Emacs config

Egidius Mysliwietz

January 14, 2025

Contents

1 Config

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

;; 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 '(default "secrets:Passwords" "~/.authinfo.gpg")
      auth-sources '(default "~/.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")

; Make sure compile warnings are shown immediately even if init is incomplete
↪  to preserve knowledge of files and location
(defun dont-delay-compile-warnings (fun type &rest args)
  (if (eq type 'bytecomp)
      (let ((after-init-time t))
        (apply fun type args))
    (apply fun type args)))
(advice-add 'display-warning :around #'dont-delay-compile-warnings)

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

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

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

(defmacro after-startup (&rest func)
  `(unless debug-my-config (add-hook! 'after-startup-hook
                             '(lambda () ,@func))))

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

(defmacro execution-time (func)
  `(let ((time (current-time)))
     ,func
     (float-time (time-since time))))

(defmacro execution-time-format (func)
  `(let ((time (current-time)))
     ,func
     (message (format "Loaded %s in %.06f." ',func (float-time (time-since
     ↪  time))))))

(defmacro load-module (module)
   `(when
        (or (not debug-my-config) (> debug-my-config 0))
      (-- debug-my-config)
      (make-thread
       (let* ((benchmark (benchmark-run (require ,module)))
              (time (car benchmark))
              (gbc (nth 1 benchmark))
              (gbt (nth 2 benchmark)))
         (message "%s - Loaded %s in %.06fs, using %s garbage collections in
         ↪  %.06fs."
                  ,debug-my-config ,module time gbc gbt)))))

;; 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 'private-config)
(load-module 'exwm)
(load-module 'exwm-randr)

; 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))))

(defun indent-buffer ()
  "Indent each nonblank line in the buffer"
  (interactive)
  (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! try)
(package! counsel)

(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! rotate)
(package! vlf)
(package! evil-escape :disable t)

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

(package! string-inflection)
(package! info-colors)
(package! modus-themes)
;(package! theme-magic)
```

```
(package! keycast)
(package! page-break-lines :recipe (:host github :repo
↪  "purcell/page-break-lines"))
(package! spray)
;(package! authinfo-color-mode
;  :recipe (:repo "lisp/authinfo-color-mode"))
(package! systemd)
; Ebooks
(package! calibredb)
(package! nov)
; 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 DPI Scaling Tweaks

```
(load-module 'dpi-scaling-tweaks)
```

4.1.1 Requirements

4.1.2 Code

```
(setq display-pixels-per-inch (getenv "XFT_DPI"))
(provide 'dpi-scaling-tweaks)
(if (string= (getenv "XFT_DPI") "192")
    (setq exwm-systemtray--icon-min-size 32)
  (setq exwm-systemtray--icon-min-size 16)
  )
```

4.2 Utility functions

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

4.2.1 Code

```
(setq home-hostname-alist '("astaroth" "jarvis")
      home? (member (system-name) home-hostname-alist))

(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)))

;;; 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))
  (message (format "Class: %s" exwm-class-name))
  (message (format "Instance: %s" exwm-instance-name))
  (message (format "Title: %s" exwm-title))
  (message (format "Type: %s" exwm-window-type))
  ;(message "%s" (exwm-class-name (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"))

(defmacro ifdirexists (dir &rest actions)
  "Execute functions taking dir as an argument if dir exists"
```

```
  `(when (file-exists-p ,dir)
     ((lambda (dir)
       ,@actions) ,dir)))

(provide 'utility-functions)
```

4.3 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)

(setq ispell-abbrev-local nil)
```

```
(defun ispell-abbrev-advice (ispell-func &rest args)
  "Advice for `ispell-word'. Creates an abbrev for the correction made.
Unless ispell-abbrev-local, abbrev will be global.
ISPELL-FUNC passed as adviced function."
  (let ((before (downcase (or (thing-at-point 'word) "")))
        after
        (res (apply ispell-func args)))

    (setq after (downcase (or (thing-at-point 'word) "")))
    (unless (string= after before)
      (define-abbrev
        (if ispell-abbrev-local local-abbrev-table global-abbrev-table) before
        ↪  after))
    (message "\"%s\" now expands to \"%s\" %sally."
             before after (if ispell-abbrev-local "loc" "glob"))
    res))

(advice-add 'ispell-word :around #'ispell-abbrev-advice)

(provide 'auto-correct)
```

4.4 Beancount

4.4.1 Requirements

4.4.2 Code

```
(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))))

(defun beancount-select-account ()
```

```
  "Select and insert account from current buffer"
  (interactive)
  (with-temp-buffer
    (beancount-open-local)
    (completing-read "Account: " (beancount-collect-unique
    ↪  beancount-account-regexp 0))))

(defun beancount-imported-transaction-change-unknown-account ()
  "Change the Unknown:account field in an imported beancount entry."
  (interactive)
  (save-excursion
    (beancount-goto-transaction-begin)
    (let ((ba (beancount-select-account)))
      (re-search-forward "Unknown:account")
      (replace-match ba)
      ))
  (beancount-finalize-transaction)
  (beancount-goto-next-transaction)
  )

(defun beancount-imported-credit-transaction-change-unknown-account ()
  "Change the Unknown:account field in an imported VR-Visa-Gold beancount
  ↪  entry."
  (interactive)
  (save-excursion
    (beancount-goto-transaction-begin)
    (re-search-forward "Assets:VR-Giro")
    (replace-match "Assets:VR-Visa-Gold")
    )
  (beancount-imported-transaction-change-unknown-account)
  )

(defun beancount-finalize-transaction ()
  "Change transaction marked with * into *"
  (interactive)
  (save-excursion
    (beancount-goto-transaction-begin)
    (re-search-forward "!")
    (replace-match "*")
    )
  )

(defun beancount-transaction-align ()
  "Align beancount transaction."
  (interactive)
  (beancount-align-to-previous-number)
  (beancount-goto-next-transaction)
  )

(defun end-of-buffer-p ()
  "Check if cursor is at the end of the buffer."
  (interactive)
  (= (point) (point-max)))


(defun recenter-middle ()
  "Like recenter-top-bottom, but only centers in middle"
```

```
  (interactive)
  (recenter nil t))

; Center screen when going to next or prev transaction
(advice-add '+beancount/next-transaction :after #'recenter-middle)
(advice-add '+beancount/previous-transaction :after #'recenter-middle)

(defun beancount-cleanup-whitespace-in-quotes (str)
  "Replace multiple spaces in STR with a single space, but only between
  ↪  quotes."
  (let ((start 0)
        (res ""))
    (while (string-match "\\(\"[^\"]*\"\\)" str start)
      (let ((match (match-string 0 str)))
        (setq res (concat res
                          (substring str start (match-beginning 0))
                          (replace-regexp-in-string "[[:space:]]\\{2,\\}" " "
                          ↪  match)))
        (setq start (match-end 0))))
    (concat res (substring str start))))

(defun beancount-cleanup-buffer ()
  "Cleanup whitespace in the current buffer."
  (interactive)
  (save-excursion
    (goto-char (point-min))
    (while (not (eobp))
      (let* ((line (buffer-substring-no-properties (line-beginning-position)
      ↪  (line-end-position)))
             (clean-line (beancount-cleanup-whitespace-in-quotes line)))
        (delete-region (line-beginning-position) (line-end-position))
        (insert clean-line)
        (forward-line 1)))))

(defun beancount-cleanup-hook ()
  "Add `beancount-cleanup-buffer` to `before-save-hook` in Beancount mode."
  (when (eq major-mode 'beancount-mode)
    (add-hook 'before-save-hook #'beancount-cleanup-buffer nil t)))

(add-hook 'beancount-mode-hook #'beancount-cleanup-hook)


  (defun beancount-note-processing-date ()
  "Adds the date the transaction was filed/processed on as note."
  (interactive)
  (save-excursion
    (beancount-goto-transaction-begin)
    (let ((start (point)))
      (evil-forward-WORD-end)
      (forward-char)
      (let ((end (point)))
        (let ((word (buffer-substring start end)))
          (beancount-goto-transaction-end)
          (insert "  processed: \"")
          (insert word)
          (insert "\"\n")
          )))))
```

```
(after! beancount
(defun +beancount--navigate-next-transaction ()
  "Move point to beginning of next transaction."
  (interactive)
  ;; make sure we actually move to the next xact, even if we are the beginning
  ;; of one now.
  (if (looking-at +beancount--payee-any-status-regex)
      (forward-line))
  (if (re-search-forward  +beancount--payee-any-status-regex nil t)
      (goto-char (match-beginning 0))
    (goto-char (point-max)))
  (if (not (or (string-equal (beancount-get-transaction-type) "*")
               (end-of-buffer-p)))
      (+beancount--navigate-next-transaction)
)))

(defun beancount-get-transaction-type ()
  "Get the type of the current transaction."
  (interactive)
  (save-excursion
    (beginning-of-line)
    (if (looking-at "^\\([0-9]\\{4\\}\\)-\\([0-9]\\{2\\}\\)-\\([0-9]\\{2\\}\\)
    ↪  \\([a-z\\*\\!]+\\)")
        (match-string 4)
      )))

(after! beancount
(defun +beancount/sort-region (beg end &optional reverse)
  "Sort the transactions inside BEG and END.
If REVERSE (the prefix arg) is non-nil, sort the transactions in reverst
↪  order."
  (interactive
   (list (region-beginning)
         (region-end)
         (and current-prefix-arg t)))
  (let* ((new-beg beg)
         (new-end end)
         (bounds (save-excursion
                   (list (+beancount--navigate-beginning-of-xact)
                         (+beancount--navigate-end-of-xact))))
         (point-delta (- (point) (car bounds)))
         (target-xact (buffer-substring (car bounds) (cadr bounds)))
         (inhibit-modification-hooks t))
    (save-excursion
      (save-restriction
        (goto-char beg)
        ;; make sure beg of region is at the beginning of a line
        (beginning-of-line)
        ;; make sure point is at the beginning of a xact
        (unless (looking-at +beancount--payee-any-status-regex)
          (+beancount--navigate-next-transaction))
        (setq new-beg (point))
        (goto-char end)
        (+beancount--navigate-next-transaction)
        ;; make sure end of region is at the beginning of next record after the
        ;; region
```

```
        (setq new-end (point))
        (narrow-to-region new-beg new-end)
        (goto-char new-beg)
        (let ((inhibit-field-text-motion t))
          (sort-subr
           reverse
           #'+beancount--navigate-next-xact
           #'+beancount--navigate-end-of-xact
           #'+beancount--sort-startkey))))
    (goto-char (point-min))
    (re-search-forward (regexp-quote target-xact))
    (goto-char (+ (match-beginning 0) point-delta))))
)


(provide 'beancount-tweaks)
```

4.5 Dired

4.5.1 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)
```

4.5.2 Dired Tweaks

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

Requirements

```
;(package! casual-dired)
(package! dired-open-with)
```

Code

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

;; Dired

;;; Casual Dired Shortcut Menu
(after! dired
  (define-key dired-mode-map (kbd "C-o") 'casual-dired-tmenu))

;;; 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))

(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\\|part\\)$"
           ↪  ; removed \\|pdf
             "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)))

;; Rotate pdf file
(defun dired-rotate-pdf ()
  "Rotate a pdf file in-place"
  (interactive)
  (cl-map 'nil '(lambda (file)
                  (let ((ext (file-name-extension file))
                        (base-name-sans-ext (file-name-sans-extension
                        ↪  (file-name-nondirectory file))))
                    (when (string-equal "pdf" ext )
                      (message file)
```

```
                    (async-shell-command-no-window
                     (format "mutool draw -R90 -o \"%s\" \"%s\"" file file))
                    )
                )) (dired-get-marked-files)))


;; Vertically split pdf file
(defun dired-split-pdf-vertical ()
  "Vertically split a pdf file in-place"
  (interactive)
  (cl-map 'nil '(lambda (file)
                  (let ((ext (file-name-extension file))
                        (base-name-sans-ext (file-name-sans-extension
                        ↪  (file-name-nondirectory file))))
                    (when (string-equal "pdf" ext )
                      (message file)
                      (async-shell-command-no-window
                       (format "mutool poster -x2 \"%s\" \"%s\"" file file))
                      )
                )) (dired-get-marked-files)))

;; Horizontally split pdf file
(defun dired-split-pdf-horizontal ()
  "Horizontally split a pdf file in-place"
  (interactive)
  (cl-map 'nil '(lambda (file)
                  (let ((ext (file-name-extension file))
                        (base-name-sans-ext (file-name-sans-extension
                        ↪  (file-name-nondirectory file))))
                    (when (string-equal "pdf" ext )
                      (message file)
                      (async-shell-command-no-window
                       (format "mutool poster -y2 \"%s\" \"%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
      :n "r" #'youtube-dl-failures-reset)
```

```
; Allow dragging files from dired to other applications
(setq dired-mouse-drag-files t)

(provide 'dired-tweaks)
```

4.5.3 Tab to Expand Subfiles

```
(load-module 'tab-to-expand-subfiles)
```

Requirements

```
(package! dired-subtree)
```

Code

```
(use-package! dired-subtree
  :defer t
  :after dired
  :config
  (bind-key "<tab>" #'dired-subtree-toggle dired-mode-map)
  (bind-key "<backtab>" #'dired-subtree-cycle dired-mode-map))
(provide 'tab-to-expand-subfiles)
```

4.6 Narrow by regex dired

```
(load-module 'narrow-by-regex-dired)
```

4.6.1 Requirements

```
(package! dired-narrow)
```

4.6.2 Code

```
(provide 'narrow-by-regex-dired)
```

4.7 Ebooks

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

4.7.1 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)
```

4.8 Editing

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

4.8.1 Requirements

```
(package! treesit-auto)
(package! avy)
(package! casual-avy :recipe (:host github :repo "kickingvegas/casual-avy"))
(package! yasnippet-snippets)
```

4.8.2 Code

```
;;; 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))))

; Casual Avy as menu for avy jumps
(setq relative nil) ; workaround for casual-avy bug, needed as long as not in
↪  melpa
(keymap-global-set "M-s" #'casual-avy-tmenu)

(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") 'evil-mc-make-cursor-move-next-line)

(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)

(use-package treesit-auto
  :demand t
  :config
  (setq treesit-auto-install 'prompt)
  (setq treesit-font-lock-level 4) ; all the fontlock details
  (global-treesit-auto-mode))

(map! :map evil-normal-state-map "t t" 'transpose-chars)

(provide 'editing)
```

4.8.3 Indent Guides

```
(load-module 'indent-guide-tweaks)
```

4.8.4 Requirements

```
(package! indent-bars :recipe (:host github :repo "jdtsmith/indent-bars"))
```

4.8.5 Code

```
(use-package! indent-bars
  :config
  (require 'indent-bars-ts)                ; not needed with straight
  :custom
  (indent-bars-treesit-support t)
  (indent-bars-treesit-ignore-blank-lines-types '("module"))
  ;; Add other languages as needed
  (indent-bars-treesit-scope '((python function_definition class_definition
  ↪   for_statement
        if_statement with_statement while_statement)))
  ;; Note: wrap may not be needed if no-descend-list is enough
  ;;(indent-bars-treesit-wrap '((python argument_list parameters ; for python,
  ↪   as an example
  ;;                                    list list_comprehension
```

```
  ;;                                    dictionary dictionary_comprehension
  ;;                                    parenthesized_expression subscript)))
  :hook ((python-base-mode yaml-mode) . indent-bars-mode))

(provide 'indent-guide-tweaks)
```

4.9 Elfeed Tweaks

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

4.9.1 Requirements

4.9.2 Code

```
;;; 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 "DEL" #'elfeed-load-summary
      :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)


  (defun elfeed-eb-garamond ()
    (buffer-face-set '(:family "EB Garamond" :height 120)))

  (add-hook! 'elfeed-show-mode-hook (hide-mode-line-mode 1))
  (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)
    (set-window-point (get-buffer-window "*elfeed-summary*")
    ↪  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
      (set-window-point (get-buffer-window "*elfeed-summary*")
      ↪  elfeed-summary--current-pos)
      (recenter-top-bottom)))
  (buf-move-up)
  )

(defun elfeed-summary-load-update ()
  "Loads the database again before updating"
  (interactive)
```

```
  (elfeed-db-load)
  (message "Refreshing db...")
  (ap/elfeed-search-update-save-filter)
  (elfeed-update)
  )

                                        ; Workaround for slowness while
                                        ↪  updating. See
                                        ;
                                        ↪  https://github.com/skeeto/elfeed/issues/293

(defvar ap/elfeed-update-complete-hook nil
  "Functions called with no arguments when `elfeed-update' is finished.")

(defvar ap/elfeed-updates-in-progress 0
  "Number of feed updates in-progress.")

(defvar ap/elfeed-search-update-filter nil
  "The filter when `elfeed-update' is called.")

(defun ap/elfeed-update-complete-hook (&rest ignore)
  "When update queue is empty, run `ap/elfeed-update-complete-hook' functions."
  (when (= 0 ap/elfeed-updates-in-progress)
    (run-hooks 'ap/elfeed-update-complete-hook)))

(add-hook 'elfeed-update-hooks #'ap/elfeed-update-complete-hook)

(defun ap/elfeed-update-message-completed (&rest _ignore)
  (message "Feeds updated"))

(add-hook 'ap/elfeed-update-complete-hook #'ap/elfeed-update-message-completed)

(defun ap/elfeed-search-update-restore-filter (&rest ignore)
  "Restore filter after feeds update."
  (when ap/elfeed-search-update-filter
    (elfeed-search-set-filter ap/elfeed-search-update-filter)
    (setq ap/elfeed-search-update-filter nil)))

(add-hook 'ap/elfeed-update-complete-hook
↪  #'ap/elfeed-search-update-restore-filter)

(defun ap/elfeed-search-update-save-filter (&rest ignore)
  "Save and change the filter while updating."
  (setq ap/elfeed-search-update-filter elfeed-search-filter)
  (setq elfeed-search-filter "#0"))


(defun ap/elfeed-update-counter-inc (&rest ignore)
  (cl-incf ap/elfeed-updates-in-progress))

(advice-add #'elfeed-update-feed :before #'ap/elfeed-update-counter-inc)

(defun ap/elfeed-update-counter-dec (&rest ignore)
  (cl-decf ap/elfeed-updates-in-progress)
  (when (< ap/elfeed-updates-in-progress 0)
    ;; Just in case
    (setq ap/elfeed-updates-in-progress 0)))
```

```
(add-hook 'elfeed-update-hooks #'ap/elfeed-update-counter-dec)

(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
(when (not (boundp 'youtube-dl-arguments))
  (setq youtube-dl-arguments '())
  )
(add-to-list 'load-path "~/.doom.d/lisp/youtube-dl-emacs")
(load-module 'youtube-dl)

(defvar youtube-dl-4k nil "Whether to download in 4k")
                                       ;(when (string= (getenv "XFT_DPI")
                                       ↪  "192")
                                       ;  (setq youtube-dl-4k t))

(if youtube-dl-4k
    (setq youtube-dl-format-string "bestvideo+bestaudio")
  (setq youtube-dl-format-string
  ↪  "bestvideo[height<=1080]+bestaudio/best[height<=1080]")
  )

(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" ,youtube-dl-format-string
                                    "--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)
  (dired-hide-details-mode))

(defun open-yt-dl-temp-videos ()
  (interactive)
  (find-file youtube-dl-temp-directory)
  (dired-hide-details-mode))


(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-shrink (img-path dimensions callback)
  "Shrink an image to dimensions (string 'WxH') aync, then execute callback"
  (let ((output-path (s-replace ".jpg" ".small.jpg" img-path)))
    (if (file-exists-p output-path)
        (funcall callback output-path)
      (let ((process
             (start-process "image-shrinking-proc" "*img-shrnk-buf*" "convert"
             ↪  img-path "-resize" dimensions output-path)))
        (set-process-sentinel process (lambda (_process _event) (funcall
        ↪  callback output-path)))))))

(defun get-max-minibuffer-height-px ()
  "Return maximum height of echo area and minibuffer in pixels"
  (if (floatp max-mini-window-height) ;height is percentage of frame height if
  ↪  float, else number of lines
      (truncate (* max-mini-window-height (frame-pixel-height)))
    (* max-mini-window-height (frame-char-height))
    ))

(defun image-popup (img-path)
  "Display image at output-path as popup"
  (tooltip-mode 1) ;; tooltip-show doesn't always work
  (image-shrink img-path (format "x%s" (get-max-minibuffer-height-px))
                '(lambda (img-path) (message
                                     (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))))
                       (img-path (concat title ".jpg"))
                       (callback (lambda (&rest _args) (image-popup
                       ↪  img-path))))
                  (if (file-exists-p img-path)
                      (funcall callback)
                    (youtube-dl-get-video-thumbnail it title callback))))
    (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 "/")))

(defun youtube-dl-failures-reset (item)
  "Reset failures."
  (interactive
   (let* ((n (1- (line-number-at-pos))))
     (list (nth n youtube-dl-items))))
  (when item
    (setf (youtube-dl-item-failures item) 0))
  (youtube-dl--redisplay)
  (youtube-dl--run))

(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 (elfeed-tube-mpv-follow-mode 1))
(set-popup-rule! "^\\*elfeed" :side 'left :size (/ 1.0 3))

(provide 'elfeed-tweaks)
```

4.9.3 Async Update Script

```
(setq doom-private-dir "~/.doom.d/")

(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/")

(message "Elfeed-update: %s"
         (add-to-list 'load-path "~/.emacs.d/.local/straight/repos/elfeed/")
         (add-to-list 'load-path
         ↪  "~/.emacs.d/.local/straight/repos/elfeed-summary/")
         (add-to-list 'load-path
         ↪  "~/.emacs.d/.local/straight/repos/elfeed-org/")
         (add-to-list 'load-path "~/.emacs.d/.local/straight/repos/dash.el/")
         (add-to-list 'load-path "~/.emacs.d/.local/straight/repos/s.el/")
         (add-to-list 'load-path
         ↪  "~/.emacs.d/.local/straight/repos/magit/lisp/")
         (add-to-list 'load-path "~/.emacs.d/.local/straight/repos/compat/")
         (require 'elfeed)
         (require 'elfeed-summary)
         (require 'elfeed-org)
         (elfeed-org)
         (elfeed-db-load)
         (elfeed-update)
         (elfeed-summary-update))
```

```
(defun elfeed-summary-load-async-update ()
  "Asynchronously update the database, load it and update summary buffer."
  (interactive)
  (async-start-process "Elfeed Update" "emacs" '(lambda (name)
                                                  (elfeed-db-load)
                                                ;(elfeed-summary-update)
                                                  (message "-> %s completed!"
                                                  ↪  name))
                       "--script" (concat doom-user-dir
                       ↪  "ext/elfeed/update.el")))
```

4.10 Interface

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

4.10.1 Requirements

```
(package! xclip)
```

4.10.2 Code

General settings

```
(setq-default
 x-stretch-cursor t)
(xclip-mode t)
```

```
(good-scroll-mode -1)
(setq-default word-wrap t)
(context-menu-mode 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

(setq display-time-day-and-date t
      display-time-24hr-format t)
      (display-time-mode 0)                             ; Enable time in the
      ↪  mode-line
      (display-battery-mode 0)
```

Font

```
;;; Unicode emojis
(if (>= emacs-major-version 27)
    (progn
      (emojify-mode nil)
    (set-fontset-font t '(#x1f000 . #x1faff)
                      (font-spec :family "Apple Color Emoji"))))
(set-face-attribute
 'default nil :stipple nil :height 100 :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")

(defvar font-garamond nil "Whether EB Garamond font is used")

(defun toggle-font-eb-garamond ()
  "Set font to EB Garamond"
  (interactive)
  (setq font-garamond (not font-garamond))
  (if font-garamond
    (set-face-attribute
 'default nil :stipple nil :height 100 :width 'normal :inverse-video nil :box
 ↪  nil :strike-through nil :overline nil :underline nil :slant 'normal
 ↪  :weight 'normal :foundry "outline" :family "EB Garamond")
(set-face-attribute
 'default nil :stipple nil :height 100 :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")))

(set-fontset-font "fontset-default" 'tagalog (font-spec :family "Noto Sans
↪  Tagalog"))
```

```
;;;; 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")

(set-fontset-font t nil (font-spec :size 16 :name "Symbols Nerd Font"))

(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...
(global-subword-mode 1)                           ; Iterate through CamelCase
↪  words
(setq battery-mode-line-format "%t")

(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))

;; Transparent mode
(defvar transparent-mode t)
(setq transparent-mode--opacity 80)
(exwm-input-set-key (kbd "s-_") '(lambda () (interactive) (--
↪  transparent-mode--opacity) (frame-trans-off)))
(exwm-input-set-key (kbd "s-*") '(lambda () (interactive) (++
↪  transparent-mode--opacity) (frame-trans-off)))

(define-minor-mode transparent-mode
  "Toggle default transparency."
  :lighter " "
  :global t
  (force-mode-line-update))

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

(defun frame-trans-off ()
  (interactive)
  (if transparent-mode
```

```
      (set-frame-parameter (selected-frame) 'alpha-background
      ↪  transparent-mode--opacity)
  (set-frame-parameter (selected-frame) 'alpha-background 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)
```

Sync Theme to Zathura

```
(load-module 'sync-theme-to-zathura)
```

Requirements

Code

```
(setq zathura-set-theme-command "~/.scripts/zathura-recolor.sh")

(defvar enables-sync-theme-to-zathura t)

(defun sync-theme-to-zathura ()
  (interactive)
  (when enables-sync-theme-to-zathura
   (let ((bg (face-attribute 'default :background))
     (fg (face-attribute 'default :foreground))
     (mbg (face-attribute 'mode-line :background))
     (mfg (face-attribute 'mode-line :foreground)))
     (call-process zathura-set-theme-command nil nil nil fg bg fg bg))))

(defun consult-theme--maybe-sync-to-zathura (theme)
  "Disable current themes and enable THEME from `consult-themes'.

The command supports previewing the currently selected theme."
  (interactive
```

```
   (list
    (let* ((regexp (consult--regexp-filter
                    (mapcar (lambda (x) (if (stringp x) x (format "\\`%s\\'"
                    ↪  x)))
                            consult-themes)))
           (avail-themes (seq-filter
                          (lambda (x) (string-match-p regexp (symbol-name x)))
                          (cons 'default (custom-available-themes))))
           (saved-theme (car custom-enabled-themes)))
      (consult--read
       (mapcar #'symbol-name avail-themes)
       :prompt "Theme: "
       :require-match t
       :category 'theme
       :history 'consult--theme-history
       :lookup (lambda (selected &rest _)
                 (setq selected (and selected (intern-soft selected)))
                 (or (and selected (car (memq selected avail-themes)))
                     saved-theme))
       :state (lambda (action theme)
                (pcase action
                  ('return (consult-theme (or theme saved-theme)))
                  ((and 'preview (guard theme)) (consult-theme theme))))
       :default (symbol-name (or saved-theme 'default))))))
  (when (eq theme 'default) (setq theme nil))
  (unless (eq theme (car custom-enabled-themes))
    (mapc #'disable-theme custom-enabled-themes)
    (when theme
      (if (custom-theme-p theme)
          (progn (enable-theme theme) (sync-theme-to-zathura))
        (progn (load-theme theme :no-confirm)
          (sync-theme-to-zathura))))))

(add-function :override (symbol-function 'consult-theme)
↪  #'consult-theme--maybe-sync-to-zathura)

(provide 'sync-theme-to-zathura)
```

4.11 EXWM Tweaks

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

4.11.1 Requirements

4.11.2 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 is deprecated
  ; no need for (exwm-config-example), has unwanted defaults and uses ido
```

```
  (advice-add 'exwm-init-remove-error :around #'exwm-init)

  (defun exwm-init-remove-error (orig-function)
   (condition-case nil
      (funcall original-function)
    (error (message "EXWM retires: Other DE/WM running"))))

  (fringe-mode -1)
  (require 'exwm-randr)
  ;(setq exwm-randr-workspace-monitor-plist '(0 "eDP-1" 1 "DP-2-1" 2 "DP-2-2" 3
  ↪  "HDMI-2" 4 "HDMI-1"))

  (defun exwm-update-monitors ()
    "Detects connected monitors and sets exwm-randr-workspace-monitor-plist
    ↪  accordingly"
    (interactive)
    (setq exwm-randr-workspace-monitor-plist
          (let ((monitors (mapcar (lambda (monitor) (alist-get 'name monitor))
          ↪  (display-monitor-attributes-list))))
            (apply #'append (cl-mapcar #'list (number-sequence 0 (1- (length
            ↪  monitors))) monitors)))))

  (exwm-update-monitors)
  ;(exwm-randr-enable)
  ; TODO FIX THIS

;; (when (string= (system-name) "astaroth")
;;    (setq exwm-randr-workspace-monitor-plist '(1 "eDP-1" 2 "DP-2-1" 3
↪  "HDMI-2" 3 "DP-2-2")))
;;  (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 'exwm-update-monitors)

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

 ; (add-hook 'exwm-randr-screen-change-hook
 ;     (lambda ()
 ;       (start-process-shell-command
 ;        "xrandr" nil "xrandr --output eDP-1 --primary --mode 1920x1080 --pos
↪  0x0 ;--rotate normal --output DP-1 --off --output HDMI-1 --off --output
↪  DP-2 --off ;--output HDMI-2 --off --output DP-2-1 --mode 1920x1080 --pos
↪  1920x0 --rotate ;normal --output DP-2-2 --off --output DP-2-3 --off")))

  ;(setq exwm-randr-screen-change-hook nil)

  (winner-mode t)
  (require 'exwm-systemtray)
  ;(exwm-systemtray-enable) Systray handled by polybar
```

```
  ; Normal copy-pasting
  (define-key exwm-mode-map (kbd "C-c") nil)
  (define-key exwm-mode-map (kbd "C-v") 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-v] . [next])
          ([?\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])))

  (add-hook 'exwm-manage-finish-hook (lambda () (when (eq major-mode
  ↪  'exwm-mode) (evil-local-mode -1))))

  (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)
      )))

(defun exwm-move-window-to-workspace(workspace-number)
  (interactive)
  (let ((frame (exwm-workspace--workspace-from-frame-or-index
  ↪  workspace-number))
        (id (exwm--buffer->id (window-buffer))))
    (exwm-workspace-move-window frame id)))
```

```
  (setq exwm-workspace-number 1
        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-r] . exwm-reset)
          ([?\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 1))) .
                      (lambda ()
                        (interactive)
                        (exwm-workspace-switch-create ,i))))
                  (number-sequence 0 8))
        ,@(mapcar (lambda (i)
                    `(,(kbd (format "M-s-%d" (+ i 1))) .
                      (lambda ()
                        (interactive)
                        (exwm-move-window-to-workspace ,i))))
                  (number-sequence 0 8))

        ))


  (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")
            (string-equal exwm-class-name "SshAskpass")
            (string-equal exwm-instance-name "Toplevel")
            (and (string-equal exwm-class-name "Thunar")
                 (or (string-equal exwm-title "File Operation Progress")
                     (string-equal exwm-title "Attention")
                     ))
            (and (string-equal exwm-class-name "Firefox")
                 (string-equal exwm-title "Save Image"))
            (and (string-equal exwm-class-name "firefox")
                 (string-equal exwm-title "Picture-in-Picture"))
            (and (string-equal exwm-class-name "KeePassXC")
                 (string-equal exwm-title "KeePassXC -  Access Request"))
            (not (progn
                   ;(message (format "Class: %s" exwm-class-name))
                   %(message (format "Title: %s" exwm-instance-name))
                   ))
            (string-equal exwm-class-name "ripdrag")
            (equal exwm-window-type `(,xcb:Atom:_NET_WM_WINDOW_TYPE_DIALOG))
            (equal exwm-window-type xcb:Atom:_NET_WM_WINDOW_TYPE_DIALOG)
            (member xcb:Atom:_NET_WM_WINDOW_TYPE_DIALOG exwm-window-type)
            )
           floating t
           floating-mode-line nil
;           width 0.4
;           height 0.4
           )
       ((or (string-equal exwm-class-name "kdeconnect.daemon"))
        floating t
        fullscreen t
        width 0
        height 0)
        ))

;; (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-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 &&
  ↪  /home/user/dox/install/rosarium/target/release/rosarium \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 t)
↪  (mu4e--goto-inbox)))

(global-unset-key (kbd "S-s-1"))
(keymap-global-set "S-s-1" #'(lambda () (interactive)
↪  ((exwm-move-window-to-workspace 1))))
(exwm-input-set-key (kbd "S-s-1") (lambda () (interactive)
↪  ((exwm-move-window-to-workspace 1))))

(exwm-input--update-global-prefix-keys)

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

(exwm-init)
(provide 'exwm-tweaks)
```

4.12 EXWM Buffer movements

```
(load-module 'exwm-buffer-movements)
```

4.12.1 Requirements

```
(package! buffer-move)
```

4.12.2 Code

```
(exwm-input-set-key (kbd "s-h") 'buf-move-left)
(exwm-input-set-key (kbd "s-l") 'buf-move-right)
(exwm-input-set-key (kbd "s-k") 'buf-move-up)
(exwm-input-set-key (kbd "s-j") 'buf-move-down)

; Resize
(exwm-input-set-key (kbd "s-<kp-8>") 'exwm-layout-enlarge-window)
(exwm-input-set-key (kbd "s-<kp-2>") 'exwm-layout-shrink-window)
(exwm-input-set-key (kbd "s-<kp-6>") 'exwm-layout-enlarge-window-horizontally)
(exwm-input-set-key (kbd "s-<kp-2>") 'exwm-layout-shrink-window-horizontally)
(exwm-input-set-key (kbd "s-<kp-5>") 'exwm-layout-toggle-mode-line)
(exwm-input--update-global-prefix-keys)
(provide 'exwm-buffer-movements)
```

4.12.3 Exwm Gaps

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

Requirements

```
(package! exwm-outer-gaps
  :recipe
```

```
  (:host github
   :repo "lucasgruss/exwm-outer-gaps"))
```

Code

```
(use-package! exwm-outer-gaps
  :after (exwm xelb)
  :config
  (exwm-outer-gaps-mode +1)
  (exwm-input-set-key (kbd "s-+") 'exwm-outer-gaps-increment)
  (exwm-input-set-key (kbd "s--") 'exwm-outer-gaps-decrement))
(provide 'exwm-gaps)
```

4.13 General

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

4.13.1 Requirements

4.13.2 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)
```

4.14 Navigation

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

4.14.1 Requirements

```
(package! switch-window)
```

4.14.2 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)

;;; Ace window
(use-package! ace-window
  :config
  (setq ace-window-display-mode nil
        aw-keys '(106 104 103 102 100 115 97 107 108)
        aw-scope 'global)
 ; ace window does not work well with exwm, it draws keys underneath the window
 ; :bind
 ; ([remap other-window] . ace-window)
  )

;;; Switch window
```

```
(use-package switch-window
  :ensure t
  :defer t
  :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-this-buffer)
(global-set-key (kbd "C-x 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)

;;; Cycle through workspaces
(exwm-input-set-key (kbd "C-TAB") '+workspace/cycle)
(global-set-key (kbd "C-TAB") '+workspace/cycle)
(exwm-input--update-global-prefix-keys)

;;; Winum mode for easy moving through windows
         (bind-key "s-=" '+workspace/switch-to-9)
         (bind-key "s-!" '+workspace/switch-to-0)
         (bind-key "s-\"" '+workspace/switch-to-1)
         (bind-key "s-§" '+workspace/switch-to-2)
         (bind-key "s-$" '+workspace/switch-to-3)
         (bind-key "s-%" '+workspace/switch-to-4)
         (bind-key "s-&" '+workspace/switch-to-5)
         (bind-key "s-/" '+workspace/switch-to-6)
         (bind-key "s-(" '+workspace/switch-to-7)
         (bind-key "s-)" '+workspace/switch-to-8)


(provide 'navigation)
```

4.15 Shortcuts

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

4.15.1 Requirements

4.15.2 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 -p2222 sermak.xyz:/ /mnt/server-de"))
  (let ((dir (f-join uni-base-folder folder)))
    (when (not (file-exists-p dir))
      (mkdir dir t))
    (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 "")))
;;; Devanagari anusvara
(global-set-key (kbd "C-c 5") (lambda () (interactive) (insert "")))
(global-set-key (kbd "C-c 6") (lambda () (interactive) (insert "")))

;;; Rectangle mark mode
(global-set-key (kbd "C-ü") (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-S-<return>") (lambda () (interactive) (+vterm/toggle
↪  nil)))

                                        ; Org agenda

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

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

(defun agenda-show-daily-habits ()
  "Open daily habits document as org column"
  (interactive)
  (agenda-daily)
  (org-columns)
  (org-modern-mode -1))

(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)
;(global-set-key (kbd "s-t") 'agenda-show-daily-habits)

                                        ; Books

(defun books ()
  (interactive)
  (find-file "/home/user/dox/books/"))

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

(provide 'shortcuts)
```

4.16 Config visit

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

4.16.1 Requirements

4.16.2 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)
```

4.17 Legacy ivy functions, still needed for vertico icons

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

4.17.1 Requirements

4.17.2 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

(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)
    ))

(use-package! counsel
  :bind
  (("M-y" . counsel-yank-pop)
   :map ivy-minibuffer-map
   ("M-y" . ivy-next-line)))

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

(after! consult
  (cl-defmethod nerd-icons-completion-get-icon (cand (_cat (eql buffer)))
    "Return the icon for the candidate CAND of completion category buffer."
    (let* ((mode (buffer-local-value 'major-mode (get-buffer cand)))
           (buffname (buffer-name (get-buffer cand)))
           (icon (nerd-icons-icon-for-mode mode))
           (parent-icon (nerd-icons-icon-for-mode
```

```
                        (get mode 'derived-mode-parent))))
      (concat
       (or
        (nerd-icons--icon-for-firefox mode buffname)
        (all-the-icons-ivy--icon-for-tor mode buffname)
        (all-the-icons-ivy--icon-for-exwm mode buffname)
        (nerd-icons--icon-for-emacs mode buffname)
        icon
        parent-icon
        (nerd-icons-faicon "nf-fa-sticky_note_o"))
       " "))
    ))

(defun nerd-icons--icon-for-emacs (mode buffname)
  (let ((m (format "%s" mode))
        (b (format "%s" buffname)))
    (when (and (string-equal m "messages-buffer-mode")
               (string-equal b "*Messages*"))
      (propertize (nerd-icons-mdicon "nf-md-message_text_outline") 'face
      ↪  '(:foreground "dim gray")))

       ))

;; Overwrite some stuff for exwm and icons in Firefox
(defun nerd-icons--icon-for-firefox (mode buffname)
  "Get icon for 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 " - "))
             (browser (format "%s" (last fnl)))
             (youtube (format "%s" (last bnl)))
             )
        (if (or (string-equal youtube "(YouTube - Mozilla Firefox)")
                (string-equal youtube "(YouTube - Mozilla Firefox (Private
                ↪  Browsing))")
                (string-equal youtube "(YouTube - Mozilla Firefox Private
                ↪  Browsing)")
                (string-equal youtube "YouTube - Mozilla Firefox Private
                ↪  Browsing")
                (string-equal youtube "YouTube - Mozilla Firefox Private
                ↪  Browsing")
                )
            (propertize (nerd-icons-faicon "nf-fa-youtube") 'face '(:foreground
            ↪  "red" :background "white"))
          (if (or
               (string-equal browser "(Mozilla Firefox)")
               (string-equal browser "(Mozilla Firefox (Private Browsing)")
               (string-equal browser "Mozilla Firefox Private Browsing")
               (string-equal browser "(Mozilla Firefox Private Browsing")
               (string-equal browser "(Mozilla Firefox Private Browsing)")
               )
              (propertize (nerd-icons-faicon "nf-fa-firefox") 'face
              ↪  '(:foreground "orange 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)")
                (nerd-icons-icon-for-url "youtube.com")
              (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)
             (propertize (nerd-icons-faicon "nf-fa-comment") 'face
             ↪  '(:foreground "deep sky blue")))
            ((string-prefix-p "Skype" buffname)
             (propertize (nerd-icons-faicon "nf-fa-skype") 'face '(:foreground
             ↪  "light blue")))
            ((string-suffix-p " - Discord" buffname)
             (propertize (nerd-icons-faicon "nf-fa-discord") 'face
             ↪  '(:foreground "purple")))
            ((string-prefix-p "OBS" buffname)
             (propertize (nerd-icons-faicon "nf-fa-video_camera") 'face
             ↪  '(:foreground "dark gray")))
            ((string-prefix-p "QEMU" buffname)
             (propertize (nerd-icons-faicon "nf-fa-desktop") 'face
             ↪  '(:foreground "dark gray")))
            ((string-prefix-p "Bluetooth Devices" buffname)
             (propertize (nerd-icons-faicon "nf-fa-bluetooth") 'face
             ↪  '(:foreground "deep sky blue")))
            ((file-directory-p buffname)
             (propertize (nerd-icons-faicon "nf-fa-folder_open") 'face
             ↪  '(:foreground "yellow")))
            ((string-suffix-p " - mpv" buffname)
             (propertize (nerd-icons-faicon "nf-fa-play") 'face '(:foreground
             ↪  "orange")))
            ((string-suffix-p " - Mozilla Thunderbird" buffname)
             (propertize (nerd-icons-mdicon "nf-md-email") 'face '(:foreground
             ↪  "sky blue")))
            ((string-match-p " - Thunar" buffname)
             (propertize (nerd-icons-faicon "nf-fa-folder_open") 'face
             ↪  '(:foreground "yellow")))
            ((string-suffix-p "\.pdf" buffname)
             (propertize (nerd-icons-faicon "nf-fa-file_pdf_o") 'face
             ↪  '(:foreground "red")))
            ((or (string-equal "st" buffname)
                 (string-prefix-p (concat (user-login-name) "@") buffname)
                 (string-prefix-p "root@" buffname)
                 (string-match "st<[0-9]+>" buffname))
             (propertize (nerd-icons-octicon "nf-oct-terminal") 'face
             ↪  '(:foreground "green")))
            )))
```

```
(provide 'search-ivy)
```

4.18 Search with vertico

```
(after! vertico
   (load-module 'search-with-vertico)
)
```

4.18.1 Requirements

```
(package! swiper)
(package! vertico-posframe)
;; (package! vertico)
;; (package! vertico :recipe (:files (:defaults "extensions/*") ; Special
↪  recipe to load extensions conveniently
;;                                :includes (vertico-indexed
;;                                           vertico-flat
;;                                           vertico-grid
;;                                           vertico-mouse
;;                                           vertico-quick
;;                                           vertico-buffer
;;                                           vertico-repeat
;;                                           vertico-reverse
;;                                           vertico-directory
;;                                           vertico-multiform
;;                                           vertico-unobtrusive
;;                                           )))
```

4.18.2 Code

```
(use-package! marginalia
  :general
  (:keymaps 'minibuffer-local-map
            "M-A" 'marginalia-cycle)
  :custom
  (marginalia-max-relative-age 0)
  (marginalia-align 'right)
  :init
  (marginalia-mode))


(global-set-key (kbd "C-ö") 'embark-act)

(require 'vertico)
(defun consult-buffer-no-auxiliary ()
  "Consult buffer without buffers starting with *"
  (interactive)
  ; consult-buffer-filter might not be loaded at startup yet
  (if (boundp 'consult-buffer-filter)
      (let ((consult-buffer-filter    (append (mapcar 'file-name-nondirectory
      ↪  org-agenda-files) (cons "\\*"       consult-buffer-filter))))
        (consult-buffer))
```

```
    (consult-buffer)))

(use-package! vertico
  :demand t                             ; Otherwise won't get loaded
  ↪  immediately
  :general
  (:keymaps '(normal insert visual motion)
            "M-." #'vertico-repeat
            )
  (:keymaps 'vertico-map
            "<tab>" #'vertico-insert ; Set manually otherwise setting
            ↪  `vertico-quick-insert' overrides this
            "<escape>" #'minibuffer-keyboard-quit
            "?" #'minibuffer-completion-help
            "C-M-n" #'vertico-next-group
            "C-M-p" #'vertico-previous-group
            ;; Multiform toggles
            "<backspace>" #'vertico-directory-delete-char
            "C-w" #'vertico-directory-delete-word
            "C-<backspace>" #'vertico-directory-delete-word
            "RET" #'vertico-directory-enter
            "C-i" #'vertico-quick-insert
            "C-o" #'vertico-quick-exit
            "M-o" #'kb/vertico-quick-embark
            "M-G" #'vertico-multiform-grid
            "M-F" #'vertico-multiform-flat
            "M-R" #'vertico-multiform-reverse
            "M-U" #'vertico-multiform-unobtrusive
            "C-l" #'kb/vertico-multiform-indexed-toggle
            "C-ö" #'embark-act
            )
  :bind (
         ("C-s" . consult-line)
         ("C-S" . consult-line-multi)
         ("C-x C-b" . consult-buffer-no-auxiliary))
  :hook ((rfn-eshadow-update-overlay . vertico-directory-tidy) ; Clean up file
  ↪  path when typing
         (minibuffer-setup . vertico-repeat-save) ; Make sure vertico state is
         ↪  saved
         )
  :init
  ;(setq vertico-count 13)
  (setq vertico-resize t)
  (setq vertico-cycle nil)
  ;; Extensions
  (setq vertico-grid-separator "       ")
  (setq vertico-grid-lookahead 50)
  (setq vertico-buffer-display-action '(display-buffer-reuse-window))
  ;(setq vertico-multiform-categories
  ; '((file reverse)
  ;   (consult-grep buffer)
  ;   (consult-location)
  ;   (imenu buffer)
  ;   (library reverse indexed)
  ;   (org-roam-node reverse indexed)
  ;   (t reverse)
  ;))
```

```
  ;(setq vertico-multiform-commands
  ; '(("flyspell-correct-*" grid reverse)
  ;   (org-refile grid reverse indexed)
  ;   (consult-yank-pop indexed)
  ;   (consult-flycheck)
  ;   (consult-lsp-diagnostics)
  ;   ))
  (defun kb/vertico-multiform-indexed-toggle ()
    "Toggle between indexed and reverse."
    (interactive)
    (vertico-multiform--display-toggle 'vertico-indexed-mode)
    (if vertico-indexed-mode
        (vertico-multiform--temporary-mode 'vertico-reverse-mode -1)
      (vertico-multiform--temporary-mode 'vertico-reverse-mode 1)))
  (defun kb/vertico-quick-embark (&optional arg)
    "Embark on candidate using quick keys."
    (interactive)
    (when (vertico-quick-jump)
      (embark-act arg)))

  ;; Workaround for problem with `tramp' hostname completions. This overrides
  ;; the completion style specifically for remote files! See
  ;; https://github.com/minad/vertico#tramp-hostname-completion
  (defun kb/basic-remote-try-completion (string table pred point)
    (and (vertico--remote-p string)
         (completion-basic-try-completion string table pred point)))
  (defun kb/basic-remote-all-completions (string table pred point)
    (and (vertico--remote-p string)
         (completion-basic-all-completions string table pred point)))
  (add-to-list 'completion-styles-alist
               '(basic-remote           ; Name of `completion-style'
                 kb/basic-remote-try-completion kb/basic-remote-all-completions
                 ↪  nil))
  :config
  (vertico-mode)
  ;; Extensions
  (vertico-multiform-mode nil)
  ;; Needed to move away from ido
  (setq completing-read-function 'completing-read-default)

  ;; Prefix the current candidate with "» ". From
  ;; https://github.com/minad/vertico/wiki#prefix-current-candidate-with-arrow
  (advice-add #'vertico--format-candidate :around
              (lambda (orig cand prefix suffix index _start)
                (setq cand (funcall orig cand prefix suffix index _start))
                (concat
                 (if (= vertico--index index)
                     (propertize "» " 'face 'vertico-current)
                   "  ")
                 cand)))
  )

(use-package orderless
  :custom
  (completion-styles '(orderless))
  (completion-category-defaults nil)    ; I want to be in control!
  (completion-category-overrides
```

```
   '((file (styles basic-remote ; For `tramp' hostname completion with
   ↪  `vertico'
                  orderless
                  ))
     ))

  (orderless-component-separator 'orderless-escapable-split-on-space)
  (orderless-matching-styles
   '(orderless-literal
     orderless-prefixes
     ;;orderless-initialism
     orderless-regexp
     ;;orderless-flex ;; Fuzzy finding
     ;; orderless-strict-leading-initialism
     ;; orderless-strict-initialism
     ;; orderless-strict-full-initialism
     ;; orderless-without-literal          ; Recommended for dispatches instead
     ))
  (orderless-style-dispatchers
   '(prot-orderless-literal-dispatcher
     prot-orderless-strict-initialism-dispatcher
     prot-orderless-flex-dispatcher
     ))
  :init
  (defun orderless--strict-*-initialism (component &optional anchored)
    "Match a COMPONENT as a strict initialism, optionally ANCHORED.
The characters in COMPONENT must occur in the candidate in that
order at the beginning of subsequent words comprised of letters.
Only non-letters can be in between the words that start with the
initials.

If ANCHORED is `start' require that the first initial appear in
the first word of the candidate.  If ANCHORED is `both' require
that the first and last initials appear in the first and last
words of the candidate, respectively."
    (orderless--separated-by
        '(seq (zero-or-more alpha) word-end (zero-or-more (not alpha)))
      (cl-loop for char across component collect `(seq word-start ,char))
      (when anchored '(seq (group buffer-start) (zero-or-more (not alpha))))
      (when (eq anchored 'both)
        '(seq (zero-or-more alpha) word-end (zero-or-more (not alpha)) eol))))

  (defun orderless-strict-initialism (component)
    "Match a COMPONENT as a strict initialism.
This means the characters in COMPONENT must occur in the
candidate in that order at the beginning of subsequent words
comprised of letters.  Only non-letters can be in between the
words that start with the initials."
    (orderless--strict-*-initialism component))

  (defun prot-orderless-literal-dispatcher (pattern _index _total)
    "Literal style dispatcher using the equals sign as a suffix.
It matches PATTERN _INDEX and _TOTAL according to how Orderless
parses its input."
    (when (string-suffix-p "=" pattern)
      `(orderless-literal . ,(substring pattern 0 -1))))
```

```
  (defun prot-orderless-strict-initialism-dispatcher (pattern _index _total)
    "Leading initialism  dispatcher using the comma suffix.
It matches PATTERN _INDEX and _TOTAL according to how Orderless
parses its input."
    (when (string-suffix-p "," pattern)
      `(orderless-strict-initialism . ,(substring pattern 0 -1))))

  (defun prot-orderless-flex-dispatcher (pattern _index _total)
    "Flex  dispatcher using the tilde suffix.
It matches PATTERN _INDEX and _TOTAL according to how Orderless
parses its input."
    (when (string-suffix-p "." pattern)
      `(orderless-flex . ,(substring pattern 0 -1))))
  )

(global-set-key (kbd "C-s") 'consult-line)

; Global vertico posframe mode
(vertico-posframe-mode 1)
(provide 'search-with-vertico)
```

4.19 Reading

4.19.1 Read Aloud

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

Requirements

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)
```

4.19.2 Speed Read

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

Requirements

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)
```

4.19.3 Bionic Reading

```
(load-module 'bionic-reading)
```

Requirements

Code

```
;;; bionic-reading.el --- Bionic reading            -*- lexical-binding: t;
↪  -*-

;; Copyright (C) 2022  ksqsf

;; Author: ksqsf <i@ksqsf.moe>
;; Keywords: convenience

;; This program 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 of the License, 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.

;; You should have received a copy of the GNU General Public License
;; along with this program.  If not, see <https://www.gnu.org/licenses/>.

;;; Commentary:

;; This package implements something similar to bionic reading in Emacs.
;;
;; Note that bionic reading is patented.

;;; Code:

(defvar-local bionic-overlays nil
```

```
  "The overlays for bionicification in the current buffer.")

;;;###autoload
(defun bionic-word ()
  "Bionicify the word at point"
  (interactive)
  (let* ((bounds (bounds-of-thing-at-point 'word))
         (beg (car bounds))
         (end (cdr bounds))
         (whole-len (- end beg)))
    (cond
     ((>= whole-len 2)
      (let* ((half-len (/ whole-len 2))
             (real-len (if (or (> whole-len 6) (= whole-len 3))
                           (+ half-len 1)
                         half-len))
             (ov (make-overlay beg (+ beg real-len))))
        (overlay-put ov 'face bionic-reading-face)
        (push ov bionic-overlays)))
     ((> (- end beg) 1)
      (let ((ov (make-overlay beg (+ beg 1))))
        (overlay-put ov 'face bionic-reading-face)
        (push ov bionic-overlays)))
     (t nil))))

;;;###autoload
(defun bionic-buffer ()
  "Bionicify all the visible parts of the current buffer."
  (interactive)
  (if (not (null bionic-overlays))
      (bionic-debuffer))
  (save-excursion
    (goto-char (point-min))
    (while (not (= (point) (point-max)))
      (if (looking-at "\\w")
          (bionic-word))
      (forward-to-word 1))))

;;;###autoload
(defun bionic-debuffer ()
  "Undo the bionicification."
  (interactive)
  (dolist (ov bionic-overlays)
    (delete-overlay ov)))

;;; Start of my own code
;; Bionic reading minor mode
(defvar bionic-reading-mode nil)
(defcustom bionic-reading-face 'bold "Face to use for bionic reading highlight.
Option: 'bold, 'error, 'warning, 'highlight or any value of M-x
↪  list-faces-display")

(define-minor-mode bionic-reading-mode
  "Makes first half of words bold to facilitate faster reading."
  :lighter " "
  :global t
  (if bionic-reading-mode
```

```
      (bionic-buffer)
    (bionic-debuffer))
  (force-mode-line-update))

(provide 'bionic-reading)
;;; bionic-reading.el ends here
```

4.20 Spaced Repetition

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

4.20.1 Requirements

4.20.2 Code

```
;;; 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 'lispy)
(require 'hydra)
(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-raise-minor-mode 'pamparam-card-mode)

(provide 'pamparam)

;;; pamparam.el ends here
```

4.20.3 Tweaks

```
(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")
  )

(after! recentf
  (add-to-list 'recentf-exclude "pamparam.*/cards")
  (add-to-list 'recentf-exclude "/pamparam/"))

;;* `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)
```

4.21 Popes

```
(when home?
  (load-module 'popes))
```

4.21.1 Requirements

4.21.2 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)
```

4.22 Keycast Tweaks

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

4.22.1 Requirements

4.22.2 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)
```

4.23 Weather

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

4.23.1 Requirements

4.23.2 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))
(provide 'weather)
```

4.24 Org

4.24.1 Org Protocol

System

Linux Set up emacsclient as org-protocol scheme-handler

```
(mkdir "~/.local/share/applications/" t)
```

```
[Desktop Entry]
Name=org-protocol
Comment=Intercept calls from emacsclient to trigger custom actions
Categories=Other;
Keywords=org-protocol;
Icon=emacs
Type=Application
Exec=emacsclient -- %u
Terminal=false
StartupWMClass=Emacs
MimeType=x-scheme-handler/org-protocol;
```

Update MIME cache

```
update-desktop-database ~/.local/share/applications/
```

Windows

```
(mkdir (concat doom-private-dir "ext/org-protocol/") t)
```

```
REGEDIT4

[HKEY_CLASSES_ROOT\org-protocol]
@="URL:Org Protocol"
"URL Protocol"=""
[HKEY_CLASSES_ROOT\org-protocol\shell]
[HKEY_CLASSES_ROOT\org-protocol\shell\open]
[HKEY_CLASSES_ROOT\org-protocol\shell\open\command]
```

```
@="\"C:\\Windows\\System32\\wsl.exe\" emacsclient \"%1\""
```

JS Bookmarks Replace `capture` subprotocol with `store-link` or `youtube-dl`

```
javascript:location.href='org-protocol://capture?' +
      new URLSearchParams({
            template: 'x', url: window.location.href,
            title: document.title, body: window.getSelection()});
```

youtube-dl handler

```
(require 'org-protocol)
(add-to-list 'org-protocol-protocol-alist
             '("Download like with youtube-dl"
               :protocol "youtube-dl"
               :function youtube-dl-protocol-handler))

(defun youtube-dl-protocol-handler (data)
  "Add url to youtube-dl download queue."
  (let ((url (plist-get data :url))
        (title (plist-get data :title)))
    (unless (string= title "about:blank")
      (youtube-dl
       (plist-get data :url)
       :title (plist-get data :title))))
  nil)
```

4.24.2 Org-Roam

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

Requirements

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

Code

```
(use-package! websocket
    :after org-roam)

(use-package! org-roam-ui
    :after org-roam ;; or :after org
;;         normally we'd recommend hooking orui after org-roam, but since
↪  org-roam does not have
;;         a hookable mode anymore, you're advised to pick something yourself
;;         if you don't care about startup time, use
;;  :hook (after-init . org-roam-ui-mode)
```

```
    :config
    (setq org-roam-ui-sync-theme t
          org-roam-ui-follow t
          org-roam-ui-update-on-save t
          org-roam-ui-open-on-start t))

(setq org-roam-directory "~/sync/org/roam")
(mkdir org-roam-directory t)
(org-roam-db-autosync-mode 1)

;; Hide the mode line in the org-roam buffer, since it serves no purpose. This
;; makes it easier to distinguish among other org buffers.
(add-hook 'org-roam-buffer-prepare-hook #'hide-mode-line-mode)

;; Since the org module lazy loads org-protocol (waits until an org URL is
;; detected), we can safely chain `org-roam-protocol' to it.
(use-package org-roam-protocol
  :after org-protocol)


(use-package org-roam-bibtex
  :after (org-roam)
  :hook (org-roam-mode . org-roam-bibtex-mode)
  :config
  (setq org-roam-bibtex-preformat-keywords
        '("=key=" "title" "url" "file" "author-or-editor" "keywords"))
  (setq orb-templates
        '(("r" "ref" plain (function org-roam-capture--get-point)
           ""
           :file-name "${slug}"
           :head "#+TITLE: ${=key=}: ${title}\n#+ROAM_KEY: ${ref}

- tags ::
- keywords :: ${keywords}

\n* ${title}\n  :PROPERTIES:\n  :Custom_ID: ${=key=}\n  :URL: ${url}\n
↪  :AUTHOR: ${author-or-editor}\n  :NOTER_DOCUMENT: %(orb-process-file-field
↪  \"${=key=}\")\n  :NOTER_PAGE: \n  :END:\n\n"

           :unnarrowed t))))

;; Actually start using templates
;(after! org-capture
  ;; For browser capture
  (add-to-list 'org-capture-templates
               '("P" "Protocol" entry ; key, name, type
                 (file+headline +org-capture-notes-file "Inbox") ; target
                 "* %^{Title}\nSource: %u, %c\n
                 ↪  #+BEGIN_QUOTE\n%i\n#+END_QUOTE\n\n\n%?"
                 :prepend t ; properties
                 :kill-buffer t))
  (add-to-list 'org-capture-templates
               '("L" "Protocol Link" entry
                 (file+headline +org-capture-notes-file "Inbox")
                 "* %? [[%:link][%(transform-square-brackets-to-round-ones
                 ↪  \"%:description\")]]\n"
                 :prepend t
```

```
                :kill-buffer t))

;  )
(map! :map evil-org-mode-map
        :leader
        (:prefix ("R")
         :desc "Insert node"
         "i" #'org-roam-node-insert
         :desc "Find node"
         "f" #'org-roam-node-find
         :desc "Capture to node"
         "c" #'org-roam-capture
         :desc "Toggle roam buffer"
         "b" #'org-roam-buffer-toggle
         :desc "Open random note"
         "r" #'org-roam-node-random
         :desc "Visit node"
         "v" #'org-roam-node-visit
         :desc "Open ORUI"
         "u" #'org-roam-ui-open))

(use-package! org-roam-bibtex
  :after org-roam)

(defun org-noter-roam-init (ref)
  "Initialize org-roam notes file for org-noter use.
  Insert citation, create notes headline, add org-noter document property"
  (interactive (list (citar-select-ref)))
  (end-of-buffer)
  (let*
      ((files
        (citar-get-files
         (list ref)))
       (file
        (car
         (gethash ref files))))
    (citar-insert-citation
     (list ref))
    (newline)
    (insert "* ")
    (insert (citar-get-value "title" ref))
    (newline)
    (org-roam-ref-add
     (concat "@" ref))
    (org-set-property "NOTER_DOCUMENT" file)
    (org-set-property "NOTER_PAGE" "1")
    ))

(provide 'org-roam-tweaks)
```

4.24.3 Org Tweaks

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

Requirements

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

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))

  (mixed-pitch-mode 1)

  ;; 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-remote-file "/media/user/keychain/finances/wallet.beancount"
      beancount-local-file "~/dox/finances/wallet.beancount")

(defun beancount-open-local ()
  "Open local beancount wallet."
  (interactive)
  (find-file beancount-local-file))

;; Capture
(setq org-default-notes-file "~/dox/notes/notes.org")
(add-to-list 'org-capture-templates
             '("b" "Beancount Entry" plain
               (file beancount-local-file)
               "bc%?" :empty-lines-before 1))

(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)
```

4.24.4 Org Functions

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

Requirements

Copy subtree without heading or children

```
(defun org-copy-subtree-only ()
  "Copy the current subtree excluding heading and children into clipboard."
  (interactive)
  (if (org-before-first-heading-p)
      (message "Not in or on an org heading")
    (save-excursion
      ;; If inside heading contents, move the point back to the heading
      ;; otherwise `org-agenda-get-some-entry-text' won't work.
      (unless (org-on-heading-p) (org-previous-visible-heading 1))
      (let ((contents (substring-no-properties
                       (org-agenda-get-some-entry-text
                        (point-marker)
                        most-positive-fixnum))))
        (message "Copied %d chars" (length contents))
        (kill-new contents)))))
```

Adds filename prefixes to counsel-org-goto-all

```
(defun add-filename-to-counsel-outline-candidates (candidates)
  "Add the filename at the beginning for CANDIDATES from
  ↪  `counsel-outline-candidates'."
  (mapcar
   (lambda (candidate)
     (let* ((marker (cdr candidate))
            (filename (buffer-file-name (marker-buffer marker)))
            (filename-abbreviated (when filename (concat (abbreviate-file-name
            ↪  filename) " ")))
            ;; Use this if you want the buffer name. It's a bit shorter.
            ;; (buffername (buffer-name (marker-buffer (cdr candidate))))
            )
       (cons (concat filename-abbreviated (car candidate)) marker)))
   candidates))
```

```
(advice-add 'counsel-outline-candidates :filter-return
↪  #'add-filename-to-counsel-outline-candidates)
```

Add text to any org headline

```
(require 'counsel)
(require 's)

(defun org-get-headline-path (prompt)
  "Select a headline in any open org file and return marker to it."
  (let (entries)
    (dolist (b (buffer-list))
      (with-current-buffer b
        (when (derived-mode-p 'org-mode)
          (setq entries
                (nconc entries
                       (counsel-outline-candidates
                        (cdr (assq 'org-mode counsel-outline-settings))
                        (counsel-org-goto-all--outline-path-prefix)))))))
    (let*
        ((sel
          (completing-read prompt entries nil t nil
                           'counsel-org-goto-history
                           ))
         (split (s-split-up-to " " sel 1))
         (filename (car split))
         (path (apply 's-split "/" (cdr split)))
         )
      (org-find-olp `(,filename ,@path))
      )))

(defun org-goto-headline ()
  "Go to any open org headline."
  (interactive)
  (org-goto-marker-or-bmk (org-get-headline-path "Goto: ")))


(defun org-insert-under-headline ()
  "Insert yanked text as last line under selected org headline."
  (interactive)
  (save-window-excursion
    (org-goto-marker-or-bmk
     (org-get-headline-path "Insert under: "))
    (outline-next-heading)
    (counsel-yank-pop)
    (newline)))

(defun org-config-new-block ()
  "Create a new code block belonging to specific module."
  (interactive)
  (let ((current-headline
         (string-inflection-title-to-lisp-case-function (nth 4
         ↪  (org-heading-components)))))
    (insert (concat "*** Description\n#+begin_src emacs-lisp :tangle modules/"
    ↪  current-headline ".el\n\n#+end_src")))
  (evil-previous-visual-line))
```

```
(provide 'org-functions)
```

4.24.5 Org Citations

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

Requirements

```
;(package! citar-capf :recipe (:host github :repo "mclear-tools/citar-capf"))
(package! zotra)
```

Code

```
(use-package! citar-embark
  :after citar embark
  :config (citar-embark-mode))

(use-package! citar
  :custom
  (org-cite-insert-processor 'citar)
  (org-cite-follow-processor 'citar)
  (org-cite-activate-processor 'citar)
  )

(after! citar
(setq citar-templates
      '((main . "${author editor:30}     ${date year issued:4}
      ↪  ${title:80}")
        (suffix . "          ${=key= id:15}    ${=type=:10}    ${tags
        ↪  keywords:*}")
        (preview . "${author editor} (${year issued date}) ${title}, ${journal
        ↪  journaltitle publisher container-title collection-title}.\n")
        (note . "Notes on ${author editor}, ${title}")))

(setq citar-symbols
      `((file ,(all-the-icons-faicon "file-o" :face 'all-the-icons-green
      ↪  :v-adjust -0.1) . " ")
        (note ,(all-the-icons-material "speaker_notes" :face
        ↪  'all-the-icons-blue :v-adjust -0.3) . " ")
        (link ,(all-the-icons-octicon "link" :face 'all-the-icons-orange
        ↪  :v-adjust 0.01) . " "))))

(setq citar-symbol-separator " ")

(defun ex/search-pdf-contents (keys-entries &optional str)
  "Search pdfs."
  (interactive (list (citar-select-refs)))
  (let ((files (hash-table-to-value-list
                (citar-get-files
                 keys-entries)))
        (search-str (or str (read-string "Search string: "))))
    (pdf-occur-search files search-str t)))

(defun hash-table-to-value-list (hashtable)
```

```
  "Convert a hash table to a list of values"
  (let ((vlist '()))
    (maphash '(lambda (key value) (push value vlist)) hashtable)
    (flatten-list vlist)))

;; with this, you can exploit embark's multitarget actions, so that you can run
↪  `embark-act-all`
(after! embark
  (add-to-list 'embark-multitarget-actions #'ex/search-pdf-contents))

(setq zotra-cli-command '("node" "/home/user/dox/install/zotra-cli/index.js"))

(provide 'org-citations)
```

4.24.6 Org Links

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

Requirements

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))

    (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)
```

4.24.7 Org Capture

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

Requirements

Code

```
(setq +org-capture-todo-file "/home/user/sync/agenda/todo.org"
      +org-capture-notes-file "/home/user/sync/agenda/notes.org"
      +org-capture-journal-file "/home/user/sync/agenda/journal.org")

; Handled by Doom Emacs
  ;; (add-to-list 'org-capture-templates
  ;;              '("Ö" "Todo" entry ; key, name, type
  ;;                (file+olp+datetree +org-capture-todo-file) ; target
  ;;                "* TODO %^{Title}\n%T %?"
  ;;                :prepend t ; properties
  ;;                :kill-buffer t))

  ;; (add-to-list 'org-capture-templates
  ;;              '("Ä" "Note" entry ; key, name, type
  ;;                (file+olp+datetree +org-capture-notes-file) ; target
  ;;                "* %^{Title} %^G\n%T %?"
  ;;                :prepend t ; properties
  ;;                :kill-buffer t))


(provide 'org-capture-tweaks)
```

4.25 Org Appear

4.25.1 Requirements

```
(package! org-appear :recipe (:host github :repo "awth13/org-appear"))
```

4.25.2 Code

```
(add-hook 'org-mode-hook 'org-appear-mode)
```

4.26 Languages

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

4.26.1 Requirements

4.26.2 Code

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

;; CANoe/CAPL
(define-derived-mode capl-mode
  c-mode "CAPL"
  "Major mode for CANoe/CAPL."
  (flycheck-mode 0))

;; 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)
```

4.27 Human languages

4.27.1 Language Baybayin

```
(load-module 'language-baybayin)
```

Requirements

Code

```
(quail-define-package
 "baybayin" "UTF-8" " " t
 "Baybayin input method."
 nil t nil nil nil nil nil nil nil nil t)

(quail-define-rules
 ("a" [" "])
 ("b" [" "])
 ("ba" [" "])
 ("be" [" "])
 ("bi" [" "])
 ("bu" [" "])
 ("bo" [" "])
 ("c" [" "])
 ("ca" [" "])
 ("ce" [" "])
 ("ci" [" "])
 ("cu" [" "])
 ("co" [" "])
```

```
("d" [" "])
("da" [" "])
("de" [" "])
("di" [" "])
("du" [" "])
("do" [" "])
("e" [" "])
("f" [" "])
("fa" [" "])
("fe" [" "])
("fi" [" "])
("fu" [" "])
("fo" [" "])
("g" [" "])
("ga" [" "])
("ge" [" "])
("gi" [" "])
("gu" [" "])
("go" [" "])
("h" [" "])
("ha" [" "])
("he" [" "])
("hi" [" "])
("hu" [" "])
("ho" [" "])
("i" [" "])
("j" [" "])
("ja" [" "])
("je" [" "])
("ji" [" "])
("ju" [" "])
("jo" [" "])
("k" [" "])
("ka" [" "])
("ke" [" "])
("ki" [" "])
("ku" [" "])
("ko" [" "])
("l" [" "])
("la" [" "])
("le" [" "])
("li" [" "])
("lu" [" "])
("lo" [" "])
("m" [" "])
("ma" [" "])
("me" [" "])
("mi" [" "])
("mu" [" "])
("mo" [" "])
("n" [" "])
("na" [" "])
("ne" [" "])
("ni" [" "])
("nu" [" "])
("no" [" "])
("ng" [" "])
```

```
("nga" [" "])
("nge" [" "])
("ngi" [" "])
("ngu" [" "])
("ngo" [" "])
("o" [" "])
("p" [" "])
("pa" [" "])
("pe" [" "])
("pi" [" "])
("pu" [" "])
("po" [" "])
("q" [" "])
("qa" [" "])
("qe" [" "])
("qi" [" "])
("qu" [" "])
("qo" [" "])
;; ("r" [" "])
;; ("ra" [" "])
;; ("re" [" "])
;; ("ri" [" "])
;; ("ru" [" "])
;; ("ro" [" "])
("r" [" "])
("ra" [" "])
("re" [" "])
("ri" [" "])
("ru" [" "])
("ro" [" "])
("s" [" "])
("sa" [" "])
("se" [" "])
("si" [" "])
("su" [" "])
("so" [" "])
("t" [" "])
("ta" [" "])
("te" [" "])
("ti" [" "])
("tu" [" "])
("to" [" "])
("u" [" "])
("v" [" "])
("va" [" "])
("ve" [" "])
("vi" [" "])
("vu" [" "])
("vo" [" "])
("w" [" "])
("wa" [" "])
("we" [" "])
("wi" [" "])
("wu" [" "])
("wo" [" "])
("x" [" "])
("xa" [" "])
```

```
 ("xe" [" "])
 ("xi" [" "])
 ("xu" [" "])
 ("xo" [" "])
 ("y" [" "])
 ("ya" [" "])
 ("ye" [" "])
 ("yi" [" "])
 ("yu" [" "])
 ("yo" [" "])
 ("z" [" "])
 ("za" [" "])
 ("ze" [" "])
 ("zi" [" "])
 ("zu" [" "])
 ("zo" [" "])
 ("," [" "])
 ("." [" "]))

(defvar tagalog-font-doctrina nil "Whether Doctrina 1593 Tagalog font is used")

(defun toggle-tagalog-font-doctrina ()
  "Set Tagalog font to Doctrina 1593"
  (interactive)
  (setq tagalog-font-doctrina (not tagalog-font-doctrina))
  (if tagalog-font-doctrina
    (set-fontset-font "fontset-default" 'tagalog (font-spec :family "Tagalog
    ↪  Doctrina 1593"))
    (set-fontset-font "fontset-default" 'tagalog (font-spec :family "Noto Sans
    ↪  Tagalog"))))

(set-fontset-font "fontset-default" 'tagalog (font-spec :family "Noto Sans
↪  Tagalog"))
;;  is for some reason not in Source Code Pro Semibold
;; (set-fontset-font "fontset-default" '(#x1700 . #x1700) (font-spec :family
↪  "Tagalog Doctrina 159"))

(provide 'language-baybayin)
```

4.27.2 Language Arabic

```
(load-module 'language-arabic)
```

Requirements

Code

```
(quail-define-package
 "phonetic arabic" "UTF-8" " " t
 "Phonetic input method."
 nil t nil nil nil nil nil nil nil nil t)

(quail-define-rules
 ("0" [" "])
 ("1" [" "])
```

```
 ("2" [" "])
 ("3" [" "])
 ("4" [" "])
 ("5" [" "])
 ("6" [" "])
 ("7" [" "])
 ("8" [" "])
 ("9" [" "])
 ("ah" [" "])
 ("aa" [" "])
 ("aaa" [" "])
 ("ae" [" "])
 ("ai" [" "])
 ("ao" [" "])
 ("au" [" "])
 ("b" [" "])
 ("t" [" "])
 ("th" [" "])
 ("j" [" "])
 ("H" [" "])
 ("kh" [" "])
 ("d" [" "])
 ("dh" [" "])
 ("r" [" "])
 ("z" [" "])
 ("s" [" "])
 ("sh" [" "])
 ("S" [" "])
 ("D" [" "])
 ("T" [" "])
 ("Z" [" "])
 ("E" [" "])
 ("gh" [" "])
 ("f" [" "])
 ("q" [" "])
 ("k" [" "])
 ("l" [" "])
 ("m" [" "])
 ("n" [" "])
 ("h" [" "])
 ("w" [" "])
 ("y" [" "])
 ("p" [" "])
 ("v" [" "])
 ("g" [" "])
 ("ch" [" "])
 ("zh" [" "])
 ("a" [" "])
 ("i" [" "])
 ("u" [" "])
 ("n" [" "])
 ("e" [" "])
 ("o" [" "])
 )
(provide 'language-arabic)
```

4.28 Email

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

4.28.1 Requirements

4.28.2 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))))))

(setq mu4e-notification-support t)
(after! mu4e-alert
  (mu4e-alert-enable-notifications)
  (mu4e-alert-set-default-style 'libnotify)
  (setq +mu4e-alert-bell-cmd '("mail-alert.sh"))
  (mu4e-update-mail-and-index 1)) ; fetch emails in background)
(provide 'email)
```

mail-alert.sh

```
#!/bin/env sh

#paplay "/usr/share/sounds/freedesktop/stereo/message.oga"

mu_find_cmd() {
    mu find --sortfield=date --reverse --maxnum=1 --fields \"$1\" flag:unread |
    ↪  tr -d '"'
}

SENDER=$(mu_find_cmd "f")
SUBJECT=$(mu_find_cmd "s")

dunstify -h string:x-dunst-stack-tag:email -i
↪  "/home/user/dox/icons/mail-unread-symbolic.svg" "$SENDER" "$SUBJECT"
```

4.28.3 Email Config

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

Requirements

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
      sendmail-program "/usr/bin/msmtp"
      smtpmail-stream-type 'starttls
      message-sendmail-f-is-evil t
      message-sendmail-extra-arguments '("--read-envelope-from")
;      message-send-mail-function 'smtpmail-send-it
      message-send-mail-function 'message-send-mail-with-sendmail
      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)
  (mu4e--goto-inbox))

(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))

; Make sure doom doesn't move mu4e article buffer into popup
; Show it in same window
(after! mu4e
  (set-popup-rule! "*mu4e-article*" :ignore 1))

;(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)

; Ignore popup rules to make sure emails are shown in same window
(after! mu4e
  (set-popup-rule! "^\\*mu4e-headers\\*" :ignore t)
  (set-popup-rule! "^\\*mu4e-view\\*" :ignore t))


(provide 'email-config)
```

4.28.4 Email Accounts

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

Requirements

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" )
                ( mu4e-drafts-folder               .
                ↪  "/gmail/[Gmail]/Entw&APw-rfe" )
                ( mu4e-sent-folder             . "/gmail/[Gmail]/Gesendet" )
                ( mu4e-trash-folder            . "/gmail/[Gmail]/Papierkorb"
                ↪  )
                ( mu4e-refile-folder           . gmail-refile )
                ))
       ,(make-mu4e-context
         :name "emysliwietz"
         :enter-func (lambda () (mu4e-message "Switching to emysliwietz
         ↪  context"))
         :leave-func (lambda () (mu4e-message "Leaving emysliwietz context"))
         ;; we match based on the contact-fields of the message
         :vars '(
                ( user-full-name               . "Egidius Mysliwietz" )
                ( user-mail-address                . "egidius@mysliwietz.de"
                ↪  )
                ( mu4e-archive-folder          . "/emysliwietz/Archive" )
                ( mu4e-drafts-folder           . "/emysliwietz/Drafts" )
                ( mu4e-sent-folder             . "/emysliwietz/Sent" )
                ( mu4e-trash-folder            . "/emysliwietz/Trash" )
                ( mu4e-refile-folder           . gmail-refile )
                ))
      ;,(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 "ru"
         :enter-func (lambda () (mu4e-message "Switch to ru context"))
         :leave-func (lambda () (mu4e-message "Leaving ru context"))
         :vars '(
                ( user-full-name                    . "Egidius Mysliwietz" )
                ( user-mail-address                .
                ↪  "egidius.mysliwietz@ru.nl" )
                ( mu4e-drafts-folder           . "/ru/Drafts" )
                ( mu4e-sent-folder             . "/ru/Sent Items" )
                ( mu4e-trash-folder            . "/ru/Deleted Items" )
```

```
                        ( mu4e-archive-folder          . "/ru/Archive" )
                ))
        ;,(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)
```

4.28.5 mbsync config

RU

```
IMAPAccount ru
Host smtp.office365.com
User egidius.mysliwietz@ru.nl
PassCmd "secret-tool lookup mail-mbsync ru"
Port 993
```

```
SSLType IMAPS
AuthMechs XOAUTH2
CertificateFile /etc/ssl/certs/ca-certificates.crt

IMAPStore ru-remote
Account ru

MaildirStore ru-local
Path ~/mail/ru/
Inbox ~/mail/ru/INBOX
Subfolders Verbatim

Channel ru
Far :ru-remote:
Near :ru-local:
Create Both
Expunge Both
Sync All
Patterns *
SyncState *

# ##########################################
```

Strato

```
##########################################

IMAPAccount emysliwietz
Host imap.strato.de
User egidius@mysliwietz.de
PassCmd "secret-tool lookup mail-mbsync emysliwietz"
#Port 993
SSLType IMAPS
AuthMechs Login
CertificateFile /etc/ssl/certs/ca-certificates.crt

IMAPStore emysliwietz-remote
Account emysliwietz

MaildirStore emysliwietz-local
Path ~/mail/emysliwietz/
Inbox ~/mail/emysliwietz/INBOX
Subfolders Verbatim

Channel emysliwietz
Far :emysliwietz-remote:
Near :emysliwietz-local:
Create Both
Expunge Both
Sync All
Patterns *
SyncState *

##########################################
```

gmail

```
# ##########################################
IMAPAccount gmail
Host imap.gmail.com
User e.p.mysliwietz@gmail.com
PassCmd "secret-tool lookup mail-mbsync gmail"
Port 993
SSLType IMAPS
AuthMechs Login
CertificateFile /etc/ssl/certs/ca-certificates.crt

IMAPStore gmail-remote
Account gmail

MaildirStore gmail-local
Path ~/mail/gmail/
Inbox ~/mail/gmail/INBOX
Subfolders Verbatim

Channel gmail
Far :gmail-remote:
Near :gmail-local:
Create Both
Expunge Both
Sync All
Patterns *
SyncState *

# ##########################################
```

mu init

```
mu init --maildir ~/mail --my-address="e.p.mysliwietz@gmail.com"
↪  --my-address="egidius@mysliwietz.de"
↪  --my-address="egidius.mysliwietz@gmx.de"
mu index
```

4.28.6 msmtprc - Sending mails

default settings

```
defaults
auth        on
tls        on
tls_trust_file        /etc/ssl/certs/ca-certificates.crt
logfile /home/user/.config/msmtp/msmtp.log
```

gmail

```
account gmail
host smtp.gmail.com
from e.p.mysliwietz@gmail.com
user e.p.mysliwietz@gmail.com
passwordeval "secret-tool lookup mail-mbsync gmail"
```

strato

```
account emysliwietz
host smtp.strato.de
port 587
from egidius@mysliwietz.de
user egidius@mysliwietz.de
passwordeval "secret-tool lookup mail-mbsync emysliwietz"
```

ru

```
account ru
host smtp.office365.com
port 587
auth xoauth2
from egidius.mysliwietz@ru.nl
user egidius.mysliwietz@ru.nl
passwordeval "secret-tool lookup mail-mbsync ru"
```

default account

```
account default : emysliwietz
```

4.29 Latex Tweaks

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

4.29.1 Requirements

4.29.2 Code

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

(setq TeX-command-extra-options "--shell-escape")
(setq TeX-view-program-selection
      '((output-pdf "Zathura")
        (output-pdf "Evince")
        (output-pdf "PDF Tools")
        ((output-dvi has-no-display-manager)
         "dvi2tty")
        ((output-dvi style-pstricks)
         "dvips and gv")
        (output-dvi "xdvi")
        (output-html "xdg-open")
        (output-pdf "preview-pane")))

(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 backup-copy-pdf-after-compilation (pdf-file)
  "Copy the newly created PDF file to the same directory as the source TeX file
  ↪  with a '.backup.pdf' suffix."
  (let ((source-directory (file-name-directory pdf-file))
        (pdf-file-name (file-name-nondirectory pdf-file))
        (backup-pdf-file (concat (file-name-sans-extension pdf-file)
        ↪  ".backup.pdf")))
        ;(message backup-pdf-file)
    (copy-file pdf-file (concat "" backup-pdf-file) t))) ;source-directory


; Create backup copy of pdf after successful compilation
; This way, you can read the pfd even while the new version is compiling
(add-hook 'TeX-after-compilation-finished-functions
↪  #'backup-copy-pdf-after-compilation)

(defun auto-async-export ()
  (let ((keywords (org-collect-keywords '("auto_async_export"))))
    (when (and keywords
               (eq major-mode 'org-mode)
               (string-equal "t" (pop (cdr (car keywords)))))
      (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)))))

(add-hook 'TeX-language-en-hook
          (lambda () (ispell-change-dictionary "english")
            (setq TeX-quote-language `("german" "\\enquote{" "}"
            ↪  ,TeX-quote-after-quote))))

(add-hook 'TeX-language-de-hook
          (lambda () (ispell-change-dictionary "german")
            (setq TeX-quote-language `("german" "\\enquote{" "}"
            ↪  ,TeX-quote-after-quote))))


  ;; 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))

(setq deft-recursive t
      deft-use-filename-as-title t)
(let* ((base-dir "/home/user/dox/bib/")
        (bibfile (concat base-dir "bib.bib"))
        (notes-dir (concat base-dir "notes/"))
        (roam-notes-dir (concat org-roam-directory "/"))
        (lib-dir (concat base-dir "papers/")))
  (setq bibtex-completion-bibliography bibfile
        bibtex-completion-library-path `(,lib-dir)
        bibtex-completion-notes-path roam-notes-dir
        bibtex-completion-pdf-field "file"
        citar-bibliography `(,bibfile)
        citar-library-paths `(,lib-dir)
        citar-notes-paths `(,roam-notes-dir)
        deft-directory roam-notes-dir
        deft-default-extension "org"
        org-noter-notes-search-path `(,roam-notes-dir)
        org-cite-global-bibliography `(,bibfile)))

(setq bibtex-completion-notes-template-multiple-files
      (concat
       "#+TITLE: ${title}\n"
       "#+ROAM_KEY: cite:${=key=}\n"
       "* TODO Notes\n"
       ":PROPERTIES:\n"
       ":Custom_ID: ${=key=}\n"
       ":NOTER_DOCUMENT: %(orb-process-file-field \"${=key=}\")\n"
       ":AUTHOR: ${author-abbrev}\n"
       ":JOURNAL: ${journaltitle}\n"
       ":DATE: ${date}\n"
       ":YEAR: ${year}\n"
       ":DOI: ${doi}\n"
       ":URL: ${url}\n"
       ":END:\n\n"
       ))


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

(after! latex
  (require 'engrave-faces)
  (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)
```

4.30 PDF and Annotation Tweaks

```
(load-module 'pdf-and-annotation-tweaks)
```

4.30.1 Requirements

```
(package! org-noter)
```

4.30.2 Code

```
(use-package! org-noter
  :after (:any org pdf-view)
  :config
  (setq
   ;; Please stop opening frames
   org-noter-always-create-frame nil
   org-noter-notes-window-location 'horizontal-split
   ;; I want to see the whole file
```

```
 org-noter-hide-other nil
 org-noter-auto-save-last-location t
 org-noter-suggest-from-attachments t
 org-noter-doc-split-percentage '(0.7 . 0.3)
 org-noter-doc-split-fraction '(0.7 . 0.3)
 )

(defun org-noter-insert-short-note ()
  "Insert note and switch focus back to pdf."
  (interactive)
  (save-window-excursion
  (org-noter-insert-note)))

(defun citar-noter-resume ()
  "Open notes to a document and resume editing"
  (interactive)
  (call-interactively 'citar-open-notes)
  (evil-goto-line)
  (org-noter)
  (pdf-view-noter-keymap-load))

(defun pdf-view-high-contrast-theme ()
  "Sets black and white high contrast theme"
  (interactive)
  (pdf-view-themed-minor-mode -1)
  (setq old-pdf-view-midnight-colors pdf-view-midnight-colors)
  (setq pdf-view-midnight-colors '("#ffeeee" . "#000000"))
  (pdf-view-midnight-minor-mode 1))

(defun pdf-view-undo-high-contrast-theme ()
  "Unsets black and white high contrast theme"
  (interactive)
  (pdf-view-themed-minor-mode -1)
  (setq pdf-view-midnight-colors old-pdf-view-midnight-colors)
  (pdf-view-midnight-minor-mode 1))



(defun pdf-view-theme-cycle ()
  "Cycle between emacs, midnight and white theme"
  (interactive)
  (cond
   ((bound-and-true-p pdf-view-themed-minor-mode)
    (progn
      (pdf-view-themed-minor-mode -1)
      (pdf-view-midnight-minor-mode 1)))
   ((bound-and-true-p pdf-view-midnight-minor-mode)
    (progn
      (pdf-view-midnight-minor-mode -1)
      (pdf-view-themed-minor-mode -1)))
   (t (progn
        (pdf-view-midnight-minor-mode -1)
        (pdf-view-themed-minor-mode 1)
        )))))

(defun org-noter-new-heading ()
  "DOES NOT WORK: Copy the current heading and the next visible heading."
```

```
  (interactive)
  (org-noter--with-valid-session
  (let ((headline (completing-read "New chapter title: " nil))
        (page (format "%s" (org-noter--get-location-page
        ↪  (org-noter--doc-approx-location))))
        )
   (save-excursion
     (switch-to-buffer (org-noter--session-notes-buffer session))
    (save-excursion
      ;; Move up to the top-level heading
      (org-up-heading-all 10)
      (let ((start (point)))
        ;; Move to the next visible heading
        (org-next-visible-heading 1)
        (let ((end (point)))
          ;; Copy the region from start to end
          (kill-ring-save start end)
          )))
    (yank)
    (save-excursion
      (org-edit-headline headline)
      (org-set-property "NOTER_PAGE" page)
    )))))

(defun pdf-view-noter-keymap-load ()
  "Load org noter keymap for pdf-view"
  (interactive)
  (map! :after pdf-view
        :map pdf-view-mode-map
        :n "n" 'org-noter-insert-short-note
        :n "N" 'org-noter-insert-note
        :n "r" 'pdf-view-rotate
        :n "i" 'pdf-view-theme-cycle
        :n "I" 'pdf-view-high-contrast-theme
        :n "c" 'org-noter-new-heading
        :ne "<down-mouse-1>" 'ignore ; Because marking would reset rotation
        :n "+" 'pdf-view-enlarge
        :n "-" 'pdf-view-shrink))

(pdf-view-noter-keymap-load)

(setq pdf-view-resize-factor (/ 5 3.0))

(defun pdf-shrink ()
  "Shrink a pdf in pdf-view.
I'm not sure why pdf-view-resize-factor isn't consistent in both directions, so
↪  I need two seperate factors."
  (interactive)
  (pdf-view-enlarge 0.6))

(defun pdf-enlarge ()
  "Enlarge a pdf in pdf-view.
I'm not sure why pdf-view-resize-factor isn't consistent in both directions, so
↪  I need two seperate factors."
  (interactive)
  (pdf-view-enlarge 0.8))
```

```
(after! pdf-view
  (bind-key (kbd "C-s") 'pdf-occur pdf-view-mode-map)
  (bind-key (kbd "<C-mouse-4>") 'pdf-enlarge pdf-view-mode-map)
  (bind-key (kbd "<C-mouse-5>") 'pdf-shrink pdf-view-mode-map)

  (add-hook 'pdf-view-hook 'pdf-view-themed-minor-mode))

(provide 'pdf-and-annotation-tweaks)
```

4.31 Tolino

```
(load-module 'tolino)
```

4.31.1 Requirements

4.31.2 Code

```
(setq tolino-dir "/home/user/dox/tolino/")

(defun citar-move-to-tolino (keys-entries &optional str)
  "Search pdfs."
  (interactive (list (citar-select-refs)))
  (let ((files (hash-table-to-value-list
                (citar-get-files
                 keys-entries))))
  (cl-map 'vector #'(lambda (f) (copy-file f tolino-dir 1 t)) files)))

(provide 'tolino)
```

4.32 Secret Service Tweaks

```
(load-module 'secret-service-tweaks)
```

4.32.1 Requirements

4.32.2 Code

```
  ;; Load and patch secrets
  (use-package! secrets
    :commands (secrets-search-items
               secrets-get-secret
               secrets-get-attributes)
    :config
    ;; Adds a patch to fix behavior with KeepassXC
    (defun secrets-unlock-item (collection item)
      "Unlock item labeled ITEM from collection labeled COLLECTION.
    If successful, return the object path of the item."
      (let ((item-path (secrets-item-path collection item)))
        (unless (secrets-empty-path item-path)
          (secrets-prompt
           (cadr
            (dbus-call-method
```

```
            :session secrets-service secrets-path secrets-interface-service
            "Unlock" `(:array :object-path ,item-path)))))
      item-path))

  ;; Adds a patch to fix behavior with KeepassXC
  (defun secrets-get-secret (collection item)
    "Return the secret of item labeled ITEM in COLLECTION.
  If there are several items labeled ITEM, it is undefined which
  one is returned.  If there is no such item, return nil.

  ITEM can also be an object path, which is used if contained in COLLECTION."
    (let ((item-path (secrets-unlock-item collection item)))
      (unless (secrets-empty-path item-path)
        (dbus-byte-array-to-string
         (nth 2
              (dbus-call-method
               :session secrets-service item-path secrets-interface-item
               "GetSecret" :object-path secrets-session-path))))))

(require 'async)
(defun secrets-get-secret-async (db-name password-name variable)
  (async-start
   ; Async function
   `(lambda ()
      (require 'secrets)
      (secrets-get-secret ,db-name ,password-name)
      )

   ; Callback
   `(lambda (result)
      (setq ,variable result)))
  )


(provide 'secret-service-tweaks)
```

4.33 AI Assistants

4.33.1 GPT.el Tweaks

ChatGPT no longer useful because of API limits

```
(load-module 'gpt-el-tweaks)
```

Requirements

```
(package! gptel :recipe (:host github :repo "karthink/gptel"))
```

Code

```
(use-package! gptel
  :defer t
  :config
  ;(secrets-get-secret-async "Passwords" "ChatGPT API Key" 'gptel-api-key)
  (async-start
```

```
   ; Async function
   `(lambda ()
      (require 'secrets)
      (secrets-get-secret "Passwords" "Gemini API Key")
      )

   ; Callback
   `(lambda (result)
      (setq gptel-backend (gptel-make-gemini "Gemini" :key result :stream t)
            gptel--system-message ""
            gptel-model 'gemini-1.5-pro-latest
    )))
  )
(provide 'gpt-el-tweaks)
```

4.34 Calendar Tweaks

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

4.34.1 Requirements

4.34.2 Code

```
;; deutsche Feiertage
;; Show holidays in Org Agenda
(setq org-agenda-include-diary t)

(setq calendar-week-start-day 1
      calendar-time-display-form '(24-hours ":" minutes)
      calendar-date-style 'european
      calendar-day-name-array ["Sonntag" "Montag" "Dienstag" "Mittwoch"
                               "Donnerstag" "Freitag" "Samstag"]
      calendar-month-name-array ["Januar" "Februar" "März" "April" "Mai"
                                 "Juni" "Juli" "August" "September"
                                 "Oktober" "November" "Dezember"])

(setq solar-n-hemi-seasons
      '("Frühlingsanfang" "Sommeranfang" "Herbstanfang" "Winteranfang"))


(setq holiday-german-general-holidays
      '((holiday-fixed 1 1 "Neujahr")
        (holiday-fixed 5 1 "1. Mai")
        (holiday-fixed 10 3 "Tag der Deutschen Einheit")))

(setq holiday-german-christian-holidays
      '((holiday-float 12 0 -4 "1. Advent" 24)
        (holiday-float 12 0 -3 "2. Advent" 24)
        (holiday-float 12 0 -2 "3. Advent" 24)
        (holiday-float 12 0 -1 "4. Advent" 24)
        (holiday-fixed 12 25 "1. Weihnachtstag")
        (holiday-fixed 12 26 "2. Weihnachtstag")
        (holiday-fixed 1 6 "Heilige Drei Könige")
        (holiday-easter-etc -48 "Rosenmontag")
        (holiday-easter-etc -3 "Gründonnerstag")
```

```
        (holiday-easter-etc  -2 "Karfreitag")
        (holiday-easter-etc   0 "Ostersonntag")
        (holiday-easter-etc  +1 "Ostermontag")
        (holiday-easter-etc +39 "Christi Himmelfahrt")
        (holiday-easter-etc +49 "Pfingstsonntag")
        (holiday-easter-etc +50 "Pfingstmontag")
        (holiday-easter-etc +60 "Fronleichnam")
        (holiday-fixed 8 15 "Mariæ Himmelfahrt")
        (holiday-fixed 11 1 "Allerheiligen")
        (holiday-fixed 11 2 "Allerseelen")
        (holiday-float 11 3 1 "Buß- und Bettag" 16)
        (holiday-float 11 0 1 "Totensonntag" 20)))

(setq holiday-filipino-general-holidays
      '((holiday-fixed 1 23 "1st Philippine Republic Day")
        (holiday-fixed 4 9 "Araw ng Kagitingan")
        (holiday-fixed 6 12 "Filipino Independence Day")
        (holiday-fixed 8 21 "Ninoy Aquino Day")
        (holiday-fixed 9 3 "Surrender of Tomoyuki Yamashita")
        (holiday-float 8 1 -1 "National Heros of the Philippines")
        (holiday-fixed 12 30 "Jose Rizal Day")))

(setq holiday-filipino-christian-holidays
      '((holiday-fixed 9 8 "Feast of the Nativity of Mary")
        (holiday-fixed 11 30 "Bonifacio Day")))

(setq holiday-us-elections
      '((holiday-sexp
       '(if (zerop (% year 4))
            (calendar-gregorian-from-absolute
              (1+ (calendar-dayname-on-or-before
                    1 (+ 6 (calendar-absolute-from-gregorian
                             (list 11 1 year)))))))
       "US Presidential Election")
))

(setq calendar-christian-all-holidays-flag t
      calendar-islamic-all-holidays-flag t)

(setq calendar-holidays
      (append holiday-general-holidays holiday-local-holidays
      ↪  holiday-other-holidays
             holiday-solar-holidays holiday-christian-holidays
             ↪  holiday-german-general-holidays
             ↪  holiday-german-christian-holidays holiday-us-elections
             ↪  holiday-hebrew-holidays holiday-islamic-holidays
             ↪  holiday-oriental-holidays holiday-filipino-general-holidays
             ↪  holiday-filipino-christian-holidays))

(defun =calendar-no-evil ()
  (interactive)
  (=calendar)
  (evil-local-mode nil))

(global-set-key (kbd "s-c") '=calendar-no-evil)

(provide 'calendar-tweaks)
```

4.34.3 Org GCal tweaks

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

Requirements

```
(package! org-gcal)
(package! request)
(package! alert)
(package! persist)
(package! aio)
(package! oauth2-auto)
```

Code

```
(require 'async)

(async-start
 (lambda ()
   (require 'secrets)
   (let ((cal1 (secrets-get-secret "Passwords" "Org GCal Calendar 1"))
         (cal2 (secrets-get-secret "Passwords" "Org GCal Calendar 2")))
     `(
       (,cal1 .  "~/sync/agenda/gcal.org")
       (,cal2 .  "~/sync/agenda/gcal.org")
       )))
 (lambda (result)
   (setq org-gcal-fetch-file-alist result)))

(setq org-gcal-notify-p t
      plstore-cache-passphrase-for-symmetric-encryption t)


(async-start
 (lambda ()
   (require 'secrets)
   `(,(secrets-get-secret "Passwords" "Org GCal Client ID")
     ,(secrets-get-secret "Passwords" "Org GCal Client Secret"))
   )
 (lambda (result)
   (let ((id (nth 0 result))
         (sc (nth 1 result))
         )
     (setq org-gcal-client-id id
           org-gcal-client-secret sc)
     (org-gcal-reload-client-id-secret)
     (require 'oauth2-auto)
     (require 'plstore)
     (require 'org-gcal)
     )))

(provide 'org-g-cal-tweaks)
```

4.35 Thinkpad Tweaks

```
;(load-module 'thinkpad-tweaks)
```

4.35.1 Requirements

4.35.2 /etc/systemd/system/battery-threshold.service

```
[Unit]
Description=Set battery charge control start and end thresholds

[Service]
Type=oneshot
ExecStart=/bin/bash -c '[ -d /sys/class/power_supply/BAT0 ] && echo 40 >
↪  /sys/class/power_supply/BAT0/charge_control_start_threshold'
ExecStart=/bin/bash -c '[ -d /sys/class/power_supply/BAT0 ] && echo 60 >
↪  /sys/class/power_supply/BAT0/charge_control_end_threshold'
ExecStart=/bin/bash -c '[ -d /sys/class/power_supply/BAT1 ] && echo 40 >
↪  /sys/class/power_supply/BAT1/charge_control_start_threshold'
ExecStart=/bin/bash -c '[ -d /sys/class/power_supply/BAT1 ] && echo 60 >
↪  /sys/class/power_supply/BAT1/charge_control_end_threshold'
User=root
```

4.35.3 /etc/systemd/system/battery-threshold.timer

```
[Unit]
Description=Run battery-threshold at startup

[Timer]
OnStartupSec=5min

[Install]
WantedBy=timers.target
```

4.35.4 Full Battery script

Charges battery to 100% full by undoing service, active until restart

```
#!/bin/env sh
# Undoes battery-threshold.service and charges battery to 100% until restart
# Run as root

/bin/bash -c '[ -d /sys/class/power_supply/BAT0 ] && echo 0 >
↪  /sys/class/power_supply/BAT0/charge_control_start_threshold'
/bin/bash -c '[ -d /sys/class/power_supply/BAT0 ] && echo 100 >
↪  /sys/class/power_supply/BAT0/charge_control_end_threshold'
/bin/bash -c '[ -d /sys/class/power_supply/BAT1 ] && echo 0 >
↪  /sys/class/power_supply/BAT1/charge_control_start_threshold'
/bin/bash -c '[ -d /sys/class/power_supply/BAT1 ] && echo 100 >
↪  /sys/class/power_supply/BAT1/charge_control_end_threshold'
```

4.36 Nano Tweaks

```
;(load-module 'nano-tweaks)
```

4.36.1 Requirements

```
(package! nano :recipe (:host github :repo "rougier/nano-emacs"))
;(package! nano-theme :recipe (:host github :repo "rougier/nano-theme"))
(package! mu4e-dashboard :recipe (:host github :repo "rougier/mu4e-dashboard"))
(package! svg-tag-mode)
(package! ts)
(package! nano-sidebar :recipe (:host github :repo "rougier/nano-sidebar"))
```

4.36.2 Code

```
(require 'nano-base-colors)
(require 'nano-faces)
(require 'nano-help)
(load-module 'nano-sidebar)
(load-module 'nano-mu4e)

(defun mu4e-headers-sender-subject (msg)
  (let* ((thread  (mu4e-message-field msg :thread))
         (level (plist-get thread :level))
         (empty-parent (and thread (plist-get thread :empty-parent)))
         (child   (and thread (> (plist-get thread :level) 0)))
         (flagged (memq 'flagged  (mu4e-message-field msg :flags)))
         (unread  (memq 'unread  (mu4e-message-field msg :flags)))
         (replied (memq 'replied (mu4e-message-field msg :flags)))
         (maildir (mu4e-get-maildir msg))
         (inbox   (string= maildir "inbox"))
         (attachment (memq 'attach (mu4e-message-field msg :flags)))
         (sent    (string= maildir "sent"))
         (tags    (mu4e-message-field msg :tags))
         (list    (mu4e-message-field msg :mailing-list))
         (sender-name     (or (mu4e-message-field (car (mu4e-message-field msg
         ↪  :from)) :name)))
         (sender-address  (or (car (cdr (car (mu4e-message-field msg
         ↪  :from))))))
         (subject (mu4e-message-field msg :subject)))
    (setq messagemsg msg)
    (concat
     (if (and (not empty-parent) child)
     ;; (if (> level 0)
         (propertize "  " 'face `(:foreground ,nano-color-subtle :height 100)))

     (mu4e-headers-button
      (string-pad (or sender-name sender-address "Unknown") 40)
      (cond  ((and child unread) 'nano-face-default)
             (child              'nano-face-faded)
             (unread             'nano-face-strong)
             (t                  'nano-face-strong))

      (format "Mails from %s" sender-address)
      (format "from:%s" sender-address))
```

```
     (cond (unread ;;(propertize "   " 'face 'nano-face-default))
            (propertize "   " 'face '(:height 100)))
           (inbox
            (propertize " • " 'face 'nano-face-faded))
           ((or (not child) empty-parent tags)
            (propertize "   " 'face 'nano-face-faded)))


     (if (or (not mu4e-headers-show-threads) (not child))
         (if list (concat (mu4e-headers-button
                           "[LIST]"
                           '(:inherit nano-face-faded)
                           (format "Mails from/to %s" list)
                           (format "list:%s" list)) " ")))

     (if tags (concat
               (mapconcat
                (function (lambda (tag)
                            (mu4e-headers-button
                             (format "[%s]" tag)
                             '(:inherit (nano-face-salient nano-face-strong))
                             (format "Mails with tag %s" tag)
                             (concat "tag:" tag))))  tags " ") " "))

     (if (or (not child) empty-parent)
     ;; (if (= level 0)
         (propertize subject 'face 'nano-face-default)))))

;(keymapp mu4e-headers-mode-map)
;(keymapp "f")


(defun mu4e-headers-button (text face help query)
  "Make a clickable button with specified FACE displaying TEXT.
   When hovered, HELP is displayed. When clicked, mu4e QUERY is executed."
  (let ((map (make-sparse-keymap)))
    (set-keymap-parent map mu4e-headers-mode-map)
    (define-key map [mouse-1] `(lambda ()
                                 (interactive) (mu4e-headers-search ,query)))
    (propertize text
                'face face
                'mouse-face `(:foreground ,nano-color-background
                              :background ,nano-color-faded)
                'local-map map
                'help-echo `(lambda (window _object _point)
                              (let (message-log-max) (message ,help))))))

(defun mu4e-headers-date-button (date face)
  (concat
 (mu4e-headers-button (format-time-string "%Y" date)
                      face
                      (format-time-string "Mails from %Y" date)
                      (format-time-string "date:%Y" date))
   (propertize "/" 'face face)
   (mu4e-headers-button (format-time-string "%m" date)
                      face
```

```
                        (format-time-string "Mails from %B %Y" date)
                        (format-time-string "date:%Y%m" date))
   (propertize "/" 'face face)
   (mu4e-headers-button (format-time-string "%d" date)
                        face
                        (format-time-string "Mails from %d %B %Y" date)
                        (format-time-string "date:%Y%m%d" date))
    ))

;; (setq package-install-upgrade-built-in t)
;; (setq nano-font-size 10
;;       nano-font-family-monospaced "Source Code Pro Semibold"
;;       nano-modeline-position 'mode-line
;;       nano-modeline-style '(3d accented)
;;       doom-one-brighter-comments t)

;; ;;Default layout (optional)
;; (require 'nano-layout)

;; ;; Theme
;; (require 'nano-faces)
;; (require 'nano-theme)
;; (require 'nano-theme-dark)
;; (require 'nano-theme-light)

;; (set-face-attribute 'default nil :family "Source Code Pro" :weight
↪  'semibold)
;; (set-face-attribute 'bold nil :family "Source Code Pro" :weight 'black)
;; (require 'nano-writer)

;; (nano-theme-set-dark)
;; (call-interactively 'nano-refresh-theme)
;; (load-theme doom-theme)

;; ;; Nano session saving (optional)
;; ;(require 'nano-session)

;; ;; Nano header & mode lines (optional)
;; ;(require 'nano-modeline)

;; ;(setq widget-image-enable nil)
(provide 'nano-tweaks)
```

4.37 External Tui

```
(load-module 'external-tui)
```

4.37.1 Requirements

```
(package! eee
      :recipe
      (:host github
       :repo "eval-exec/eee.el"
       :files (:defaults "*.el" "*.sh")))
```

4.37.2 Code

```
;(setq ee-terminal-command "wezterm")
(require 'eee)
(setq ee-terminal-command "kitty")
(exwm-input-set-key (kbd "s-S-SPC") 'ee-yazi)
(map! :leader "y" 'ee-yazi)
;(exwm-input--update-global-prefix-keys)
(provide 'external-tui)
```

4.38 Bible

```
(load-module 'bible)
```

4.38.1 Requirements

4.38.2 Code

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

(defun bible-books-long (binary)
  "Return list of books of the Bible"
  (split-string-and-unquote (shell-command-to-string (format "%s -l | awk
  ↪  'NF{NF--}; 1' | sed '1s/^\xEF\xBB\xBF//'" binary)) "\n"))

(defun bible-books-short (binary)
  "Return list of abbreviated books of the Bible"
  (split-string-and-unquote (shell-command-to-string (format "%s -l | awk
  ↪  '{print $(NF)}' | sed 's/^\(//;s/\)$//;1s/^\xEF\xBB\xBF//'" binary))
  ↪  "\n"))

(defun bible-text (binary book-chapter-and-verse)
  "Return text of Bible binary at specified book, chapter and verse"
  (let* ((bcv1 (string-replace "." "" book-chapter-and-verse))
         (bcv (string-replace "," ":" bcv1)))
    (shell-command-to-string (format "%s %s | sed 's/\r//g'" binary
    ↪  (replace-regexp-in-string "\n" "" bcv)))))

(setq bible-minor-mode--last-book ""
      bible-minor-mode--last-chapter-and-verse "")

(defun bible-select-internal (binary)
  "Ask user to select bible verse using kjv-like binary"
  (let* ((book (ivy-read "Select Book: " (bible-books-long binary) :preselect
  ↪  bible-minor-mode--last-book))
         (chapter-and-verse (ivy-read (format "Choose Chapter and Verse of %s:
         ↪  " book) '() :initial-input
         ↪  bible-minor-mode--last-chapter-and-verse)))
    (setq bible-minor-mode--last-book book
          bible-minor-mode--last-chapter-and-verse chapter-and-verse)
    `(,book ,chapter-and-verse)))


(defun bible-select (binary)
  "Ask user to select bible verse using kjv-like binary"
```

```
 (let* ((pair (bible-select-internal binary))
        (book (car pair))
        (chapter-and-verse (car (cdr pair)))
        )
    (format "%s %s" book chapter-and-verse)))



(defun bible-insert (&optional verse)
  "Inserts text of the Bible of selected verse"
  (interactive)
  (if verse
      (insert (bible-text (bible-minor-mode--default-bible-binary) verse))
    (insert (bible-text (bible-minor-mode--default-bible-binary) (bible-select
    ↪  (bible-minor-mode--default-bible-binary))))))

(defun kjv ()
  "Insert Bible Verse from the King James Version"
  (interactive)
  (let ((bin "kjv"))
    (insert (bible-text bin (bible-select bin)))))

(defun menge ()
  "Insert Bible Verse from the Menge Bible"
  (interactive)
  (let ((bin "menge"))
    (insert (bible-text bin (bible-select bin)))))

(defun vul ()
  "Insert Bible Verse from the Vulgate"
  (interactive)
  (let ((bin "vul"))
    (insert (bible-text bin (bible-select bin)))))

(defun sxx ()
  "Insert Bible Verse from the Septuagint"
  (interactive)
  (let ((bin "grb"))
    (insert (bible-text bin (bible-select bin)))))

(defun grb ()
  "Insert Bible Verse from the Septuagint"
  (interactive)
  (sxx))

(setq bible-minor-mode--bible-binary-list '("menge" "kjv" "vul" "grb"))

(defun bible-minor-mode--default-bible-binary ()
  "Default bible binary to use"
  (car bible-minor-mode--bible-binary-list))

(defun set-bible-version ()
  "Change the version of the bible"
  (interactive)
  (let ((s
```

```
        (ivy-read "Select bible version: " bible-minor-mode--bible-binary-list
        ↪  :require-match t :preselect
        ↪  (bible-minor-mode--default-bible-binary))))
    (setq bible-minor-mode--bible-binary-list (append `(,s) (delete s
    ↪  bible-minor-mode--bible-binary-list)))
    (message (bible-minor-mode--default-bible-binary))))

(defun cycle-bible-version ()
  (setq bible-minor-mode--bible-binary-list (append (cdr-safe
  ↪  bible-minor-mode--bible-binary-list) `(,(car
  ↪  bible-minor-mode--bible-binary-list)))))

;;; Bible Mode

(defun bible-mode--current-verse ()
  "Current Verse at point"
  (message
   (save-excursion
     (car (s-split-up-to "\t" (thing-at-point 'line) 1))
     )))

(defun bible-mode--goto-verse (verse)
  (goto-char 0)
  (search-forward verse))

(define-derived-mode bible-mode text-mode
↪  (bible-minor-mode--default-bible-binary) "Major mode for reading the
↪  Bible.")

(map! :map bible-mode-map
      :n "x" #'bible-mode--cycle-version)

;;; Bible Minor Mode
(defun bible-tooltip (window object pos)
  (let ((verse (bible-mode--current-verse)))
    (tooltip-mode 1)
    (setq bible-minor-mode--last-verse verse)
    (let ((bible-verse (bible-text (bible-minor-mode--default-bible-binary)
    ↪  verse)))
                                        ;(message bible-verse)
      (if (> (length bible-verse) 2000)
          (substring bible-verse 0 2000)
        bible-verse))
    ))

(defun bible-show-verse ()
  "Shows tooltip for verse at point"
  (interactive)
  (setq bible-minor-mode--last-verse nil)
  (tooltip-show (bible-tooltip nil nil (point))))

(defun bible-minor-mode-highlight ()
  "Highlight bible verses"
  (font-lock-add-keywords nil `((,bible-minor-mode--bible-verse-regex 0 '(face
  ↪  font-lock-keyword-face help-echo bible-tooltip))))
  (font-lock-flush))
```

```
(defun bible-minor-mode-revert ()
  "Undo highlight bible verses"
  (font-lock-remove-keywords nil `((,bible-minor-mode--bible-verse-regex 0
  ↪  '(face font-lock-keyword-face help-echo bible-tooltip))))
  (font-lock-flush))

;;;###autoload
(define-minor-mode bible-minor-mode
  "Highlight bible verses and provide tooltips for them"
  :lighter " "
  :global nil
  (if bible-minor-mode
      (bible-minor-mode-highlight)
    (bible-minor-mode-revert)))

(add-hook 'org-mode-hook 'bible-minor-mode)

(setq bible-books-short-manual-additions '("Gen"))

(defun bible-books-regex ()
  "Combines all bible books into a partial regex"
  (string-join (append (bible-books-short "kjv")
                       (bible-books-long "kjv")
                       (bible-books-short "menge")
                       (bible-books-long "menge")
                       bible-books-short-manual-additions)
               "\\|"))


"Creates a regex for verses of all bible books"
(when (not (boundp 'bible-minor-mode--bible-verse-regex))
  (setq bible-minor-mode--bible-verse-regex
        (format
         "\\(%s\\)
         ↪  \\([[:digit:]]+\\(\\(\\:\\|\\,\\)[[:digit:]]+\\)?\\)\\(-[[:digit:]]+\\(\\(\\:\\|\\,\\)[[:digit:]]+\\
         (bible-books-regex))))
                                       ;bible-minor-mode--bible-verse-regex)

(provide 'bible)
```

4.38.3 org-link-bible

```
(load-module 'ol-bible)
```

4.38.4 Requirements

4.38.5 Code

```
                                       ; Implements a link to a bible section
↪  or verse
                                       ; [[bible:Gen~1:1][Genesis 1:1]]
(load-module 'bible)
(load-module 'ol)

(org-link-set-parameters "bible"
```

```
                        :follow #'org-bible-open
                        :export #'org-bible-export
                        :store #'org-bible-store-link)

(defun org-bible-store-link ())

(define-derived-mode bible-view-mode org-mode
↪   (bible-minor-mode--default-bible-binary)
  "A mode to view the bible in.")

(map! :map bible-view-mode-map
      :nvi "ö" #'bible-view-cycle-language
      :nvi "x" #'bible-view-cycle-language)

(defun bible-view-cycle-language ()
  "Cycle the language in a bible view buffer."
  (interactive)
  (cycle-bible-version)
  (let (pos (point))
    (erase-buffer)
    (bible-insert (org-bible-get-chapter-verse))
    (goto-char pos)
    ))

(defun bible-view ()
  "View a passage of the Bible"
  (interactive)
  (org-bible-open (bible-select (bible-minor-mode--default-bible-binary)) nil))


(defun org-bible-open (chapter-verse _)
  "Visit the bible page on CHAPTER-VERSE."
  (let ((buf (get-buffer-create (format "*Bible: %s*" chapter-verse))))
    (with-current-buffer buf
      (erase-buffer)
      (bible-view-mode)
      (bible-insert chapter-verse)
      (goto-char 0)
                                        ;(bible-mode 1)
      )
    (display-buffer buf '((display-buffer-pop-up-window)))
    ))

;; (defun org-man-store-link (&optional _interactive?)
;;   "Store a link to a bible page."
;;   (when (memq major-mode '(bible-mode))
;;     ;; This is a man page, we do make this link.
;;     (let* ((page (org-man-get-page-name))
;;            (link (concat "man:" page))
;;            (description (format "Man page for %s" page)))
;;       (org-link-store-props
;;        :type "man"
;;        :link link
;;        :description description))))

(defun org-bible-get-chapter-verse ()
  "Extract the bible chapter and verse from the buffer name."
```

```
  ;; This works for `bible-view-mode'.
  (if
      (string-match "\\*Bible: \\(.*\\)\\*" (buffer-name))
      (match-string 1 (buffer-name))
    (error "Cannot find bible chapter"))
  )

(defun org-bible-export (link description format _)
  "Export a bible link from Org files."
  (let ((latex (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 "\\textbv{%s}{}" link))
      (`texinfo (format "@uref{%s,%s}" path desc))
      (`ascii (format "%s (%s)" desc path))
      (_ path))))

(defun get-corresponding-element (element list1 list2)
  "Return the corresponding element in LIST2 for ELEMENT in LIST1."
  (let ((index (cl-position element list1 :test 'equal)))  ; Find the index of
  ↪  ELEMENT in LIST1
    (if index
        (nth index list2)  ; Return the corresponding element from LIST2
      nil)))  ; Return nil if ELEMENT is not found



(defun org-bible-insert ()
  "Insert a link to a bible chapter and verse."
  (interactive)

  (let* ((bb (bible-minor-mode--default-bible-binary))
         (pair (bible-select-internal bb))
         (book-long (car pair))
         (cv (car (cdr pair)))
         (book-short (get-corresponding-element book-long (bible-books-long bb)
         ↪  (bible-books-short bb))))
    (insert (format "[[bible:%s %s][%s %s]]" book-short cv book-long cv))))

(provide 'ol-bible)
```

4.39 Workarounds

```
(load-module 'workarounds)
```

4.39.1 Requirements

```
; Temporary workaround, remove at emacs 29
(package! transient
      :recipe (:host github :repo "magit/transient"))

(package! with-editor
          :recipe (:host github :repo "magit/with-editor"))
```

```
(unpin! magit)

(package! diff-hl :disable t)
```

4.39.2 Code

```
;(global-diff-hl-mode -1)
;(remove-hook 'dired-mode-hook '+vc-gutter-enable-maybe-h)
(provide 'workarounds)
```