

(defun dailyplan()
  (interactive)
  (find-file (shell-command-to-string "date
  ↪  +'/home/user/dp/dailyplan/%Y/%Y-%m/%Y-%m-%d.org' | tr -d '\n'"))
  (end-of-buffer))

;(add-hook 'find-file-hook 'dailyplan-hook)
;(defun dailyplan-hook ()
;  (when (string= (buffer-file-name) "dailyplan.org")
;    (agenda-today)))

(defun books()
  (interactive)
  (find-file "~/pCloudDrive/agenda/books.org"))

(defun thesis()
  (interactive)
  (find-file "~/nextcloud/bachelor/thesis/structure.tex"))

(defun projects()
  (interactive)
  (find-file "~/pCloudDrive/agenda/currprojects.org"))

(defun movies()
```

```
  (interactive)
  (find-file "~/pCloudDrive/agenda/movies.org"))

(defun reviews()
  (interactive)
  (find-file "~/pCloudDrive/agenda/reviews/2018.org")
  (split-and-follow-vertically)
  (find-file "~/pCloudDrive/agenda/reviews/template.org"))

(defun ceres()
  (interactive)
  (find-file "/ssh:user@sermak.xyz:~"))

(defun ceres-root()
  (interactive)
  (find-file "/ssh:user@sermak.xyz|sudo:root@sermak.xyz:/"))

(defun jarvis()
  (interactive)
  (find-file "/ssh:user@sermak.xyz|sudo:root@jarvis:/"))

(defun jarvis-root()
  (interactive)
  (find-file "/ssh:user@sermak.xyz|ssh:user@jarvis:/"))

(global-set-key (kbd "C-c z d") 'dailyplan)
(global-set-key (kbd "C-c z b") 'books)
(global-set-key (kbd "C-c z m") 'movies)
(global-set-key (kbd "C-c z r") 'reviews)
(global-set-key (kbd "C-c z p") 'projects)
(global-set-key (kbd "C-c z t") 'thesis)
(global-set-key (kbd "C-c z e") 'mu4e)
(global-set-key (kbd "C-c z s c") 'ceres)
(global-set-key (kbd "C-c z s r") 'ceres-root)

;; University
(setq uni-base-folder "/mnt/server-de/mnt/backup/backups/pre_master/Uni")

(defun open-uni-folder (folder)
  "Mount/Open university folder specified as FOLDER."
  (when (not (file-exists-p uni-base-folder))
    (shell-command "sshfs sermak.xyz:/ /mnt/server-de"))
  (find-file (f-join uni-base-folder folder)))

(defmacro uni-folder-shortcut (shortcut folder funcname)
  `(progn
  (defun ,funcname ()
    ,(format "Open Uni/%s" folder)
     (interactive)
     (open-uni-folder ,folder))
  (global-set-key (kbd (concat "C-c u " ,shortcut)) ',funcname)))

(uni-folder-shortcut "u" "" uni)
(uni-folder-shortcut "6" "6" uni6)
(uni-folder-shortcut "l 1" "6/Orthodox Liturgy I" orthodox-liturgy-1)
(uni-folder-shortcut "l 2" "6/Orthodox Liturgy II" orthodox-liturgy-2)
(uni-folder-shortcut "h 1" "6/Orthodox History I" orthodox-history-1)
```

```
(uni-folder-shortcut "h 2" "6/Orthodox History II" orthodox-history-2)
(uni-folder-shortcut "t 1" "6/Orthodox Theology I" orthodox-theology-1)
(uni-folder-shortcut "t 2" "6/Orthodox Theology II" orthodox-theology-2)
(uni-folder-shortcut "s" "6/Orthodox Scripture" orthodox-scripture)
(uni-folder-shortcut "a" "6/Orthodox Anthropology" orthodox-anthropology)
(uni-folder-shortcut "w" "6/War and Statesbuilding in Afghanistan"
↪  war-and-statesbuilding)
(uni-folder-shortcut "e" "6/Exegesis of the Old and New Testament" exegesis)
(uni-folder-shortcut "m" "6/Monte Carlo Techniques" monte-carlo)

;; Tones
(global-set-key (kbd "C-c -") (lambda () (interactive) (insert "")))
(global-set-key (kbd "C-c ^") (lambda () (interactive) (insert "")))
;;; Chinese tones
(global-set-key (kbd "C-c 1") (lambda () (interactive) (insert "")))
(global-set-key (kbd "C-c 2") (lambda () (interactive) (insert "")))
(global-set-key (kbd "C-c 3") (lambda () (interactive) (insert "")))
(global-set-key (kbd "C-c 4") (lambda () (interactive) (insert "")))

;;; Rectangle mark mode
(global-set-key (kbd "C-ö") (lambda () (interactive) (rectangle-mark-mode)))
;;; Sudo-edit
(use-package! sudo-edit
  :bind ("C-c s" . sudo-edit))

(defun rededicate-window ()
  "Toggles window dedication in the selected window."
  (interactive)
  (let ((dedication (not (window-dedicated-p (selected-window)))))
    (message (format "%s" dedication))
    (set-window-dedicated-p (selected-window) dedication)))

(global-set-key (kbd "s-<return>") (lambda () (interactive) (+vterm/toggle
↪  nil)))

; Org agenda

(defun agenda-folder ()
  (interactive)
  (find-file "/home/user/sync/agenda/"))

(defun agenda-uni ()
  (interactive)
  (find-file "/home/user/sync/agenda/uni.org"))

(defun agenda-personal ()
  (interactive)
  (find-file "/home/user/sync/agenda/personal.org"))

(global-set-key (kbd "C-c a a") 'agenda-folder)
(global-set-key (kbd "C-c a u") 'agenda-uni)
(global-set-key (kbd "C-c a p") 'agenda-personal)

; Books

(defun books ()
  (interactive)
```

```
  (find-file "/home/user/dox/books/"))

(global-set-key (kbd "C-c b") 'books)

(provide 'shortcuts)
```

4.14 Config visit

```
(load-module 'config-visit)
```

4.14.1 Requirements

4.14.2 Code

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

(setq module-dir (concat doom-private-dir "modules/"))

(setq-default custom-file (expand-file-name ".custom.el" doom-private-dir))
(when (file-exists-p custom-file)
  (load custom-file))

(defun config-visit ()
  (interactive)
  (find-file (concat doom-private-dir "config.org")))

(defun init-visit ()
  (interactive)
  (find-file (concat doom-private-dir "init.el")))

(defun packages-visit ()
  (interactive)
  (find-file (concat doom-private-dir "packages.el")))

(defun module-visit ()
  (interactive)
  (find-file module-dir))

(defun recompile-modules ()
  (interactive)
  (digit-argument nil)
  (byte-recompile-directory module-dir 0))

(defun config-reload ()
  (interactive)
  ;;(recompile-modules)
  (org-babel-tangle-file (concat doom-private-dir "config.org"))
  (load-file (expand-file-name (concat doom-private-dir "config.el"))))

(global-set-key (kbd "C-c e c") 'config-visit)
(global-set-key (kbd "C-c e p") 'packages-visit)
(global-set-key (kbd "C-c e i") 'init-visit)
(global-set-key (kbd "C-c e m") 'module-visit)
(global-set-key (kbd "C-c r") 'config-reload)
```

```
(provide 'config-visit)
```

4.15 Search

```
(load-module 'search)
```

4.15.1 Requirements

4.15.2 Code

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

;; Swiper / Ivy / Counsel
;;  Swiper gives us a really efficient incremental search with regular
↪  expressions
;;  and Ivy / Counsel replace a lot of ido or helms completion functionality
(use-package! counsel
  :bind
  (("M-y" . counsel-yank-pop)
   :map ivy-minibuffer-map
   ("M-y" . ivy-next-line)))

(use-package! ivy
  :diminish (ivy-mode)
  :bind (("C-x C-b" . ivy-icon-switch-buffer))
  :config
  (ivy-mode 1)
  (setq ivy-use-virtual-buffers t
        ivy-count-format "%d/%d "
        ivy-height 20
        enable-recursive-minibuffers t
        ivy-display-style 'fancy))

(use-package! all-the-icons-ibuffer
  :init (all-the-icons-ibuffer-mode 1))

(defun ivy-icon-switch-buffer ()
  "ivy-switch-buffer with icons"
  (interactive)
  (condition-case nil
      (all-the-icons-ivy-setup))

(defun all-the-icons-ivy--buffer-transformer (b s)
  "Return a candidate string for buffer B named S preceded by an icon.
      Try to find the icon for the buffer's B `major-mode'.
      If that fails look for an icon for the mode that the `major-mode' is
  ↪  derived from."
  (let ((mode (buffer-local-value 'major-mode b))
        (buffname (replace-regexp-in-string "<.*>$" "" s)))
    (format (concat "%s" all-the-icons-spacer "%s")
            (propertize "\t" 'display (or
                                       (all-the-icons-ivy--icon-for-mode mode)
                                       (all-the-icons-ivy--icon-for-mode (get
                                       ↪  mode 'derived-mode-parent))
```

```
                                        (all-the-icons-ivy--icon-for-firefox
                                        ↪  mode buffname)
                                        (all-the-icons-ivy--icon-for-tor mode
                                        ↪  buffname)
                                        (all-the-icons-ivy--icon-for-exwm mode
                                        ↪  buffname)
                                        (funcall

                                         ↪  all-the-icons-ivy-family-fallback-for-buffer

                                         ↪  all-the-icons-ivy-name-fallback-for-buffer)))
            (all-the-icons-ivy--buffer-propertize b s))))


  (ivy-switch-buffer))

(setq all-the-icons-ivy-file-commands '(counsel-find-file counsel-file-jump
↪  counsel-recentf counsel-projectile-find-file counsel-projectile-find-dir))

;; Overwrite some stuff for exwm and icons in Firefox
(defun all-the-icons-ivy--icon-for-firefox (mode buffname)
  "Apply `all-the-icons-icon-for-url' on Firefox window in exwm-mode.
     Assuming that url is in title like in Keepass Helper extension."
  (if (string-equal (format "%s" mode) "exwm-mode")
      (let ((bnl (split-string buffname " - "))
            (fnl (split-string buffname " - ")))
            (let ((browser (format "%s" (last fnl))))
      (if (or (string-equal browser "(Mozilla Firefox)") (string-equal browser
      ↪  "(Mozilla Firefox (Private Browsing))"))
          (all-the-icons-faicon "firefox" :face 'all-the-icons-red)
        )))))

;; Overwrite some stuff for exwm and icons in Tor Browser
(defun all-the-icons-ivy--icon-for-tor (mode buffname)
  "Apply youtube icon on Tor Browser window in exwm-mode.
     Not assuming that url is in title like in Keepass Helper extension, for
  ↪  privacy."
  (if (string-equal (format "%s" mode) "exwm-mode")
      (let ((bnl (split-string buffname " - ")))
        (if (string-equal (format "%s" (last bnl)) "(Tor Browser)")
            (if (string-equal (format "%s" (last bnl 2)) "(YouTube Tor
            ↪  Browser)")
                (all-the-icons-icon-for-url "youtube.com" :face
                ↪  'all-the-icons-red)
              (all-the-icons-faicon "user-secret" :face 'all-the-icons-red)
             )))))

;; Overwrite some stuff for exwm
(defun all-the-icons-ivy--icon-for-exwm (mode buffname)
  "Hard-code some icons for common programs."
  (if (string-equal (format "%s" mode) "exwm-mode")
      (cond ((string-prefix-p "Signal" buffname)
             (all-the-icons-faicon "comment" :face 'all-the-icons-blue-alt))
            ((string-prefix-p "Skype" buffname)
             (all-the-icons-faicon "skype" :face 'all-the-icons-blue))
            ((string-suffix-p " - Discord" buffname)
             (all-the-icons-faicon "simplybuilt" :face 'all-the-icons-purple))
```

```
            ((string-prefix-p "OBS" buffname)
             (all-the-icons-faicon "video-camera" :face
             ↪  'all-the-icons-purple-alt))
            ((string-equal "Volume Control" buffname)
             (all-the-icons-faicon "volume-up" :face
             ↪  'all-the-icons-purple-alt))
            ((file-directory-p buffname)
             (all-the-icons-faicon "folder-open" :face 'all-the-icons-yellow))
            ((string-suffix-p " - mpv" buffname)
             (all-the-icons-faicon "play" :face 'all-the-icons-orange))
            ((string-suffix-p "\.java" buffname)
             (all-the-icons-alltheicon "java" :face 'all-the-icons-orange))
            ((or(string-equal "st" buffname) (string-prefix-p (concat
            ↪  (user-login-name) "@") buffname) (string-prefix-p "root@"
            ↪  buffname))
             (all-the-icons-faicon "terminal" :face 'all-the-icons-green))
            )))


(use-package! swiper
  :bind (("C-s" . swiper)
         ("C-r" . swiper)
         ("C-c C-r" . ivy-resume)
         ("M-x" . counsel-M-x)
         ("C-x C-f" . counsel-find-file))
  :config
  (progn
    (ivy-mode 1)
    (setq ivy-use-virtual-buffers t)
    (setq ivy-display-style 'fancy)
    (define-key read-expression-map (kbd "C-r") 'counsel-expression-history)
    ))

;; IDO
;;; enable ido mode
(setq ido-enable-flex-matching nil)
(setq ido-create-new-buffer 'always)
(setq ido-everywhere t)
(ido-mode 1)

;;; ido vertical
(use-package! ido-vertical-mode
  :init
  (ido-vertical-mode 1)
  (setq ido-vertical-define-keys 'C-n-and-C-p-only))

;;; smex
(use-package! amx
  :init (amx-initialize)
  :config
  (setq amx-backend 'ivy
        -show-key-bindings t)
  :bind
  ("M-x" . amx))

;;; switch buffer
(global-set-key (kbd "C-x b") 'ido-switch-buffer)
```

```
(provide 'search)
```

4.16 Read Aloud

```
(load-module 'read-aloud)
```

4.16.1 Requirements

4.16.2 Code

```
;;; read-aloud.el -*- lexical-binding: t; -*-

;;; read-aloud.el --- A simple interface to TTS engines  -*- lexical-binding:
↪  t; -*-

;; Author: Alexander Gromnitsky <alexander.gromnitsky@gmail.com>
;; Version: 0.0.2
;; Package-Requires: ((emacs "24.4"))
;; Keywords: multimedia
;; URL: https://github.com/gromnitsky/read-aloud.el

;; This file is not part of GNU Emacs.

;;; License:

;; MIT

;;; Commentary:

;; This package uses an external TTS engine (like flite) to pronounce
;; the word at or near point, the selected region or a whole buffer.

;;; Code:

(defvar read-aloud-engine "speech-dispatcher")
(setq read-aloud-engine "flite")
(defvar read-aloud-engines
  '("speech-dispatcher"                         ; Linux/FreeBSD only
    (cmd "spd-say" args ("-e" "-w") kill "spd-say -S")
    "flite"                                     ; Cygwin?
    (cmd "flite" args nil)
    "jampal"                                    ; Windows
    (cmd "cscript" args ("C:\\Program Files\\Jampal\\ptts.vbs" "-r" "5"))
    "say"                                       ; macOS
    (cmd "say" args nil)
    ))

(defvar read-aloud-max 160)                     ; chars
(defface read-aloud-text-face '((t :inverse-video t))
  "For highlighting the text that is being read")
```

```
(require 'cl-lib)
(require 'subr-x)

(defvar read-aloud-word-hist '())        ; (*-current-word) uses it
(defconst read-aloud--logbufname "*Read-Aloud Log*")

;; this should be in cl-defstruct
(defconst read-aloud--c-pr nil)
(defconst read-aloud--c-buf nil)
(defconst read-aloud--c-bufpos nil)
(defconst read-aloud--c-locked nil)
(defconst read-aloud--c-overlay nil)

(defun read-aloud--log(msg &optional args)
  (let ((buf (get-buffer-create read-aloud--logbufname)))
    (with-current-buffer buf
      (goto-char (point-max))
      (insert-before-markers (format (concat msg "\n") args))
      )))

(defun read-aloud-test ()
  "Open a new tmp buffer, insert a string, try to read it."
  (let ((buf (get-buffer-create "*Read-Aloud Test*")))

    (with-current-buffer buf
      (erase-buffer)
      (insert "Here lies the body of William Jay,
          Who died maintaining his right of way--
          He was right, dead right, as he speed along,
          But he's just as dead as if he were wrong."))

    ;; show our logs
    (switch-to-buffer read-aloud--logbufname)
    (goto-char (point-max))

    (read-aloud--u-switch-to-buffer buf)
    (goto-char (point-min))

    (setq read-aloud--c-buf buf)
    (setq read-aloud--c-bufpos 1)
    (read-aloud-buf)))

;;;###autoload
(defun read-aloud-change-engine()
  "Select another TTS engine."
  (interactive)
  (setq read-aloud-engine
        (ido-completing-read
         "read aloud with: "
         (cl-loop for (key _) on read-aloud-engines by 'cddr
                  collect key)
         nil nil nil nil read-aloud-engine
         )))

(defun read-aloud--cmd ()
  (or (plist-get (lax-plist-get read-aloud-engines read-aloud-engine) 'cmd)
      (user-error "Failed to get the default TTS engine")) )
```

```
(defun read-aloud--args ()
  (plist-get (lax-plist-get read-aloud-engines read-aloud-engine) 'args))

(defun read-aloud--valid-str-p (str)
  (and str (not (equal "" (string-trim str)))))

(defun read-aloud--overlay-rm()
  (when read-aloud--c-overlay
    (delete-overlay read-aloud--c-overlay)
    (setq read-aloud--c-overlay nil)))

(defun read-aloud--overlay-make(beg end)
  (when (and beg end)
    (setq read-aloud--c-overlay (make-overlay beg end))
    (overlay-put read-aloud--c-overlay 'face 'read-aloud-text-face) ))

(defun read-aloud--reset()
  "Reset internal state."
  (setq read-aloud--c-pr nil)
  (setq read-aloud--c-buf nil)
  (setq read-aloud--c-bufpos nil)
  (setq read-aloud--c-locked nil)

  (read-aloud--overlay-rm)
  (read-aloud--log "RESET"))

(cl-defun read-aloud--string(str source)
  "Open an async process, feed its stdin with STR. SOURCE is an
     arbitual string like 'buffer', 'word' or 'selection'."
  (unless (read-aloud--valid-str-p str) (cl-return-from read-aloud--string))

  (let ((process-connection-type nil)) ; (start-process) requires this

    (if read-aloud--c-locked (error "Read-aloud is LOCKED"))

    (setq read-aloud--c-locked source)
    (condition-case err
        (setq read-aloud--c-pr
              (apply 'start-process "read-aloud" nil
                     (read-aloud--cmd) (read-aloud--args)))
      (error
       (read-aloud--reset)
       (user-error "External TTS engine failed to start: %s"
                   (error-message-string err))) )

    (set-process-sentinel read-aloud--c-pr 'read-aloud--sentinel)
    (setq str (concat (string-trim str) "\n"))
    (read-aloud--log "Sending: `%s`" str)
    (process-send-string read-aloud--c-pr str)
    (process-send-eof read-aloud--c-pr)
    ))

(defun read-aloud--sentinel (process event)
  (let ((source read-aloud--c-locked))

    (setq event (string-trim event))
```

```
    (if (equal event "finished")
        (progn
          (read-aloud--overlay-rm)
          (setq read-aloud--c-locked nil)
          (cond
           ((equal source "buffer") (read-aloud-buf))
           ((equal source "word") t)           ; do nothing
           ((equal source "selection") t) ; do nothing
           (t (error "Unknown source: %s" source))) )

      ;; else
      (read-aloud--reset)
      (user-error "%s ended w/ the event: %s" process event)
      )))

;;;###autoload
(defun read-aloud-stop ()
  "Ask a TTS engine to stop."
  (interactive)
  (kill-process read-aloud--c-pr)

  ;; if a tts engine has a separate step to switch itself off, use it
  (let ((c (plist-get (lax-plist-get read-aloud-engines read-aloud-engine)
  ↪  'kill)))
    (when c
      (start-process-shell-command "read-aloud-kill" read-aloud--logbufname
      ↪  c)))

  (read-aloud--log "INTERRUPTED BY USER"))

(defun read-aloud-reset()
  (interactive)
    (setq read-aloud--c-buf (current-buffer)))

;;;###autoload
(cl-defun read-aloud-buf()
  "Read the current buffer, highlighting words along the
      read. Run it again to stop reading."
  (interactive)

  (when read-aloud--c-locked
    (read-aloud-stop)
    (cl-return-from read-aloud-buf))
  (unless read-aloud--c-buf (setq read-aloud--c-buf (current-buffer)))
  (unless read-aloud--c-bufpos (setq read-aloud--c-bufpos (point)))

  (let (tb)
    (with-current-buffer read-aloud--c-buf
      (when (eobp)
        (read-aloud--log "END OF BUFFER")
        (read-aloud--reset)
        (cl-return-from read-aloud-buf))

      (setq tb (read-aloud--grab-text read-aloud--c-buf read-aloud--c-bufpos))
      (unless tb
        (progn
          (read-aloud--log "SPACES AT THE END OF BUFFER")
```

```
          (read-aloud--reset)
          (cl-return-from read-aloud-buf)))

      ;; highlight text
      (read-aloud--overlay-make (plist-get tb 'beg) (plist-get tb 'end))

      (goto-char (plist-get tb 'end))
      (read-aloud--string (plist-get tb 'text) "buffer")

      (setq read-aloud--c-bufpos (plist-get tb 'end))
      )))

(cl-defun read-aloud--grab-text(buf point)
  "Return (text \"omglol\" beg 10 end 20) plist or nil on
      eof. BUF & POINT are the starting location for the job."
  (let (max t2 p pstart chunks pchunk)

    (with-current-buffer buf
      (save-excursion
        (goto-char point)
        (skip-chars-forward "[\\-,.:!;[:space:]\r\n]")

        (setq max (+ (point) read-aloud-max))
        (if (> max (point-max)) (setq max (point-max)))
        (setq t2 (buffer-substring-no-properties (point) max))

        (if (string-empty-p (string-trim-right t2))
            ;; we have spaces at the end of buffer, there is nothing to grab
            (cl-return-from read-aloud--grab-text nil))

        (setq pstart (point))

        (unless (= max (point-max))
          (progn
            ;; look for the 1st non-space in `t` from the end & cut
            ;; off that part
            (setq p (string-match "[[:space:]\r\n]"
                                  (read-aloud--u-str-reverse t2)) )
            (if p (setq t2 (substring t2 0 (- (length t2) p 1))) )))

        (setq chunks
              (split-string t2 "[,.:!;]\\|\\(-\\|\n\\|\r\n\\)\\{2,\\}" t))
        (if chunks
            (progn
              (search-forward (car chunks))
              (setq pchunk (point))
              (search-backward (car chunks))
              (setq pstart (point))
              (setq t2 (buffer-substring-no-properties pstart pchunk)) ))

        (read-aloud--log "text grab: `%s`" t2)
        `(text ,t2
               beg ,pstart
               end ,(+ pstart (length t2)))
        ))))

(cl-defun read-aloud--current-word()
```

```
  "Pronounce a word under the pointer. If under there is rubbish,
      ask user for an additional input."
  (let* ((cw (read-aloud--u-current-word))
         (word (nth 2 cw)))

    (unless (and word (string-match "[[:alnum:]]" word))
      ;; maybe we should share the hist list w/ `wordnut-completion-hist`?
      (setq word (read-string "read aloud: " word 'read-aloud-word-hist)) )

    (read-aloud--overlay-make (nth 0 cw) (nth 1 cw))
    (read-aloud--string word "word")
    ))

;;;###autoload
(cl-defun read-aloud-this()
  "Pronounce either the selection or a word under the pointer."
  (interactive)

  (when read-aloud--c-locked
    (read-aloud-stop)
    (cl-return-from read-aloud-selection))

  (if (use-region-p)
      (read-aloud--string
       (buffer-substring-no-properties (region-beginning) (region-end))
       "selection")
    (read-aloud--current-word)) )




(defun read-aloud--u-switch-to-buffer(buf)
  (unless (eq (current-buffer) buf)
    (unless (cdr (window-list))
      (split-window-vertically))
    (other-window 1)
    (switch-to-buffer buf)))

;; for emacs < 25
(defun read-aloud--u-str-reverse (str)
  "Reverse the STR."
  (apply #'string (reverse (string-to-list str))))

(defun read-aloud--u-current-word()
  "This is a modified (current-word) that doesn't take any args &
      return (beg end word) or nil."
  (save-excursion
    (let* ((oldpoint (point)) (start (point)) (end (point))
           (syntaxes "w_")
           (not-syntaxes (concat "^" syntaxes)))
      (skip-syntax-backward syntaxes) (setq start (point))
      (goto-char oldpoint)
      (skip-syntax-forward syntaxes) (setq end (point))
      (when (and (eq start oldpoint) (eq end oldpoint))
        ;; Look for preceding word in same line.
        (skip-syntax-backward not-syntaxes (line-beginning-position))
        (if (bolp)
```

```
            ;; No preceding word in same line.
            ;; Look for following word in same line.
            (progn
              (skip-syntax-forward not-syntaxes (line-end-position))
              (setq start (point))
              (skip-syntax-forward syntaxes)
              (setq end (point)))
          (setq end (point))
          (skip-syntax-backward syntaxes)
          (setq start (point))))
      ;; If we found something nonempty, return it as a list.
      (unless (= start end)
        (list start end (buffer-substring-no-properties start end)))
      )))
(provide 'read-aloud)
```

4.17 Speed Read

```
(load-module 'speed-read)
```

4.17.1 Requirements

4.17.2 Code

```
;;; read-single.el -*- lexical-binding: t; -*-

(use-package! spray
  :commands spray-mode
  :config
  (setq spray-wpm 400
        spray-height 300)
  (defun spray-mode-hide-cursor ()
    "Hide or unhide the cursor as is appropriate."
    (if spray-mode
        (setq-local spray--last-evil-cursor-state evil-normal-state-cursor
                    evil-normal-state-cursor '(nil))
      (setq-local evil-normal-state-cursor spray--last-evil-cursor-state)))
  (add-hook 'spray-mode-hook #'spray-mode-hide-cursor)
  (map! :map spray-mode-map
        :n "<return>" #'spray-start/stop
        :n "SPC" #'spray-start/stop
        :n "f" #'spray-faster
        :n "s" #'spray-slower
        :n "t" #'spray-time
        :n "<right>" #'spray-forward-word
        :n "h" #'spray-forward-word
        :n "<left>" #'spray-backward-word
        :n "l" #'spray-backward-word
        :n "q" #'spray-quit))

(provide 'speed-read)
```

4.18 Spaced Repetition

```
(load-module 'spaced-repetition)
```

4.18.1 Requirements

4.18.2 Code

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

;;; pamparam.el --- Simple and fast flashcards. -*- lexical-binding: t -*-

;; Copyright (C) 2016-2020 Oleh Krehel

;; Author: Oleh Krehel <ohwoeowho@gmail.com>
;; URL: https://github.com/abo-abo/pamparam
;; Version: 0.1.0
;; Package-Requires: ((emacs "26.1") (lispy "0.27.0") (worf "0.1.0")
↪  (ivy-posframe "0.5.5"))
;; Keywords: outlines, hypermedia, flashcards, memory

;; This file is not part of GNU Emacs

;; This file is free software; you can redistribute it and/or modify
;; it under the terms of the GNU General Public License as published by
;; the Free Software Foundation; either version 3, or (at your option)
;; any later version.

;; This program is distributed in the hope that it will be useful,
;; but WITHOUT ANY WARRANTY; without even the implied warranty of
;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;; GNU General Public License for more details.

;; For a full copy of the GNU General Public License
;; see <http://www.gnu.org/licenses/>.

;;; Commentary:
;;
;; An example master file is given in doc/sets/capitals/capitals.org.
;; Use `hydra-pamparam/body' as the entry point.
;; See README.org for more info.

;;; Code:

;;* Requires
(require 'worf)
(require 'ivy)

(defgroup pamparam nil
  "Simple and fast flashcards."
  :group 'flashcards)

;;* Pure
(defun pamparam-sm2 (card-stats q)
```

```
  "Determine the next iteration of CARD-STATS based on Q.
      CARD-STATS is (EASE-FACTOR . INTERVALS), the result has the
      same shape, with updated values.
      EASE-FACTOR - the previous ease factor of the card.  All cards are
      initialized with EASE-FACTOR of 2.5.  It will decrease for
      difficult cards, but not below 1.3.
      INTERVALS - list of integer day intervals between repetitions.
      Q - the quality of the answer:
        5 - perfect response
        4 - correct response after a hesitation
        3 - correct response recalled with serious difficulty
        2 - incorrect response; where the correct one seemed easy to recall
        1 - incorrect response; the correct one remembered
        0 - complete blackout"
  (let ((EF (car card-stats))
        (intervals (cdr card-stats)))
    (setq EF (max 1.3 (+ EF 0.1 (* (- q 5) (+ 0.08 (* (- 5 q) 0.02))))))
    (if (< q 3)
        (cons EF (cons 1 intervals))
      (cons EF
            (cons
             (cond ((null intervals)
                    1)
                   ((= (car intervals) 1)
                    6)
                   (t
                    (round (* EF (car intervals)))))
             intervals)))))

;;* Card files
(defun pamparam-card-insert-score (score actual-answer)
  "Insert SCORE into the current card file."
  (goto-char (point-min))
  (outline-show-all)
  (if (re-search-forward "^\\*\\* scores" nil t)
      (outline-end-of-subtree)
    (forward-line 2)
    (insert "** scores\n")
    (backward-char))
  (when actual-answer
    (kill-new actual-answer))
  (insert (format-time-string "\n| <%Y-%m-%d> ")
          (format "| %d |" score)
          (format " %s |"
                  (or actual-answer "")))
  (org-table-align))

(defun pamparam-wdiff (actual-answer)
  (let ((expected-answer
         (save-excursion
           (goto-char (point-max))
           (skip-chars-backward "\n")
           (buffer-substring-no-properties
            (line-beginning-position)
            (line-end-position)))))
    (when (and actual-answer
               (not (pamparam-equal actual-answer expected-answer))
```

```
              (executable-find "wdiff"))
      (message
       (string-trim
        (shell-command-to-string
         (format
          "wdiff -i <(echo \"%s\") <(echo \"%s\")"
          actual-answer
          (string-trim-right expected-answer "[.?!]")))))))))

(defun pamparam-card-read-stats ()
  (goto-char (point-min))
  (if (re-search-forward "^\\*\\* stats\n" nil t)
      (let ((beg (point))
            (exp1 (read (current-buffer)))
            (exp2 (read (current-buffer)))
            ease-factor intervals)
        (if (and (eq (nth 0 exp1) 'setq)
                 (eq (nth 1 exp1) 'ease-factor)
                 (numberp (nth 2 exp1)))
            (setq ease-factor (nth 2 exp1))
          (error "Bad sexp %S" exp1))
        (if (and (eq (nth 0 exp2) 'setq)
                 (eq (nth 1 exp2) 'intervals))
            (setq intervals (cadr (nth 2 exp2)))
          (error "Bad sexp %S" exp2))
        (delete-region beg (point))
        (cons ease-factor intervals))
    (if (re-search-forward "^\\*\\* scores\n" nil t)
        (progn
          (outline-end-of-subtree)
          (insert "\n** stats\n")
          (list 2.5))
      (error "** scores not found"))))

(defun pamparam-card-insert-stats (stats)
  (insert (format "(setq ease-factor %f)\n" (car stats)))
  (insert (format "(setq intervals '%S)" (cdr stats))))

(defun pamparam-delete-region (beg end)
  (let ((str (buffer-substring-no-properties beg end)))
    (delete-region beg end)
    str))

(defun pamparam-save-buffer ()
  (let ((inhibit-message t))
    (write-file (buffer-file-name)))
  (pamparam-card-abbreviate))

(defun pamparam-card-abbreviate ()
  (let ((fname (file-name-nondirectory (buffer-file-name))))
    (when (> (length fname) 60)
      (rename-buffer
       (concat "card-" (substring fname 0 6) ".org")))))

(defun pamparam-card-score (score &optional actual-answer)
  (let* ((card-file (file-name-nondirectory (buffer-file-name)))
         (todo-file (pamparam-todo-file))
```

```
        (state (with-current-buffer todo-file
                 (goto-char (point-min))
                 (search-forward card-file)
                 (goto-char (+ 2 (line-beginning-position)))
                 (buffer-substring-no-properties
                  (point)
                  (progn
                    (forward-word)
                    (point)))))
        (save-silently t)
        (inhibit-read-only t))
    (cond ((string= state "REVIEW")
           (with-current-buffer todo-file
             (goto-char (point-min))
             (search-forward card-file)
             (if (or (= score 5)
                     (= score 4)
                     (= score 3))
                 (let ((org-log-done nil)
                       (inhibit-message t))
                   (org-todo 'done))
               (let ((item (pamparam-delete-region
                            (line-beginning-position)
                            (1+ (line-end-position)))))
                 (goto-char (point-max))
                 (insert item)))
             (pamparam-save-buffer))
           (pamparam-save-buffer))
          ((string= state "DONE")
           (if (y-or-n-p "Card already done today.  Re-rate? ")
               (pamparam--card-score score t actual-answer)
             (user-error "This card is already done today")))
          ((string= state "TODO")
           (pamparam--card-score score nil actual-answer))
          (t
           (user-error "Unexpected state: %s" state)))
    (with-current-buffer todo-file
      (pamparam--recalculate-progress))
    (outline-show-all)))

(defun pamparam-card-manual-score ()
  "Score the card 0-5 manually."
  (interactive)
  (undo)
  (let ((score (completing-read "score: " '("0" "1" "2" "3" "4" "5") nil t)))
    (pamparam-card-score (string-to-number score))))

(defun pamparam--todo-from-file (card-file)
  (if (string-match "\\`\\([^-]+\\)-" card-file)
      (format
       "* TODO [[file:cards/%s/%s][%s]]\n"
       (substring card-file 0 2)
       card-file
       (match-string 1 card-file))
    (error "Unexpected file name")))

(defun pamparam--card-score (score &optional already-done actual-answer)
```

```
  (let ((card-file (file-name-nondirectory (buffer-file-name)))
        stats
        new-interval)
    (save-excursion
      (pamparam-card-insert-score score actual-answer)
      (setq stats (pamparam-card-read-stats))
      (setq stats (pamparam-sm2 stats score))
      (pamparam-card-insert-stats stats)
      (setq new-interval (nth 1 stats))
      (unless already-done
        (let* ((todo-entry (pamparam--todo-from-file card-file))
               str)
          (with-current-buffer (pamparam-todo-file)
            (goto-char (point-min))
            (when (search-forward card-file)
              (if (memq score '(4 5))
                  (progn
                    (beginning-of-line)
                    (if (looking-at "\\* \\(TODO\\|REVIEW\\)")
                        (replace-match "DONE" nil nil nil 1)
                      (error "Unexpected")))
                (setq str (buffer-substring-no-properties
                           (+ 7 (line-beginning-position))
                           (1+ (line-end-position))))
                (delete-region
                 (line-beginning-position)
                 (1+ (line-end-position)))
                (goto-char (point-max))
                (insert "* REVIEW " str))
              (pamparam-save-buffer)))
          (with-current-buffer (pamparam-todo-file new-interval)
            (goto-char (point-min))
            (unless (search-forward todo-entry nil t)
              (goto-char (point-max))
              (insert todo-entry)
              (pamparam-save-buffer))
            (kill-buffer))))
      (pamparam-save-buffer)
      (pamparam-wdiff actual-answer))))

(defvar-local pamparam-card-answer-validate-p nil)

(defcustom pamparam-card-answer-function #'pamparam-card-answer-at-point
  "Select how to answer the card."
  :type '(choice
          (const :tag "Answer at point" pamparam-card-answer-at-point)
          (const :tag "Answer in a child frame"
          ↪  pamparam-card-answer-posframe)))

(defun pamparam-card-answer-at-point ()
  "Answer the current card.
     Enter the answer at point, then press \".\" to validate."
  (goto-char (point-min))
  (when (re-search-forward "^\\* m$" nil t)
    (delete-region (point-min) (match-beginning 0)))
  (goto-char (point-min))
  (insert "* \n")
```

```
  (goto-char 3)
  (setq pamparam-card-answer-validate-p t)
  (outline-hide-body))

(defvar pamparam-posframe-keymap
  (let ((map (make-sparse-keymap)))
    (define-key map (kbd "C-v") #'pamparam-card-reveal)
    (define-key map (kbd ".") #'ivy-done)
    map)
  "The keymap for `pamparam-card-answer-posframe'")

(defun pamparam-card-reveal ()
  (interactive)
  (with-current-buffer (ivy-state-buffer ivy-last)
    (pamparam-shifttab)))

(defun pamparam--ivy-read-posframe (prompt)
  (let ((ivy-posframe-state (bound-and-true-p ivy-posframe-mode)))
    (unless ivy-posframe-state
      (ivy-posframe-mode 1))
    (unwind-protect
         (let ((ivy-add-newline-after-prompt t))
           (ivy-read prompt nil
                     :keymap pamparam-posframe-keymap))
      (unless ivy-posframe-state
        (ivy-posframe-mode -1)))))

(defun pamparam-card-answer-posframe ()
  (outline-hide-body)
  (read-only-mode 1)
  (let* ((card-front
          (save-excursion
            (goto-char (point-min))
            (zo-down 1)
            (substring-no-properties (org-get-heading))))
         (answer (pamparam--ivy-read-posframe
                  (concat card-front ": "))))
    (unless (string= answer "")
      (pamparam-card-validate answer (pamparam--card-true-answer)))
    (remove-overlays (point-min) (point-max) 'invisible 'outline)
    (read-only-mode 1)))

(defun pamparam-card-answer ()
  "Answer the current card."
  (funcall pamparam-card-answer-function))

(defvar pamparam-is-redo nil)

(defun pamparam--card-true-answer ()
  (save-excursion
    (goto-char (point-max))
    (re-search-backward "^\\*")
    (beginning-of-line 2)
    (buffer-substring-no-properties
     (point)
     (1- (point-max)))))
```

```
(defun pamparam-card-validate-maybe (&optional arg)
  "Validate the given answer and score the current card.
      The given answer is the text between the card's first heading and
      point."
  (interactive "p")
  (if pamparam-card-answer-validate-p
      (let ((tans (pamparam--card-true-answer))
            (actual-answer (buffer-substring-no-properties
                            (+ (line-beginning-position) 2)
                            (line-end-position))))
        (delete-region (point-min)
                       (1+ (line-end-position)))
        (setq pamparam-card-answer-validate-p nil)
        (pamparam-card-validate actual-answer tans))
    (self-insert-command arg)))

(defun pamparam-card-validate (actual-answer correct-answer)
  "Give a card score, comparing ACTUAL-ANSWER to CORRECT-ANSWER."
  (if (pamparam-equal actual-answer correct-answer)
      (if (save-excursion
            (goto-char (point-max))
            (re-search-backward "^\\* ")
            (overlays-in (point) (point-max)))
          (if pamparam-is-redo
              (pamparam-card-score 4)
            (pamparam-card-score 5))
        (pamparam-card-score 3))
    (pamparam-card-score 0 actual-answer)))

;;* Equivalence testing
(defvar pamparam-equiv-hash (make-hash-table :test 'equal))

(defvar pamparam-equiv-classes '(("we" "wij")
                                 ("je" "jij")
                                 ("ze" "zij")
                                 ("u" "jij")
                                 ("dichtbij" "vlakbij")
                                 ("test" "toets")))

(defun pamparam-make-equivalent (a b)
  (puthash a b pamparam-equiv-hash)
  (puthash b b pamparam-equiv-hash))

(dolist (c pamparam-equiv-classes)
  (pamparam-make-equivalent (car c) (cadr c)))

(defun pamparam-equal (sa sb)
  "Check if the answer SA matches the question SB.
      When SB has multiple lines, SA may match one of them."
  (if (string-match-p "\n" sb)
      (let ((sbl (split-string sb "\n" t))
            res)
        (while (and (null res) (setq sb (pop sbl)))
          (setq res (pamparam-equal-single sa sb)))
        res)
    (pamparam-equal-single sa sb)))
```

```
(defun pamparam-equal-single (sa sb)
  "Check if SA matches SB."
  (let ((lista (pamparam-sloppy sa))
        (listb (pamparam-sloppy sb))
        (res t)
        a b
        ah)
    (while (and res lista)
      (setq a (pop lista))
      (setq b (pop listb))
      (unless (or (string= a b)
                  (and (setq ah (gethash a pamparam-equiv-hash))
                       (equal ah
                              (gethash b pamparam-equiv-hash))))
        (setq res nil)))
    (and res (null listb))))

(defun pamparam-sloppy (str)
  (mapcar #'downcase
          (split-string str "[.,?!: ]" t)))

(defvar pamparam-load-file-name (or load-file-name
                                    (buffer-file-name)))

(defvar pamparam-path (expand-file-name
                       "doc/sets/capitals/capitals.pam"
                       (file-name-directory pamparam-load-file-name))
  "Point to a default repository. In case you call `pamparam-drill'
      while not in any repo, this repo will be selected.")

(defvar pamparam-alist
  (list (cons (expand-file-name "capitals.org"
                                (file-name-directory pamparam-path))
              pamparam-path))
  "Map a master file to the corresponding repository.
      Otherwise, the repository will be in the same directory as the master
  ↪  file.")

;;* Schedule files
(defun pamparam-repo-directory (file)
  "Return the Git repository that corresponds to FILE."
  (or (cdr (assoc file pamparam-alist))
      (if file
          (expand-file-name
           (concat
            (file-name-sans-extension
             (file-name-nondirectory
              file))
            ".pam/"))
        (locate-dominating-file default-directory ".git"))))

(defun pamparam-repo-init (repo-dir)
  "Initialize REPO-DIR Git repository."
  (if (file-exists-p repo-dir)
      (unless (file-directory-p repo-dir)
        (error "%s must be a directory" repo-dir))
    (make-directory repo-dir)
```

```
    (let ((default-directory repo-dir))
      (shell-command "git init")
      (make-directory "cards/"))))

(defvar pamparam-new-cards-per-day 75)

(defun pamparam-card-delete (file)
  "Delete the card in FILE.
      When called interactively, delete the card in the current buffer."
  (interactive (list (buffer-file-name)))
  (when (and (file-exists-p file)
             (y-or-n-p
              (format "Really delete %s? "
                      (file-name-nondirectory file))))
    (delete-file file)
    (when (string= (buffer-file-name) file)
      (kill-buffer))
    (pamparam--update-card
     (file-name-nondirectory file)
     nil)))

(defun pamparam--update-card (prev-file new-entry)
  (let ((prev-scheduled (pamparam-cmd-to-list (format "git grep %s"
  ↪  (shell-quote-argument prev-file))))
        (save-silently t))
    (dolist (prev prev-scheduled)
      (unless (string-match
      ↪  "\\`\\([^:]+\\):.*\\[\\[file:cards/\\(.*\\)\\]\\[.*\\]\\]'" prev)
        (user-error "Bad scheduled item: %s" prev))
      (let ((schedule-file
             (expand-file-name
              (match-string 1 prev)))
            (entry (match-string 2 prev)))
        (with-temp-buffer
          (insert-file-contents schedule-file)
          (when (re-search-forward entry nil t)
            (if new-entry
                (replace-match new-entry)
              (delete-region
               (line-beginning-position)
               (1+ (line-end-position)))))
          (write-file schedule-file))))))

(defvar pamparam-hash-card-name->file nil)
(defvar pamparam-hash-card-body->file nil)

(defun pamparam-cmd-to-list (cmd &optional directory)
  (let ((default-directory (or directory default-directory)))
    (split-string
     (shell-command-to-string cmd)
     "\n" t)))

(defun pamparam-cards (repo-dir)
  (pamparam-cmd-to-list
   "git ls-files cards/"
   repo-dir))
```

```
(defun pamparam-visited-cards (repo-dir)
  (pamparam-cmd-to-list
   "git grep --files-with-matches '^\\*\\* scores'"
   repo-dir))

(defun pamparam-unvisited-cards (repo-dir)
  (pamparam-cmd-to-list
   "git grep --files-without-match '^\\*\\* scores' | grep cards/"
   repo-dir))

(defun pamparam-pile (repo-dir)
  "Pile up all unvisited cards into a single file."
  (let ((unvisited-cards (pamparam-unvisited-cards repo-dir))
        (schedule-files (pamparam-cmd-to-list "git ls-files --full-name
        ↪  pamparam-*-[0-9][0-9].org"))
        (save-silently t))
    (dolist (sf schedule-files)
      (with-current-buffer (find-file (expand-file-name sf repo-dir))
        (dolist (card unvisited-cards)
          (goto-char (point-min))
          (while (search-forward card nil t)
            (delete-region (line-beginning-position) (1+
            ↪  (line-end-position)))))
        (pamparam-save-buffer)
        (kill-buffer)))
    (with-current-buffer (find-file (expand-file-name "pampile.org" repo-dir))
      (delete-region (point-min) (point-max))
      (dolist (card unvisited-cards)
        (insert (pamparam--todo-from-file (file-name-nondirectory card))))
      (pamparam-save-buffer)
      (kill-buffer))))

(defun pamparam-pull (arg &optional buffer)
  "Pull ARG cards into BUFFER.
     When called interactively, use today's schedule file."
  (interactive
   (list (read-number "how many cards: ")
         (pamparam-todo-file)))
  (let ((save-silently t)
        cards)
    (setq arg (min 100 arg))
    (switch-to-buffer buffer)
    (with-current-buffer (find-file-noselect
                          (expand-file-name "pampile.org"))
      (goto-char (point-min))
      (end-of-line arg)
      (setq cards (pamparam-delete-region (point-min)
                                          (min (1+ (point))
                                               (point-max))))
      (pamparam-save-buffer)
      (kill-buffer))
    (pamparam-goto-schedule-part)
    (insert cards)
    (pamparam-save-buffer)))

(defun pamparam-goto-schedule-part ()
  (goto-char (point-min))
```

```
  (if (re-search-forward "^\\*" nil t)
      (goto-char (match-beginning 0))
    (goto-char (point-max))))

(defun pamparam--recompute-git-cards (repo-dir)
  (setq pamparam-hash-card-name->file (make-hash-table :test 'equal))
  (setq pamparam-hash-card-body->file (make-hash-table :test 'equal))
  (let ((git-files (pamparam-cards repo-dir)))
    (dolist (gf git-files)
      (if (string-match
      ↪  "\\`cards/[0-9a-f]\\{2\\}/\\([^-]+\\)-\\([^.]+\\)\\.org\\'" gf)
          (progn
            (puthash (match-string 1 gf) gf pamparam-hash-card-name->file)
            (puthash (match-string 2 gf) gf pamparam-hash-card-body->file))
        (error "Unexpected file name %s" gf)))))

(defun pamparam--replace-card (_card-front _card-body repo-dir card-file
↪  prev-file)
  (let* ((full-name (expand-file-name prev-file repo-dir))
         (old-metadata
          (with-temp-buffer
            (insert-file-contents full-name)
            (goto-char (point-min))
            (when (looking-at "\\* m$")
              (outline-end-of-subtree)
              (buffer-substring-no-properties
               (point-min)
               (1+ (point)))))))
    (pamparam-kill-buffer-of-file full-name)
    (delete-file full-name)
    (let ((default-directory repo-dir)
          (fnn (file-name-nondirectory card-file)))
      (pamparam--update-card prev-file (concat (substring fnn 0 2) "/" fnn)))
    old-metadata))

(eval-and-compile
  (if (eq system-type 'windows-nt)
      (defun pamparam-spit (str file)
        (with-current-buffer (find-file-noselect file)
          (erase-buffer)
          (insert str)
          (save-buffer)
          (kill-buffer (current-buffer))))
    (defun pamparam-spit (str file)
      (let ((cmd (format "echo '%s' > %s"
                         (replace-regexp-in-string "'" "'\\''" str t t)
                         (shell-quote-argument file))))
        (unless (= 0 (call-process-shell-command cmd))
          (error "Command failed: %s" cmd))))))

(defun pamparam-slurp (f)
  (with-temp-buffer
    (insert-file-contents f)
    (buffer-string)))

(defun pamparam-update-card (card-front card-body repo-dir)
  (let* ((card-front-id (md5 card-front))
```

```
         (card-body-id (md5 card-body))
         (prev-file
          (or
           (gethash card-front-id pamparam-hash-card-name->file)
           (gethash card-body-id pamparam-hash-card-body->file)))
         (subdir (substring card-front-id 0 2))
         (card-file
          (concat
           "cards/" subdir "/" card-front-id "-" card-body-id ".org"))
         (full-card-file (expand-file-name card-file repo-dir))
         (metadata nil))
    (cond ((null prev-file))
          ((string= card-file prev-file))
          (t
           (when (file-exists-p (expand-file-name prev-file repo-dir))
             (setq metadata (pamparam--replace-card
                             card-front card-body repo-dir card-file
                             ↪  prev-file)))))
    (unless (file-exists-p (expand-file-name card-file repo-dir))
      (let* ((txt
              (concat
               (or metadata "* m\n#+STARTUP: content\n")
               (format "* %s\n%s" card-front card-body))))
        (make-directory (file-name-directory full-card-file) t)
        (pamparam-spit txt full-card-file)
        (cons (if metadata
                  'update
                'new)
              card-file)))))

(defconst pamparam-card-source-regexp "^\\*+ .*:cards:")

(defun pamparam-sync ()
  "Synchronize the current `org-mode' master file to the cards repository.
     Create the cards repository if it doesn't exist.
     Each card is uniquely identifiable by either its front or its
     back.  So if you want to modify both the front and the back, first
     modify the front, call `pamparam-sync', then modify the back and call
     `pamparam-sync' again.  Otherwise, there's no way to \"connect\" the
     new card to the old one, and the old card will remain in the
     repository, while the new card will start with empty metadata."
  (interactive)
  (unless (eq major-mode 'org-mode)
    (error "Must be in `org-mode' file"))
  (when (pamparam--cards-available-p)
    (let ((repo-dir (pamparam-repo-directory (buffer-file-name)))
          (make-backup-files nil))
      (pamparam-repo-init repo-dir)
      (pamparam--recompute-git-cards repo-dir)
      (pamparam--sync repo-dir))))

(defun pamparam-kill-buffer-of-file (fname)
  (dolist (buf (buffer-list))
    (when (equal fname (buffer-file-name buf))
      (kill-buffer buf))))

(defvar org-keyword-properties)
```

```
(defun pamapram--cards-at-level-one-p ()
  (let ((alist (if (boundp 'org-file-properties)
                   org-file-properties
                 org-keyword-properties)))
    (assoc-string "pamparam" alist t)))

(defun pamparam--cards-available-p ()
  (or (pamapram--cards-at-level-one-p)
      (save-excursion
        (goto-char (point-min))
        (if (re-search-forward pamparam-card-source-regexp nil t)
            t
          (error "No outlines with the :cards: tag found")))))

(defun pamparam--sync (repo-dir)
  (let ((old-point (point))
        (processed-headings nil)
        (new-cards nil)
        (updated-cards nil))
    (goto-char (point-min))
    (let* ((cards-at-level-one-p (pamapram--cards-at-level-one-p))
           (regex (if cards-at-level-one-p
                      "\\*+ .*$"
                    pamparam-card-source-regexp)))
      (while (re-search-forward regex nil t)
        (when cards-at-level-one-p
          (beginning-of-line))
        (lispy-destructuring-setq (processed-headings new-cards updated-cards)
            (pamparam-sync-current-outline
             processed-headings new-cards updated-cards repo-dir))))
    (goto-char old-point)
    (when (or new-cards updated-cards)
      (let ((pile-fname (expand-file-name "pampile.org" repo-dir)))
        (pamparam-kill-buffer-of-file pile-fname)
        (pamparam-schedule-today
         (mapcar #'pamparam--todo-from-file new-cards)
         (find-file-noselect pile-fname)))
      (shell-command-to-string
       (format
        "cd %s && git add . && git commit -m %s"
        (shell-quote-argument repo-dir)
        (shell-quote-argument
         (cond ((null updated-cards)
                (format "Add %d new card(s)" (length new-cards)))
               ((null new-cards)
                (format "Update %d card(s)" (length updated-cards)))
               (t
                (format "Add %d new card(s), update %d cards"
                        (length new-cards)
                        (length updated-cards)))))))))
    (message "%d new cards, %d updated, %d total"
             (length new-cards)
             (length updated-cards)
             (length processed-headings))))
```

```
(defun pamparam--card-info ()
  (let* ((bnd (worf--bounds-subtree))
         (str (lispy--string-dwim bnd))
         front back)
    (cond ((string-match "^\\*+ a\n\\(.*\\)" str)
           (setq front (substring str 0 (match-beginning 0)))
           (setq back (concat "* a\n" (match-string 1 str)))
           (setq front (string-trim-left front))
           (goto-char (cdr bnd)))
          ((string-match "\\`\\*+ \\(.*\\)\n\\([^*]+\\)\\(?:\n\\*\\)?" str)
           (setq front (match-string 1 str))
           (setq back (match-string 2 str))
           (goto-char (+ (car bnd) (match-end 2)))
           (setq back (string-trim-right back)))
          ((string-match "\\`\\(\\*+ \\).*{\\([^}]+\\)}.*\\'" str)
           (setq front
                 (concat (substring str (match-end 1) (1- (match-beginning 2)))
                         "[...]"
                         (substring str (1+ (match-end 2)))))
           (setq back (match-string 2 str)))
          (t
           (error "unexpected")))
    (cons front back)))

(defun pamparam-sync-current-outline (processed-headings new-cards
↪  updated-cards repo-dir)
  (let ((end (save-excursion
               (outline-end-of-subtree)
               (point))))
    (while (re-search-forward "^\\*+ \\(.*\\)$" end t)
      (let* ((card-info (pamparam--card-info))
             (card-front (car card-info))
             (card-body (cdr card-info))
             card-file)
        (if (member card-front processed-headings)
            (error "Duplicate heading encountered: %s" card-front)
          (push card-front processed-headings))
        (when (setq card-info (pamparam-update-card card-front card-body
        ↪  repo-dir))
          (setq card-file (file-name-nondirectory (cdr card-info)))
          (cond ((eq (car card-info) 'new)
                 (push card-file new-cards))
                ((eq (car card-info) 'update)
                 (push card-file updated-cards))))))
    (list processed-headings new-cards updated-cards)))

(defun pamparam-default-directory ()
  (if (string-match "^\\(.*\\.pam/\\)" default-directory)
      (expand-file-name (match-string 1 default-directory))
    pamparam-path))

(defun pamparam-kill-buffers ()
  (let* ((pdir (pamparam-default-directory))
         (cards-dir (expand-file-name "cards/" pdir)))
    (dolist (b (buffer-list))
      (when (buffer-file-name b)
        (let ((dir (file-name-directory (buffer-file-name b))))
```

```
          (when (or (equal dir cards-dir)
                    (and (equal dir pdir)
                         (not (equal (file-name-nondirectory
                                      (buffer-file-name b))
                                     (pamparam-schedule-file
                                     ↪  (current-time))))))
            (kill-buffer b)))))))

(defun pamparam-schedule-file (time)
  (let ((year (format-time-string "%Y" time))
        (current-year (format-time-string "%Y" (current-time)))
        (base (format-time-string "pam-%Y-%m-%d.org" time)))
    (if (string= year current-year)
        base
      (let ((dir (expand-file-name
                  year (expand-file-name "years"
                  ↪  (pamparam-default-directory)))))
        (unless (file-exists-p dir)
          (make-directory dir t))
        (expand-file-name base dir)))))

(defun pamparam-todo-file (&optional offset)
  (setq offset (or offset 0))
  (let* ((default-directory (pamparam-default-directory))
         (todo-file (expand-file-name
                     (pamparam-schedule-file
                      (time-add
                       (current-time)
                       (days-to-time offset)))))
         (save-silently t))
    (unless (file-exists-p todo-file)
      (save-current-buffer
        (find-file todo-file)
        (insert "#+SEQ_TODO: TODO REVIEW | DONE\n")
        (when (eq offset 0)
          (pamparam-pull 10 (current-buffer))
          (message "Schedule was empty, used `pamparam-pull' for 10 cards"))
        (pamparam-save-buffer)))
    (find-file-noselect todo-file)))

(defvar pamparam-last-reschedule nil)

(defun pamparam-schedule-today (cards &optional buffer)
  (with-current-buffer (or buffer (pamparam-todo-file))
    (pamparam-goto-schedule-part)
    (dolist (card cards)
      (insert card))
    (let ((save-silently t))
      (pamparam-save-buffer))))

(defvar-local pamparam--progress nil
  "Cache the current progress.")

(defun pamparam-current-progress ()
  (with-current-buffer (pamparam-todo-file)
    (or pamparam--progress
        (pamparam--recalculate-progress))))
```

```
(defun pamparam--recalculate-progress ()
  (setq pamparam--progress
        (let ((n-done 0)
              (n-todo 0)
              (n-review 0))
          (save-excursion
            (goto-char (point-min))
            (while (re-search-forward "^\\* \\(TODO\\|DONE\\|REVIEW\\)" nil t)
              (let ((ms (match-string 1)))
                (cond ((string= ms "TODO")
                       (cl-incf n-todo))
                      ((string= ms "DONE")
                       (cl-incf n-done))
                      ((string= ms "REVIEW")
                       (cl-incf n-review)))))
            (list n-done n-todo n-review)))))

(defun pamparam-mode-line ()
  (cl-destructuring-bind (n-done n-todo n-review)
      (pamparam-current-progress)
    (format "(pam: %d/%d+%d)" n-done n-todo n-review)))

(defvar pamparam-day-limit 50
  "Limit for today's repetitions.
     All cards above this number that would be scheduled for today
     will instead be moved to tomorrow.")

(defun pamparam-merge-schedules (from to)
  "Copy items FROM -> TO. Delete FROM."
  (let ((from-lines
         (cl-remove-if-not
          (lambda (s) (string-match-p "^\\*" s))
          (split-string (pamparam-slurp from) "\n" t)))
        (to-lines (split-string (pamparam-slurp to) "\n" t)))
    (pamparam-spit
     (mapconcat #'identity
                (append to-lines from-lines)
                "\n")
     to)
    (delete-file from)))

(defun pamparam-carryover-year-maybe ()
  "Move e.g. years/2018/*.org to . if the current year is 2018."
  (let* ((today (calendar-current-date))
         (year (nth 2 today))
         (default-directory (pamparam-default-directory))
         (year-directory (format "years/%d" year)))
    (when (file-exists-p year-directory)
      (let ((year-files (directory-files year-directory nil "org$")))
        (dolist (file year-files)
          (let ((file-from (expand-file-name file year-directory))
                (file-to (expand-file-name file)))
            (if (file-exists-p file-to)
                (pamparam-merge-schedules file-from file-to)
              (rename-file file-from file-to)))))
      (delete-directory year-directory))))
```

```
(defun pamparam-check ()
  "Check the repo for inconsistencies and fix them.
     Check that all existing cards are scheduled, and only once.
     Check that there are no scheduled unexisting cards."
  (interactive)
  (let* ((default-directory (pamparam-default-directory))
         (all-cards (pamparam-cards default-directory))
         (all-schedules (delq nil
                              (mapcar
                               (lambda (s)
                                 (when (string-match "file:\\([^]]+\\)" s)
                                   (match-string 1 s)))
                               (pamparam-cmd-to-list
                                "git grep '^\\* TODO'"))))
         (unscheduled-cards (cl-set-difference
                             all-cards
                             all-schedules
                             :test #'equal))
         (unexisting-cards (cl-set-difference
                            all-schedules
                            all-cards
                            :test #'equal))
         (all-schedules-nodups (delete-dups (copy-sequence all-schedules)))
         (duplicate-cards (cl-set-difference all-schedules
         ↪  all-schedules-nodups)))
    (with-current-buffer (find-file-noselect "pampile.org")
      (goto-char (point-min))
      (dolist (card unscheduled-cards)
        (insert (format "* TODO [[file:%s][%s]]\n"
                        card (nth 1 (split-string card "[-.]")))))
      (save-buffer))
    (dolist (card (append duplicate-cards unexisting-cards))
      (let ((occurences (pamparam-cmd-to-list (format "git grep %s" card))))
        (dolist (occ (if (= (length occurences) 1)
                         occurences
                       (cdr occurences)))
          (with-current-buffer (find-file-noselect (car (split-string occ
          ↪  ":")))
            (goto-char (point-min))
            (re-search-forward card)
            (delete-region (line-beginning-position)
                           (1+ (line-end-position)))
            (save-buffer)))))))

(defun pamparam-reschedule-maybe ()
  (pamparam-carryover-year-maybe)
  (let ((today (calendar-current-date)))
    (unless (and pamparam-last-rechedule
                 (<
                  (calendar-absolute-from-gregorian today)
                  (calendar-absolute-from-gregorian pamparam-last-rechedule)))
      (setq pamparam-last-rechedule today)
      (let* ((today-file (pamparam-todo-file))
             (today-file-name (file-name-nondirectory
                               (buffer-file-name today-file)))
             (pdir (file-name-directory
```

```
                    (buffer-file-name today-file)))
             (all-files (directory-files pdir nil "org$"))
             (idx (cl-position today-file-name all-files
                               :test 'equal))
             (old-files (reverse (cl-subseq all-files 0 idx))))
        (dolist (old-file old-files)
          (setq old-file (expand-file-name old-file pdir))
          (let (cards)
            (with-current-buffer (find-file-noselect old-file)
              (goto-char (point-min))
              (while (re-search-forward "^\\* \\(TODO\\|REVIEW\\) " nil t)
                (push (buffer-substring-no-properties
                       (point) (1+ (line-end-position)))
                      cards)))
            (pamparam-schedule-today (mapcar (lambda (s) (concat "* TODO " s))
                                             (nreverse cards)))
            (delete-file old-file)))
        (with-current-buffer today-file
          (goto-char (point-min))
          (when (re-search-forward "^\\* TODO" nil t pamparam-day-limit)
            (beginning-of-line 2)
            (let ((rescheduled (buffer-substring-no-properties
                                (point) (point-max))))
              (delete-region (point) (point-max))
              (save-buffer)
              (with-current-buffer (pamparam-todo-file 1)
                (goto-char (point-max))
                (insert rescheduled)
                (save-buffer)))))))))

;;;###autoload
(defun pamparam-drill ()
  "Start a learning session.
     When `default-directory' is in a *.pam repository, use that repository.
     Otherwise, use the repository that `pamparam-path' points to.
     See `pamparam-sync' for creating and updating a *.pam repository.
     If you have no more cards scheduled for today, use `pamparam-pull'."
  (interactive)
  (pamparam-reschedule-maybe)
  (let (card-link card-file)
    (when (bound-and-true-p pamparam-card-mode)
      (when (buffer-modified-p)
        (pamparam-save-buffer))
      (kill-buffer))
    (delete-other-windows)
    (split-window-vertically)
    (pamparam-kill-buffers)
    (switch-to-buffer (pamparam-todo-file))
    (goto-char (point-min))
    (when (re-search-forward "^* \\(TODO\\|REVIEW\\) " nil t)
      (recenter 5)
      (setq card-link (buffer-substring-no-properties
                       (point) (line-end-position)))
      (beginning-of-line)
      (set-window-point (selected-window) (point)))
    (other-window 1)
    (if (null card-link)
```

```
        (message "%d cards learned/reviewed today. Well done!"
                 (cl-count-if
                  (lambda (x) (string-match "^\\* DONE" x))
                  (split-string (with-current-buffer (pamparam-todo-file)
                                  (buffer-string)) "\n")))
      (unless (string-match "\\`\\[\\[file:\\([^]]+\\)\\]\\[.*\\]\\]\\'"
      ↪  card-link)
        (error "Bad entry in %s: %s" (pamparam-todo-file) card-link))
      (setq card-file (match-string 1 card-link))
      (switch-to-buffer
       (find-file-noselect
        (expand-file-name card-file (pamparam-default-directory))))
      (pamparam-card-mode))))

(defun pamparam-commit ()
  "Commit the current progress using Git."
  (interactive)
  (let* ((repo-dir (pamparam-repo-directory (buffer-file-name)))
         (default-directory (if (file-exists-p repo-dir)
                                repo-dir
                              (pamparam-default-directory)))
         (status (pamparam-cmd-to-list "git status"))
         (card-count
          (cl-count-if
           (lambda (s)
             (or (string-match "modified.*cards/" s)
                 (string-match "new file.*cards/" s)))
           status))
         (card-str (if (= card-count 1)
                       "card"
                     "cards")))
    (message
     (replace-regexp-in-string
      "%" "%%"
      (shell-command-to-string
       (format
        "git add . && git commit -m 'Do %s %s'"
        card-count card-str))))))

(defun pamparam-unschedule-card (card-file)
  "Unschedule CARD-FILE everywhere and schedule it for today."
  (let* ((repo-dir (locate-dominating-file card-file ".git"))
         (s-files (pamparam-cmd-to-list (format "git add . && git grep
         ↪  --files-with-matches %s" (shell-quote-argument card-file))
                                        repo-dir)))
    (dolist (file s-files)
      (with-current-buffer (find-file-noselect (expand-file-name file
      ↪  repo-dir))
        (save-excursion
          (goto-char (point-min))
          (while (re-search-forward card-file nil t)
            (delete-region (line-beginning-position)
                           (1+ (line-end-position))))
          (let ((save-silently t))
            (pamparam-save-buffer)))
        (unless (equal (current-buffer) (pamparam-todo-file))
          (kill-buffer))))
```

```
    (with-current-buffer (pamparam-todo-file)
      (pamparam-goto-schedule-part)
      (if (re-search-forward "^\\* \\(TODO\\|REVIEW\\)" nil t)
          (goto-char (match-beginning 0))
        (goto-char (point-max)))
      (insert (pamparam--todo-from-file card-file)))))

(defun pamparam-card-redo ()
  "Redo the current card without penalty."
  (interactive)
  (if (string-match-p "cards/.*org\\'" (buffer-file-name))
      (let ((fname (buffer-file-name)))
        (pamparam-save-buffer)
        (pamparam-cmd-to-list (format "git checkout -- %s"
        ↪  (shell-quote-argument fname)))
        (revert-buffer nil t nil)
        (pamparam-unschedule-card (file-name-nondirectory fname))
        (setq-local pamparam-is-redo t)
        (pamparam-card-mode))
    (user-error "Applies only to card files")))

(defun pamparam-shifttab ()
  "Hide/show everything."
  (interactive)
  (let ((inhibit-message t))
    (when (eq org-cycle-global-status 'overview)
      (setq org-cycle-global-status 'contents))
    (setq this-command last-command)
    (org-cycle-internal-global)))

;;* `pamparam-card-mode'
(defvar pamparam-card-mode-map
  (let ((map (make-sparse-keymap)))
    (worf-define-key map (kbd "q") 'bury-buffer)
    (worf-define-key map (kbd "R") 'pamparam-card-redo
                     :break t)
    (worf-define-key map (kbd "n") 'pamparam-drill
                     :break t)
    (worf-define-key map (kbd "D") 'pamparam-card-delete)
    (define-key map (kbd ".") 'pamparam-card-validate-maybe)
    (define-key map (kbd "M-m") 'pamparam-card-manual-score)
    (define-key map (kbd "<S-iso-lefttab>") 'pamparam-shifttab)
    map))

(define-minor-mode pamparam-card-mode
  "Minor mode for Pam cards.
\\{pamparam-card-mode-map}"
  :lighter " p"
  (when pamparam-card-mode
    (if (eq major-mode 'org-mode)
        (progn
          (pamparam-card-abbreviate)
          (setq-local mode-line-format
                      `((pamparam-card-mode
                         (:eval (pamparam-mode-line)))
                        ,@(assq-delete-all
                           'pamparam-card-mode
```

```
                          (default-value 'mode-line-format))))
          (force-mode-line-update t)
          (setq org-cycle-global-status 'contents)
          (goto-char (point-min))
          (pamparam-card-answer))
      (pamparam-card-mode -1))))

(lispy-mode t)
(lispy-raise-minor-mode 'pamparam-card-mode)

(defun pamparam-latin ()
  (interactive)
  (find-file "~/dox/pamparam/latin/Latin.org"))

(defun pamparam-tagalog ()
  (interactive)
  (find-file "~/dox/pamparam/tagalog/Tagalog.org"))

(defun pamparam-drill-latin ()
  (interactive)
  (find-file "~/dox/pamparam/latin/Latin.pam")
  (pamparam-drill))

(defun pamparam-drill-tagalog ()
  (interactive)
  (find-file "~/dox/pamparam/tagalog/Tagalog.pam")
  (pamparam-drill))

(defun pamparam-magit-commit ()
  (interactive)
  (find-file "~/dox/pamparam/")
  (magit)
  )

(defun pamparam-push ()
  (interactive)
  (async-shell-command "cd ~/dox/pamparam/ && ~/dox/pamparam/update.sh")
  )

;;* `hydra-pamparam'
(defhydra hydra-pamparam (:exit t)
  "pam"
  ("t" pamparam-drill-tagalog "tagalog")
  ("l" pamparam-drill-latin "latin")
  ("d" pamparam-drill "drill")
  ("s" pamparam-sync "sync")
  ("m" pamparam-pull "more cards")
  ("p" pamparam-push "push")
  ("gl" pamparam-latin "goto latin")
  ("gt" pamparam-tagalog "goto tagalog")
  ("q" nil "quit"))
(hydra-set-property 'hydra-pamparam :verbosity 1)

(global-set-key (kbd "C-c v") 'hydra-pamparam/body)

(setq pamparam-path "/home/user/dox/pamparam/pamparam.pam")
```

```
(provide 'spaced-repetition)
```

4.19 Accounting

```
(load-module 'accounting)
```

4.19.1 Requirements

4.19.2 Code

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

(use-package! beancount
  :mode ("\\.beancount\\'" . beancount-mode)
  :init
  (after! all-the-icons
    (add-to-list 'all-the-icons-icon-alist
                 '("\\.beancount\\'" all-the-icons-material "attach_money"
                 ↪  :face all-the-icons-lblue))
    (add-to-list 'all-the-icons-mode-icon-alist
                 '(beancount-mode all-the-icons-material "attach_money" :face
                 ↪  all-the-icons-lblue)))
  :config
  (setq beancount-electric-currency t)
  (defun beancount-bal ()
    "Run bean-report bal."
    (interactive)
    (let ((compilation-read-command nil))
      (beancount--run "bean-report"
                      (file-relative-name buffer-file-name) "bal")))
  (map! :map beancount-mode-map
        :n "TAB" #'beancount-align-to-previous-number
        :i "RET" (cmd! (newline-and-indent)
        ↪  (beancount-align-to-previous-number))))

(provide 'accounting)
```

4.20 Popes

```
(load-module 'popes)
```

4.20.1 Requirements

4.20.2 Code

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

(defun get-pope-image ()
  (let* (
         (folder (concat doom-private-dir "/scripts/popes/images/"))
         (files (directory-files folder nil "\\.png\\'"))
         (number-popes (length files))
         (pope-img (nth (random number-popes) files)))
```

```
    (setq currently-displayed-pope (replace-regexp-in-string ".png" ""
    ↪  pope-img))
    (setq pope-info (shell-command-to-string (concat "grep -m1 \""
    ↪  currently-displayed-pope "\" " folder "../pope_info.txt")))
    (concat folder pope-img)
    )
  )


(defun psalm ()
  (shell-command-to-string "~/.scripts/psalms.sh de"))


(setq fancy-splash-last-size nil)
(setq fancy-splash-last-theme nil)
(defun set-appropriate-splash (&rest _)
   (setq fancy-splash-image (get-pope-image))
   (setq +doom-dashboard-banner-padding '(5 . 5))
   (setq fancy-splash-last-theme doom-theme)
   (+doom-dashboard-reload))

(add-hook 'window-size-change-functions #'set-appropriate-splash)
(add-hook 'doom-load-theme-hook #'set-appropriate-splash)

(defun doom-dashboard-phrase ()
  "Get a splash phrase, flow it over multiple lines as needed, and make fontify
  ↪  it."
  (mapconcat
   (lambda (line)
     (+doom-dashboard--center
      +doom-dashboard--width
      (with-temp-buffer
        (insert-text-button
         line
         'action
         (lambda (_) (+doom-dashboard-reload t))
         'face 'doom-dashboard-menu-title
         'mouse-face 'doom-dashboard-menu-title
         'help-echo currently-displayed-pope
         'follow-link t)
        (buffer-string))))
   (split-string
    (with-temp-buffer
      (insert (concat "Seine Heiligkeit " currently-displayed-pope "\n\n"
      ↪  pope-info "\n" "Heiliger Vater, bete für uns." "\n\n"))
      (setq fill-column (min 70 (/ (* 2 (window-width)) 3)))
      (fill-region (point-min) (point-max))
      (buffer-string))
    "\n")
   "\n"))

(defadvice! doom-dashboard-widget-loaded-with-phrase ()
  :override #'doom-dashboard-widget-loaded
  (setq line-spacing 0.2)
  (insert
   "\n\n"
   (propertize
```

```
    (+doom-dashboard--center
     +doom-dashboard--width
     (doom-display-benchmark-h 'return))
    'face 'doom-dashboard-loaded)
   "\n"
   (doom-dashboard-phrase)
   (psalm)
   "\n"))

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

(provide 'popes)
```

4.21 Keycast Tweaks

```
(load-module 'keycast-tweaks)
```

4.21.1 Requirements

4.21.2 Code

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

(use-package! keycast
  :commands keycast-mode
  :config
  (define-minor-mode keycast-mode
    "Show current command and its key binding in the mode line."
    :global t
    (if keycast-mode
        (progn
          (add-hook 'pre-command-hook 'keycast--update t)
          (add-to-list 'global-mode-string '("" mode-line-keycast " ")))
      (remove-hook 'pre-command-hook 'keycast--update)
      (setq global-mode-string (remove '("" mode-line-keycast " ")
      ↪  global-mode-string))))
  (custom-set-faces!
    '(keycast-command :inherit doom-modeline-debug
                      :height 0.9)
    '(keycast-key :inherit custom-modified
                  :height 1.1
                  :weight bold)))

(provide 'keycast-tweaks)
```

4.22 Weather

```
(load-module 'weather)
```

4.22.1 Requirements

4.22.2 Code

```
;;; wttrin.el --- Emacs frontend for weather web service wttr.in -*-
↪  lexical-binding: t; -*-
;; Copyright (C) 2016 Carl X. Su

;; Author: Carl X. Su <bcbcarl@gmail.com>
;;         ono hiroko (kuanyui) <azazabc123@gmail.com>
;; Version: 0.2.0
;; Package-Requires: ((emacs "24.4") (xterm-color "1.0"))
;; Keywords: comm, weather, wttrin
;; URL: https://github.com/bcbcarl/emacs-wttrin
;;
;; Modifications made by @tecosaur

;;; Commentary:

;; Provides the weather information from wttr.in based on your query condition.

;;; Code:

(require 'url)
(require 'xterm-color)

(defgroup wttrin nil
  "Emacs frontend for weather web service wttr.in."
  :prefix "wttrin-"
  :group 'comm)

(defcustom wttrin-default-api-version 1
  "Specifies which version of the wttrin API to use."
  :group 'wttrn
  :type '(choice (const 1) (const 2)))

(defcustom wttrin-default-cities '("Amsterdam"
                                   "Baghdad"
                                   "Beijing"
                                   "Brussels"
                                   "Buenos Aires"
                                   "Cairo"
                                   "Delhi"
                                   "Gurnsey"
                                   "Ho Chi Ming City"
                                   "Hong Kong"
                                   "Istanbul"
                                   "Johannesburg"
                                   "Köln"
                                   "Kuala Lumpur"
                                   "Leipzig"
                                   "Lima"
                                   "London"
                                   "Madrid"
                                   "Manila"
                                   "Mexico City"
                                   "Miami"
```

```
                                    "Moscow"
                                    "Mumbai"
                                    "München"
                                    "New York"
                                    "Nijmegen"
                                    "Paris"
                                    "Seoul"
                                    "Shanghai"
                                    "Singapore"
                                    "Surat"
                                    "Sydney"
                                    "Tokyo"
                                    "Toronto"
                                    ;; and the fun one!
                                    "Moon")
  "Specify default cities list for quick completion."
  :group 'wttrin
  :type 'list)

(defcustom wttrin-default-accept-language '("Accept-Language" .
↪  "en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4")
  "Specify default HTTP request Header for Accept-Language."
  :group 'wttrin
  :type '(list)
  )

(defun wttrin-fetch-raw-string (query &optional api-version)
  "Get the weather information based on your QUERY."
  (unless api-version (setq api-version wttrin-default-api-version))
  (let ((url-user-agent "curl"))
    (add-to-list 'url-request-extra-headers wttrin-default-accept-language)
    (with-current-buffer
        (url-retrieve-synchronously
         (concat "http://v" (number-to-string api-version) ".wttr.in/" query)
         (lambda (status) (switch-to-buffer (current-buffer))))
      (decode-coding-string (buffer-string) 'utf-8))))

(defun wttrin-exit ()
  (interactive)
  (quit-window t))

(defun wttrin-query (city-name &optional api-version)
  "Query weather of CITY-NAME via wttrin, and display the result in new
  ↪  buffer."
  (let ((raw-string (wttrin-fetch-raw-string city-name api-version)))
    (if (string-match "ERROR" raw-string)
        (message "Cannot get weather data. Maybe you inputed a wrong city
        ↪  name?")
      (let ((buffer (get-buffer-create (format "*wttr.in - %s*" city-name))))
        (switch-to-buffer buffer)
        (setq buffer-read-only nil)
        (erase-buffer)
        (insert (xterm-color-filter raw-string))
        (goto-char (point-min))
        (save-excursion
          (re-search-forward "^$")
          (delete-region (point-min) (1+ (point))))
```

```
        (save-excursion
          (while  (re-search-forward "(B" nil t)
            (delete-region (match-beginning 0) (match-end 0))))
        (use-local-map (make-sparse-keymap))
        (local-set-key "q" 'wttrin-exit)
        (local-set-key "g" 'wttrin)
        (setq buffer-read-only t)))))

;;;###autoload
(defun wttrin (city &optional api-version)
  "Display weather information for CITY."
  (interactive
   (cond ((equal current-prefix-arg nil)
          (list "" nil))
         ((equal current-prefix-arg 1)
          (list "" 1))
         ((equal current-prefix-arg 2)
          (list "" 2))
         (t (list
             (completing-read "City name: " wttrin-default-cities nil nil
                              (when (= (length wttrin-default-cities) 1)
                                (car wttrin-default-cities)))))))
  (wttrin-query city api-version))
```

4.23 Org Tweaks

```
(load-module 'org-tweaks)
```

4.23.1 Requirements

```
(package! engrave-faces)
```

4.23.2 Code

```
;;; org-tweaks.el -*- lexical-binding: t; -*-
;;; Code:
(after! org
  (ifdirexists "~/sync/org/"
               (setq org-directory dir))
  (ifdirexists "~/sync/agenda"
               (setq org-agenda-files (directory-files "~/sync/agenda/" t (rx
               ↪  ".org" eos))))
  (setq org-todo-keywords '((sequence "TODO(t)" "LECT(l)" "EXAM(e)" "MEET(m)"
  ↪  "PROJ(p)" "LOOP(L)" "START(s)" "WAIT(w)" "HOLD(h)" "IDEA(i)" "INPRO(n)"
  ↪  "OPT(o)" "READ(r)" "|" "DONE(d)" "KILL(k)")
                            (sequence "[ ](T)" "[-](S)" "[?](W)" "|" "[X](D)")
                            (sequence "|" "OKAY(O)" "YES(Y)" "NO(N)"))
        org-startup-folded t
        org-log-done 'time
        org-log-reschedule 'time
        initial-major-mode 'org-mode
        org-export-async-init-file
        ↪  "/home/user/.doom.d/ext/export/org-export-init.el"
```

```
        org-latex-src-block-backend 'engraved)

  ;;; Add org mode to txt and archive files
  (add-to-list 'auto-mode-alist '("\\.\\(org\\|org_archive\\|txt\\)$" .
  ↪  org-mode))

  ;; Org Babel
  (setq org-confirm-babel-evaluate nil
        org-src-fontify-natively t
        org-src-tab-acts-natively t
        org-auto-tangle-default t)
  ;; Auto tangle
  (add-hook 'org-mode-hook 'org-auto-tangle-mode)
)

(setq beancount-main-file "/media/user/keychain/finances/wallet.beancount"
      beancount-local-file "~/dox/notes/wallet.org")
;; Capture
(setq org-default-notes-file "~/dox/notes/notes.org")
(setq org-capture-templates
      '(("b" "Beancount Entry" plain
         (file beancount-local-file)
         ;"%(progn (yas-expand-snippet (yas-lookup-snippet \"beancount\"
         ↪  'org-mode)) nil)")))
         "bc")))

(defun org-agenda-export-to-ics ()
  "Exports current org agenda buffer to ics, treating DEADLINES as dates"
  (interactive)
  (with-temp-buffer
    (cl-map 'nil #'insert-file-contents org-agenda-files)
    (replace-regexp-entire-buffer "<.*> \\(<.*>\\)" "\\1")
    (replace-regexp-entire-buffer "\\(<.*>\\) <.*>" "\\1")
    (replace-regexp-entire-buffer "SCHEDULED: \\(<.*>\\)" "\\1")
    (replace-regexp-entire-buffer "DEADLINE: \\(<.*>\\)" "\\1")
    (message (org-icalendar-export-to-ics))))

(provide 'org-tweaks)
```

4.24 Languages

```
(load-module 'languages)
```

4.24.1 Requirements

4.24.2 Code

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

;; Rust
(setq rustic-format-trigger 'on-save
      rustic-format-on-save t)

;; Latin
;;
```

```
;;;###autoload
(define-minor-mode latin-minor-mode
  "Minor mode for writing Church Latin"
  :lighter "  "
  :global t)

(defun latin-minor-mode--insert-ae ()
  "Replace ae with æ"
  (interactive)
  (if (bound-and-true-p latin-minor-mode)
  (if (eq (char-before) ?a)
      (progn
        (backward-delete-char 1)
        (insert "æ"))
    (if (eq (char-before) ?A)
        (progn
          (backward-delete-char 1)
          (insert "Æ")
          )
      (insert "e")))
  (self-insert-command 1)))

(defun latin-minor-mode--insert-versicle ()
  "Replace VV with  "
  (interactive)
  (if (bound-and-true-p latin-minor-mode)
  (if (eq (char-before) ?V)
      (progn
        (backward-delete-char 1)
        (insert " "))
      (insert "V"))
  (self-insert-command 1)))

(defun latin-minor-mode--insert-response ()
  "Replace RR with  "
  (interactive)
  (if (bound-and-true-p latin-minor-mode)
  (if (eq (char-before) ?R)
      (progn
        (backward-delete-char 1)
        (insert " "))
      (insert "R"))
  (self-insert-command 1)))



(map! :map latin-minor-mode-map
      :n "e" #'latin-minor-mode--insert-ae
      :n "R" #'latin-minor-mode--insert-response
      :n "V" #'latin-minor-mode--insert-versicle)

;; For some reason, that doesn't work...
;;(progn
;;  (global-set-key (kbd "e") 'latin-minor-mode--insert-ae)
;;  (global-set-key (kbd "R") 'latin-minor-mode--insert-response)
;;  (global-set-key (kbd "V") 'latin-minor-mode--insert-versicle))
```

```
(provide 'languages)
```

4.25 Email

```
(load-module-if 'mu4e 'email)
```

4.25.1 Requirements

4.25.2 Code

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

(setq mu4e-mu-binary "/bin/mu")

;;; mu4e reindexing when tmp file exists
(after! mu4e
  (defvar mu4e-reindex-request-file "/tmp/mu_reindex_now"
    "Location of the reindex request, signaled by existance")
  (defvar mu4e-reindex-request-min-seperation 5.0
    "Don't refresh again until this many second have elapsed.
       Prevents a series of redisplays from being called (when set to an
    ↪  appropriate value)")

  (defvar mu4e-reindex-request--file-watcher nil)
  (defvar mu4e-reindex-request--file-just-deleted nil)
  (defvar mu4e-reindex-request--last-time 0)

  (defun mu4e-reindex-request--add-watcher ()
    (setq mu4e-reindex-request--file-just-deleted nil)
    (setq mu4e-reindex-request--file-watcher
          (file-notify-add-watch mu4e-reindex-request-file
                                 '(change)
                                 #'mu4e-file-reindex-request)))

  (defadvice! mu4e-stop-watching-for-reindex-request ()
    :after #'mu4e~proc-kill
    (if mu4e-reindex-request--file-watcher
        (file-notify-rm-watch mu4e-reindex-request--file-watcher)))

  (defadvice! mu4e-watch-for-reindex-request ()
    :after #'mu4e~proc-start
    (mu4e-stop-watching-for-reindex-request)
    (when (file-exists-p mu4e-reindex-request-file)
      (delete-file mu4e-reindex-request-file))
    (mu4e-reindex-request--add-watcher))

  (defun mu4e-file-reindex-request (event)
    "Act based on the existance of `mu4e-reindex-request-file'"
    (if mu4e-reindex-request--file-just-deleted
        (mu4e-reindex-request--add-watcher)
      (when (equal (nth 1 event) 'created)
        (delete-file mu4e-reindex-request-file)
        (setq mu4e-reindex-request--file-just-deleted t)
        (mu4e-reindex-maybe t))))
```

```
  (defun mu4e-reindex-maybe (&optional new-request)
    "Run `mu4e~proc-index' if it's been more than
        `mu4e-reindex-request-min-seperation'seconds since the last request,"
    (let ((time-since-last-request (- (float-time)
                                      mu4e-reindex-request--last-time)))
      (when new-request
        (setq mu4e-reindex-request--last-time (float-time)))
      (if (> time-since-last-request mu4e-reindex-request-min-seperation)
          (mu4e~proc-index nil t)
        (when new-request
          (run-at-time (* 1.1 mu4e-reindex-request-min-seperation) nil
                       #'mu4e-reindex-maybe))))))

(provide 'email)
```

4.26 Email Config

```
(load-module-if 'mu4e 'email-config)
```

4.26.1 Requirements

4.26.2 Code

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

(require 'mu4e)
(require 'smtpmail)
(define-key mu4e-view-mode-map (kbd "f") 'mu4e-view-go-to-url)

(setq mu4e-root-maildir "~/mail"
      ;mu4e-get-mail-command "offlineimap -q -f INBOX"
      mu4e-get-mail-command "mbsync -a || true"
      mu4e-update-interval 300 ;; second
      mu4e-compose-signature-auto-include nil
      mu4e-view-show-images t
      mu4e-view-prefer-html t
      mu4e-html2text-command "iconv -c -t utf-8 | pandoc -f html -t plain"
      mu4e-headers-auto-update t
      mu4e-compose-format-flowed t
      smtpmail-stream-type 'starttls
      mu4e-view-show-addresses t
      mu4e-split-view 'single-window ;; horizontal (default), vertical
      mu4e-attachment-dir "~/Downloads"
      smtpmail-queue-mail nil
      smtpmail-queue-dir "~/mail/queue/cur"
      mu4e-compose-in-new-frame nil
      mu4e-compose-dont-reply-to-self t
      mu4e-headers-date-format "%Y-%m-%d %H:%M"
      message-kill-buffer-on-exit nil
      mu4e-confirm-quit nil
      mu4e-context-policy 'ask-if-none
      mu4e-compose-context-policy 'always-ask
      mu4e-headers-results-limit 500
      mu4e-use-fancy-chars t)
```

```
(defun mu4e--view-quit-and-back ()
  "Quit mu4e view buffer and go back to mu4e"
  (interactive)
  (mu4e~view-quit-buffer)
  (=mu4e))

(defun mu4e--goto-inbox ()
  "Goto mu4e inbox"
  (interactive)
  (mu4e~headers-jump-to-maildir "/gmail/INBOX"))

(map! :map mu4e-view-mode-map
      :after mu4e-view
      :n "<backspace>" 'mu4e--view-quit-and-back)

(global-set-key (kbd "s-m") 'mu4e--goto-inbox)

(when (fboundp 'imagemagick-register-types)
  (imagemagick-register-types))

;(require 'org-mu4e)
;(setq org-mu4e-convert-to-html t
;org-mu4e-link-query-in-headers-mode nil)

(require 'org-mime)

;; this seems to fix the babel file saving thing
(defun org-mu4e-mime-replace-images (str current-file)
"Replace images in html files with cid links."
(let (html-images)
(cons
(replace-regexp-in-string ;; replace images in html
"src=\"\\([^\"]+\\)\""
(lambda (text)
(format
        "src=\"./:%s\""
        (let* ((url (and (string-match "src=\"\\([^\"]+\\)\"" text)
                         (match-string 1 text)))
        (path (expand-file-name
                url (file-name-directory current-file)))
        (ext (file-name-extension path))
        (id (replace-regexp-in-string "[\/\\\\]" "_" path)))
        (add-to-list 'html-images
                (org-mu4e-mime-file
                (concat "image/" ext) path id))
        id)))
str)
html-images)))

(add-to-list 'mu4e-view-actions
'("ViewInBrowser" . mu4e-action-view-in-browser) t)


(provide 'email-config)
```

4.27 Email Accounts

```
(load-module-if 'mu4e 'email-accounts)
```

4.27.1 Requirements

4.27.2 Code

```
;; Email Accounts
(require 'email-refile)
(require 'smtpmail)
(setq +mu4e-gmail-accounts '(("e.p.mysliwietz@gmail.com" . "/gmail"))
      mu4e-contexts
      `(
       ,(make-mu4e-context
          :name "gmail"
          :enter-func (lambda () (mu4e-message "Switching to gmail context"))
          :leave-func (lambda () (mu4e-message "Leaving gmail context"))
          :vars '(
                  ( user-full-name                  . "Egidius Mysliwietz" )
                  ( user-mail-address                 .
                  ↪  "e.p.mysliwietz@gmail.com" )
                  ( smtpmail-mail-address         . "e.p.mysliwietz@gmail.com")
                  ( smtpmail-smtp-user            . "e.p.mysliwietz@gmail.com")
                  ( mu4e-drafts-folder            . "/gmail/[Gmail]/Entwürfe" )
                  ( mu4e-sent-folder              . "/gmail/[Gmail]/Gesendet" )
                  ( mu4e-trash-folder             . "/gmail/[Gmail]/Papierkorb"
                  ↪  )
                  ( mu4e-refile-folder            . gmail-refile )
                  ( smtpmail-default-smtp-server . "smtp.gmail.com" )
                  ( smtpmail-smtp-server          . "smtp.gmail.com" )
                  ( smtpmail-local-domain         . "gmail.com" )
                  ( smtpmail-smtp-service         . 587 )
                  ))
        ,(make-mu4e-context
          :name "emysliwietz"
          :enter-func (lambda () (mu4e-message "Switching to egidius context"))
          :leave-func (lambda () (mu4e-message "Leaving egidius context"))
          ;; we match based on the contact-fields of the message
          :vars '(
                  ( user-full-name                . "Egidius Mysliwietz" )
                  ( user-mail-address                 . "egidius@mysliwietz.de"
                  ↪  )
                  ( smtpmail-mail-address         . "egidius@mysliwietz.de")
                  ( smtpmail-smtp-user            . "egidius@mysliwietz.de")
                  ( mu4e-archive-folder           . "/egidius/Archive" )
                  ( mu4e-drafts-folder            . "/egidius/Drafts" )
                  ( mu4e-sent-folder              . "/egidius/Sent" )
                  ( mu4e-trash-folder             . "/egidius/Trash" )
                  ( mu4e-refile-folder            . gmail-refile )
                  ( smtpmail-default-smtp-server . "smtp.strato.de" )
                  ( smtpmail-smtp-server          . "smtp.strato.de" )
                  ( smtpmail-local-domain         . "strato.de" )
                  ( smtpmail-smtp-service         . 465 )
                  ))
      ;,(make-mu4e-context
```

```
      ;:name "xgmx"
      ;:enter-func (lambda () (mu4e-message "Switching to gmx context"))
      ;:leave-func (lambda () (mu4e-message "Leaving gmx context"))
      ;:vars '(
              ;( user-full-name                . "Egidius Mysliwietz" )
              ;( user-mail-address               .
              ↪  "egidius.mysliwietz@gmx.de" )
              ;( smtpmail-mail-address        .
              ↪  "egidius.mysliwietz@gmx.de")
              ;( smtpmail-smtp-user           .
              ↪  "egidius.mysliwietz@gmx.de")
              ;( mu4e-archive-folder          . "/gmx/Archiv" )
              ;( mu4e-drafts-folder           . "/gmx/Entwürfe" )
              ;( mu4e-sent-folder             . "/gmx/Gesendet" )
              ;( mu4e-trash-folder            . "/gmx/Gelöscht" )
              ;( smtpmail-default-smtp-server . "mail.gmx.net" )
              ;( smtpmail-smtp-server         . "mail.gmx.net" )
              ;( smtpmail-local-domain        . "gmx.net" )
              ;( smtpmail-smtp-service        . 465 )
              ;))
    ;,(make-mu4e-context
      ;:name "radboud"
      ;:enter-func (lambda () (mu4e-message "Switch to radboud context"))
      ;:leave-func (lambda () (mu4e-message "Leaving radboud context"))
      ;:vars '(
              ;( user-full-name                . "Egidius Mysliwietz" )
              ;( user-mail-address               .
              ↪  "e.mysliwietz@student.ru.nl" )
              ;( smtpmail-mail-address        .
              ↪  "e.mysliwietz@student.ru.nl")
              ;( smtpmail-smtp-user           . "s1000796")
              ;( mu4e-drafts-folder           . "/radboud/Drafts" )
              ;( mu4e-sent-folder             . "/radboud/Sent Items" )
              ;( mu4e-trash-folder            . "/radboud/Trash" )
              ;( smtpmail-default-smtp-server . "smtp-auth.ru.nl" )
              ;( smtpmail-smtp-server         . "smtp-auth.ru.nl" )
              ;( smtpmail-local-domain        . "ru.nl" )
              ;( smtpmail-smtp-service        . 587 )
              ;))
    ;,(make-mu4e-context
      ;:name "eindhoven"
      ;:enter-func (lambda () (mu4e-message "Switch to eindhoven context"))
      ;:leave-func (lambda () (mu4e-message "Leaving eindhoven context"))
      ;:vars '(
              ;( user-full-name                . "Egidius Mysliwietz" )
              ;( user-mail-address               .
              ↪  "e.p.j.mysliwietz@student.tue.nl" )
              ;( smtpmail-mail-address        .
              ↪  "e.p.j.mysliwietz@student.tue.nl")
              ;( smtpmail-smtp-user           .
              ↪  "e.p.j.mysliwietz@student.tue.nl")
              ;( mu4e-archive-folder          . "/eindhoven/Archive" )
              ;( mu4e-drafts-folder           . "/eindhoven/Drafts" )
              ;( mu4e-sent-folder             . "/eindhoven/Sent Items" )
              ;( mu4e-trash-folder            . "/eindhoven/Trash" )
              ;( smtpmail-default-smtp-server . "smtp.office365.com" )
              ;( smtpmail-smtp-server         . "smtp.office365.com" )
```

```
                ;( smtpmail-local-domain        . "office365.com" )
                ;( smtpmail-smtp-service        . 587 )
                ;))
       ;,(make-mu4e-context
          ;:name "ntu"
          ;:enter-func (lambda () (mu4e-message "Switch to ntu context"))
          ;:leave-func (lambda () (mu4e-message "Leaving ntu context"))
          ;:vars '(
                ;( user-full-name                . "Egidius Mysliwietz" )
                ;( user-mail-address               .
                ↪  "n1903483e@e.ntu.edu.sg" )
                ;( smtpmail-mail-address        . "n1903483e@e.ntu.edu.sg")
                ;( smtpmail-smtp-user           . "n1903483e@e.ntu.edu.sg")
                ;( mu4e-archive-folder          . "/ntu/Archive" )
                ;( mu4e-drafts-folder           . "/ntu/Drafts" )
                ;( mu4e-sent-folder             . "/ntu/Sent Items" )
                ;( mu4e-trash-folder            . "/ntu/Trash" )
                ;( smtpmail-default-smtp-server . "smtp.office365.com" )
                ;( smtpmail-smtp-server         . "smtp.office365.com" )
                ;( smtpmail-local-domain        . "office365.com" )
                ;( smtpmail-smtp-service        . 587 )
                ;))
       ))


(provide 'email-accounts)
```

4.28 Latex Tweaks

```
(load-module 'latex-tweaks)
```

4.28.1 Requirements

4.28.2 Code

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

(setq TeX-command-extra-options "--shell-escape")
(setq TeX-engine 'luatex)
(add-hook 'org-mode-hook
          (lambda ()
            (local-set-key (kbd "<f6>")
                           (lambda () (interactive) (org-latex-export-to-pdf
                           ↪  t)))))
(defun force-compile ()
  "Set the file modification times on the current file, then call
      TeX-command-sequence.
      This forces a complete recompilation of the document, even if the source
      (.tex) is older than any existing outputs (.pdf etc)."
  (interactive)
  (set-buffer-modified-p t) ;; sets mod time to current time
  (save-buffer)
  (TeX-command-sequence t t))

(defun auto-async-export ()
```

```
  (when (and (eq major-mode 'org-mode)
             (string-equal "t" (pop (cdr (car (org-collect-keywords
             ↪  '("auto_async_export")))))))
    (message "Exporting to pdf...")
    (org-latex-export-to-pdf t)
    (when (eq major-mode 'latex-mode)
      (force-compile))
    )
  )

(defun org-after-save-cmd ()
  (interactive)
  (when (eq major-mode 'org-mode)
  (let ((cmd (cdr (car (org-collect-keywords '("on_save_cmd"))))))
        (when cmd
  (async-shell-command-no-window (pop cmd))))))
(setq password-cache t ; enable password caching
      password-cache-expiry 36000) ; for ten hours (time in secs)

(add-hook 'after-save-hook 'auto-async-export)
(add-hook 'after-save-hook 'org-after-save-cmd)
                                       ;(global-set-key [f6] (lambda ()
                                       ↪  (interactive)
                                       ↪  (org-latex-export-to-pdf t)))
(setq org-latex-compiler "lualatex")
(setq-default TeX-master nil)
(add-to-list 'org-latex-packages-alist
             '("AUTO" "polyglossia" t ("xelatex" "lualatex")))

(defun latex-word-count ()
  "Return latex word count using texlive"
  (interactive)
  (let ((file-name (if (eq major-mode 'latex-mode)
                       (buffer-file-name)
                     (if (eq major-mode 'org-mode)
                         (file-name-with-extension (buffer-file-name) "tex")
                       (buffer-file-name)))))
  (message (shell-command-to-string (format "texcount -1 -merge -template={1}
  ↪  %s" file-name)))))

  ;; Latex classes
  (setq org-latex-subtitle-separate t
        org-latex-subtitle-format "\\subtitle{%s}")
  (setq org-latex-classes '(("article" "\\documentclass[a4wide,10pt]{article}"
                             ("\\section{%s}" . "\\section*{%s}")
                             ("\\subsection{%s}" . "\\subsection*{%s}")
                             ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                             ("\\paragraph{%s}" . "\\paragraph*{%s}")
                             ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))
                            ("report" "\\documentclass[11pt]{report}"
                             ("\\part{%s}" . "\\part*{%s}")
                             ("\\chapter{%s}" . "\\chapter*{%s}")
                             ("\\section{%s}" . "\\section*{%s}")
                             ("\\subsection{%s}" . "\\subsection*{%s}")
                             ("\\subsubsection{%s}" . "\\subsubsection*{%s}"))
```

```
                            ("artikel" "\\documentclass[fancy, modern,
                            ↪  twocolumn, titlepage=head, paper=a4,
                            ↪  12pt]{artikel}"
                             ("\\section{%s}" . "\\section*{%s}")
                             ("\\subsection{%s}" . "\\subsection*{%s}")
                             ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                             ("\\paragraph{%s}" . "\\paragraph*{%s}")
                             ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))
                            ("thesis" "\\documentclass[fancy, modern,
                            ↪  twocolumn, titlepage=thesis, paper=a4,
                            ↪  12pt]{artikel}"
                             ("\\section{%s}" . "\\section*{%s}")
                             ("\\subsection{%s}" . "\\subsection*{%s}")
                             ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                             ("\\paragraph{%s}" . "\\paragraph*{%s}")
                             ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))
                            ("book" "\\documentclass[11pt]{book}"
                             ("\\part{%s}" . "\\part*{%s}")
                             ("\\chapter{%s}" . "\\chapter*{%s}")
                             ("\\section{%s}" . "\\section*{%s}")
                             ("\\subsection{%s}" . "\\subsection*{%s}")
                             ("\\subsubsection{%s}" .
                             ↪  "\\subsubsection*{%s}"))))

(setq org-export-headline-levels 5)

; Allow headlines, but not content, to not be exported (used for structuring
↪  org files)
(require 'ox-extra)
(ox-extras-activate '(ignore-headlines))


(let* ((base-dir "/home/user/dox/bib/")
       (bibfile (concat base-dir "bib.bib"))
      (notes-dir (concat base-dir "notes/"))
      (lib-dir (concat base-dir "papers/")))
  (setq bibtex-completion-bibliography bibfile
        citar-bibliography `(,bibfile)
        bibtex-completion-library-path `(,lib-dir)
        bibtex-completion-notes-path notes-dir
        citar-library-paths `(,lib-dir)
        citar-notes-paths `(,notes-dir)
        org-cite-global-bibliography `(,bibfile)))

(setq org-export-with-sub-superscripts "{}"
      org-export-with-smart-quotes nil)

(engrave-faces-use-theme 'doom-one)

(setq org-preview-latex-process-alist
   '((dvipng :programs
      ("dvilualatex" "dvipng")
      :description "dvi > png" :message "you need to install the programs:
      ↪  dvilualatex and dvipng." :image-input-type "dvi" :image-output-type
      ↪  "png" :image-size-adjust
      (1.0 . 1.0
)
```

```
      :latex-compiler
      ("dvilualatex -interaction nonstopmode -output-directory %o %f")
      :image-converter
      ("dvipng -D %D -T tight -o %O %f")
      :transparent-image-converter
      ("dvipng -D %D -T tight -bg Transparent -o %O %f"))
     (dvisvgm :programs
      ("lualatex" "dvisvgm")
      :description "dvi > svg" :message "you need to install the programs:
      ↪  lualatex and dvisvgm." :image-input-type "dvi" :image-output-type
      ↪  "svg" :image-size-adjust
      (1.7 . 1.5)
      :latex-compiler
      ("lualatex -interaction nonstopmode -output-directory %o %f")
      :image-converter
      ("dvisvgm %f -n -b min -c %S -o %O"))
     (imagemagick :programs
      ("lualatex" "convert")
      :description "pdf > png" :message "you need to install the programs:
      ↪  lualatex and imagemagick." :image-input-type "pdf" :image-output-type
      ↪  "png" :image-size-adjust
      (1.0 . 1.0)
      :latex-compiler
      ("lualatex -interaction nonstopmode -output-directory %o %f")
      :image-converter
      ("convert -density %D -trim -antialias %f -quality 100 %O"))))

(provide 'latex-tweaks)
```

4.29 Org Links

```
(load-module 'org-links)
```

4.29.1 Requirements

4.29.2 Code

```
;;; ol-man.el - Support for links to man pages in Org mode
(require 'ol)

(org-link-set-parameters "b"
                         :follow #'org-b-open
                         :export #'org-b-export
                         :store #'org-b-store-link)

(defun org-b-open (verse _)
  "Visit the verse."
  (funcall org-b-command path))

(defun org-b-store-link ()
  "Store a link to a man page."
  (when (memq major-mode '(Man-mode woman-mode))
    ;; This is a man page, we do make this link.
    (let* ((page (org-man-get-page-name))
           (link (concat "b:" page))
```

```
           (description (format "B page for %s" page)))
      (org-link-store-props
       :type "b"
       :link link
       :description description))))

(defun org-man-get-page-name ()
  "Extract the page name from the buffer name."
  ;; This works for both `Man-mode' and `woman-mode'.
  (if (string-match " \\(\\S-+\\)\\*" (buffer-name))
      (match-string 1 (buffer-name))
    (error "Cannot create link to this man page")))

(defun org-b-export (link description format _)
  "Export a man page link from Org files."
  (let ((path (format "http://man.he.net/?topic=%s&section=all" link))
        (desc (or description link)))
    (pcase format
      (`html (format "<a target=\"_blank\" href=\"%s\">%s</a>" path desc))
      (`latex (format "\\href{%s}{%s}" path desc))
      (`texinfo (format "@uref{%s,%s}" path desc))
      (`ascii (format "%s (%s)" desc path))
      (t path))))

(provide 'org-links)
```