1 My Doom Emacs Configuration

1.1 Global Settings

```
(message "> Initializing Emacs")
(setq user-full-name         "Christian Romney"
      user-mail-address      "christian.a.romney@gmail.com"
      calendar-location-name "Pembroke Pines, FL"
      ;; for sunrise, sunset, phases of moon
      calendar-latitude      26.0
      calendar-longitude     -80.3)

(setq doom-scratch-initial-major-mode 'lisp-interaction-mode
      initial-major-mode              'lisp-interaction-mode
      inhibit-startup-message         t)

(setq confirm-kill-emacs              nil
      use-short-answers               t
      enable-dir-local-variables      t
      enable-local-variables          t
      kill-buffer-query-functions     (remq 'process-kill-buffer-query-function
                                            kill-buffer-query-functions))

(setq native-comp-async-report-warnings-errors 'silent)
(setq shell-file-name (executable-find "fish"))
(setq-default vterm-shell "/opt/homebrew/bin/fish")
(setq-default explicit-shell-file-name "/opt/homebrew/bin/fish")
(message "> configuring: ")
```

1.2 Custom Functions

These helpers are used by the configuration that follows. Most functions have to do with file and directory handling and parsing. Others are for wrangling whitespace.

```
(message "  ...custom functions...")

(require 'cl-lib)
(require 'cl-seq)

(defun instrument (fnsym msg)
```

```
  "Instrument the FNSYM function by logging MSG."
  (advice-add fnsym :before (lambda (&rest args) (message msg))))

(defun rk/get-ffmpeg-device ()
  "Gets the list of devices available to ffmpeg.
The output of the ffmpeg command is pretty messy, e.g.
  [AVFoundation indev @ 0x7f867f004580] AVFoundation video devices:
  [AVFoundation indev @ 0x7f867f004580] [0] FaceTime HD Camera (Built-in)
  [AVFoundation indev @ 0x7f867f004580] AVFoundation audio devices:
  [AVFoundation indev @ 0x7f867f004580] [0] Cam Link 4K
  [AVFoundation indev @ 0x7f867f004580] [1] MacBook Pro Microphone
so we need to parse it to get the list of devices.
The return value contains two lists, one for video devices and one for audio devices.
Each list contains a list of cons cells, where the car is the device number and the cdr
  (unless (string-equal system-type "darwin")
    (error "This function is currently only supported on macOS"))

  (let ((lines (string-split (shell-command-to-string "ffmpeg -list_devices true -f avf
    (cl-loop with at-video-devices = nil
             with at-audio-devices = nil
             with video-devices = nil
             with audio-devices = nil
             for line in lines
             when (string-match "AVFoundation video devices:" line)
             do (setq at-video-devices t
                      at-audio-devices nil)
             when (string-match "AVFoundation audio devices:" line)
             do (setq at-audio-devices t
                      at-video-devices nil)
             when (and at-video-devices
                       (string-match "\\[\\([0-9]+\\)\\] \\(.+\\)" line))
             do (push (cons (string-to-number (match-string 1 line)) (match-string 2 li
             when (and at-audio-devices
                       (string-match "\\[\\([0-9]+\\)\\] \\(.+\\)" line))
             do (push (cons (string-to-number (match-string 1 line)) (match-string 2 li
             finally return (list (nreverse video-devices) (nreverse audio-devices))))))

(defun rk/find-device-matching (string type)
  "Get the devices from `rk/get-ffmpeg-device' and look for a device
matching `STRING'. `TYPE' can be :video or :audio."
```

```
  (let* ((devices (rk/get-ffmpeg-device))
         (device-list (if (eq type :video)
                          (car devices)
                        (cadr devices))))
    (cl-loop for device in device-list
             when (string-match-p string (cdr device))
             return (car device))))

(defcustom rk/default-audio-device nil
  "The default audio device to use for whisper.el and outher audio processes."
  :type 'string)

(defun rk/select-default-audio-device (&optional device-name)
  "Interactively select an audio device to use for whisper.el and other audio processes
If `DEVICE-NAME' is provided, it will be used instead of prompting the user."
  (interactive)
  (let* ((audio-devices (cadr (rk/get-ffmpeg-device)))
         (indexes (mapcar #'car audio-devices))
         (names (mapcar #'cdr audio-devices))
         (name (or device-name (completing-read "Select audio device: " names nil t))))
    (setq rk/default-audio-device (rk/find-device-matching name :audio))
    (when (boundp 'whisper--ffmpeg-input-device)
      (setq whisper--ffmpeg-input-device (format ":%s" rk/default-audio-device)))))

(defun cr/list-microphones ()
  "Gets the system audio devices"
  ;; rk/get-ffmpeg-device returns video then audio alists
  (cl-second (rk/get-ffmpeg-device)))

(defun cr/re-find-microphone (regex)
  (cl-rassoc regex (cr/list-microphones)
             :test (lambda (pred s) (string-match-p pred s))))

(defun cr/microphone-name (device)
  "Extracts the label from the microphone DEVICE pair (index . label)."
  (cdr device))

(defun cr/mkdirp (path)
  "Ensures the directory path exists, creating any parents as
needed. Returns the expanded pathname."
```

```
  (let ((abspath (expand-file-name path)))
    (if (file-exists-p abspath)
        abspath
      (progn
        (make-directory abspath 'parents)
        abspath))))

(defun cr/touch (path)
  "Ensures the file path exists, creating any parents as needed.
Returns the expanded pathname."
  (let ((abspath (expand-file-name path)))
    (if (file-exists-p abspath)
        abspath
      (progn
        (make-empty-file abspath 'parents)
        abspath))))

(defun cr/read-file-as-string (path)
  "Reads the given file as a string."
  (string-trim
   (with-temp-buffer
     (insert-file-contents (expand-file-name path))
     (buffer-string))))

(defun cr/keychain-api-token-for-host (host)
  "Reads the keychain internet password for the given host."
  (string-trim
   (shell-command-to-string
    (string-join `("security find-internet-password -s " ,host " -w") ""))))

(defun cr/port-open-p (port)
  "Returns t if the given port is in use, nil otherwise."
  (= 0 (call-process "lsof" nil nil nil "-P" "-i"
                     (concat "TCP:" (number-to-string port)))))

(defun cr/read-auth-field (field &rest params)
  (require 'auth-source)
  (let ((match (car (apply #'auth-source-search params))))
    (if match
        (let ((secret (plist-get match field)))
```

```
          (if (functionp secret)
              (funcall secret)
            secret))
      (error "%s not found for %S" field params))))

(defun cr/read-auth-username (&rest params)
  (apply #'cr/read-auth-field :user params))

(defun cr/read-auth-password (&rest params)
  (apply #'cr/read-auth-field :secret params))

(defun cr/just-one-space ()
  "Command to delete all but one whitespace character."
  (interactive)
  (just-one-space -1))

(defun cr/delete-horizontal-space ()
  "Command to delete all whitespace. Depends on smartparens, which
Doom loads early."
  (interactive)
  (just-one-space -1)
  (sp-backward-delete-char))
```

1.3 Appearance

```
(message "  ...appearance...")
(setq default-frame-alist
  '((fullscreen . maximized)))

(setq display-line-numbers-type   nil
  doom-theme                  'romney-light
  doom-variable-pitch-font    (font-spec :family "Metropolis" :size 18)
  doom-font                   (font-spec :family "MonaspiceNe Nerd Font Mono" :size 20)
  doom-serif-font             (font-spec :family "Times New Roman" :size 20)
  doom-themes-enable-bold     t
  doom-themes-enable-italic   t
  doom-themes-padded-modeline t)

(setq-default tab-width 2)
(setq-default cursor-type 'bar)
```

```
(menu-bar-mode -1)
(tool-bar-mode -1)
(scroll-bar-mode -1)
(pixel-scroll-precision-mode t)
(add-hook 'prog-mode-hook #'rainbow-delimiters-mode)
(add-hook 'prog-mode-hook #'rainbow-mode)
```

1.4 Doom-Specific Settings

```
(message "  ...Doom customizations...")
(doom-themes-visual-bell-config)

(add-to-list 'doom-large-file-size-alist
             '("\\.\\(?:clj[sc]?\\|dtm\\|edn\\)\\'" . 0.5))

;; file locations
(setq doom-cache-dir user-emacs-directory)
(setq +default-want-RET-continue-comments nil)
(setq +file-templates-dir (cr/mkdirp (expand-file-name "snippets" doom-private-dir)))
(setq yas--default-user-snippets-dir +file-templates-dir)
```

1.5 Built-In Modes and Packages

1.5.1 Abbrev Mode

Enable abbreviations. Keep my abbreviations file in my source-controlled Doom directory.

```
(message "  ...built-ins...")
(setq abbrev-file-name (expand-file-name  "etc/abbrev_defs" doom-private-dir)
      save-abbrevs     'silent)

(setq-default abbrev-mode t)
```

1.5.2 Auto-Save Mode

Automatically save org-mode files after 5 seconds of inactivity.

```
(use-package auto-save-mode
  :hook (org-mode . auto-save-visited-mode)
  :init
```

```
  (setq auto-save-visited-interval 5)) ;; seconds
```

1.5.3 Bookmarks

Save file locations.

```
(setq bookmark-default-file     (expand-file-name "etc/bookmarks" doom-private-dir)
      bookmark-old-default-file bookmark-default-file
      bookmark-file             bookmark-default-file
      bookmark-sort-flag        t)
```

1.5.4 Dired

These settings are optimized for Mac OS with the Homebrew version of the GNU `ls` utility. I also like the keybindings for navigating up and opening Finder.app.

```
(after! dired
  (add-hook 'dired-mode-hook #'diredfl-mode)
  (map!
   :map dired-mode-map
   "C-l" #'dired-up-directory)
  (when IS-MAC
    (setq insert-directory-program "gls"
          dired-listing-switches   "-aBhl --group-directories-first")
    (map!
     :map dired-mode-map
     "r"  #'+macos/reveal-in-finder)))
```

1.6 Completion

The combination of company-mode with the modern suite of Vertico, Orderless, Consult, Embark and Marginalia is really well-behaved.

```
(message "  ...completion...")
(when (modulep! :completion vertico)
  (use-package! vertico
    :demand t
    :defer t
    :bind
    (("C-x B"    . #'+vertico/switch-workspace-buffer)
```

```
     :map vertico-map
     ("C-l"      . #'vertico-directory-up)) ;; behave like helm to go up a level
    :config
    (setq vertico-cycle t
          read-extended-command-predicate #'command-completion-default-include-p
          orderless-matching-styles     '(orderless-literal
                                          orderless-initialism
                                          orderless-regexp)
          completion-category-defaults  '((email (styles substring)))
          completion-category-overrides '((file (styles orderless
                                                 partial-completion)))

          marginalia-align              'right))

  (use-package! consult
    :defer t
    :config
    (setq consult-grep-args
          "ggrep --null --line-buffered --color=never --ignore-case \
--exclude-dir=.git --line-number -I -r .")
    :bind
    (("M-g g"   . #'consult-goto-line)
     ("M-i"     . #'consult-imenu)
     ("C-c M-o" . #'consult-multi-occur)
     ("C-x b"   . #'consult-buffer)
     ("C-x 4 b" . #'consult-buffer-other-window)
     ("C-x 5 b" . #'consult-buffer-other-frame)
     ("C-c s r" . #'consult-ripgrep)
     ("C-c s g" . #'consult-git-grep)
     ("C-x r b" . #'consult-bookmark)
     ("C-x r i" . #'consult-register-load)
     ("C-x r s" . #'consult-register-store)
     ("C-h P"   . #'describe-package)
     ("C-h W"   . #'consult-man)))

  (use-package! embark
    :defer t
    :bind
    (("C-." . embark-act)         ;; pick some comfortable binding
     ("M-." . embark-dwim)        ;; good alternative: M-.
```

```
    ) ;; alternative for ‘describe-bindings’
   :init
   ;; Replace the key help with a completing-read interface
   (setq prefix-help-command #’embark-prefix-help-command)
   :config
   ;; Hide the modeline of the Embark live/completions buffers
   (add-to-list ’display-buffer-alist
                ’("\\‘\\*Embark Collect \\(Live\\|Completions\\)\\*"
                  nil
                  (window-parameters (mode-line-format . none)))))

 (defun cr/org-link-qrencode (url)
   "Display a QR code for URL in a buffer. Taken from Sacha Chua’s config."
   (let ((buf (save-window-excursion (qrencode--encode-to-buffer url))))
     (display-buffer-in-side-window buf ’((side . right)))))

 (use-package! qrencode
   :after (embark)
   :config
   (map!
    (:map embark-org-link-map
     :desc "QR encode stored link" "q" #’cr/org-link-qrencode)))

 ;; Consult users will also want the embark-consult package.
 (use-package! embark-consult
   :defer t
   :after (embark consult)
   :demand t ; only necessary if you have the hook below
   ;; if you want to have consult previews as you move around an
   ;; auto-updating embark collect buffer
   :hook
   (embark-collect-mode . consult-preview-at-point-mode)))

(when (modulep! :completion company)
 (use-package! company
   :defer t
   :config
   (setq company-idle-delay 0.5)))
```

1.7 Navigation

I like repeated searches to remain in the middle of the screen so I don't have to scan my monitor for the place where I've landed. I can always stare at the center of the screen and find my search results. With pulsar I can recenter after jumps and highlight the search term.

```
(message "  ...navigation...")
(use-package! pulsar
  :defer t
  :after consult
  :init
  (setq pulsar-pulse t
        pulsar-delay 0.065
        pulsar-iterations 9
        pulsar-face 'pulsar-yellow
        pulsar-highlight-face 'pulsar-red)
  (pulsar-global-mode t)
  :config
  ;; integration with the `consult' package:
  (add-hook 'consult-after-jump-hook #'pulsar-recenter-middle)
  (add-hook 'consult-after-jump-hook #'pulsar-reveal-entry)

  ;; integration with the built-in `isearch':
  (add-hook 'isearch-mode-end-hook #'pulsar-recenter-middle)
  (advice-add 'isearch-forward :after #'pulsar-recenter-middle)
  (advice-add 'isearch-repeat-forward :after #'pulsar-recenter-middle)
  (advice-add 'isearch-backward :after #'pulsar-recenter-middle)
  (advice-add 'isearch-repeat-backward :after #'pulsar-recenter-middle)

  ;; integration with C-v / M-v page scrolling
  (advice-add 'scroll-up-command :after #'pulsar-recenter-middle)
  (advice-add 'scroll-down-command :after #'pulsar-recenter-middle)

  ;; integration with the built-in `imenu':
  (add-hook 'imenu-after-jump-hook #'pulsar-recenter-middle)
  (add-hook 'imenu-after-jump-hook #'pulsar-reveal-entry))
```

1.8 Spell Checking

Ensure custom spelling dictionaries are source controlled.

```
(when (modulep! :checkers spell)
  (message "  ...spell checking...")
  (setq spell-fu-directory
        (cr/mkdirp (expand-file-name "etc/spell-fu/" doom-private-dir)))
  (add-hook 'spell-fu-mode-hook
            (lambda ()
              (spell-fu-dictionary-add (spell-fu-get-ispell-dictionary "en"))
              (spell-fu-dictionary-add
               (spell-fu-get-personal-dictionary
                "en-personal"
                (expand-file-name "aspell.en.pws" spell-fu-directory))))))
```

1.9 Org Mode

1.9.1 Files and Directories

Set up all directory and file paths.

```
;; source directories
(defvar +code-dir "~/src/"
  "Root for source code")

(defvar +foss-dir (cr/mkdirp (expand-file-name "open" +code-dir))
  "Root for open source")

;; main directory
(defvar +docs-dir "~/Documents/"
  "Root for all documents")

(defvar +personal-dir (expand-file-name "personal" +docs-dir)
  "Location of my personal documents")
(defvar +info-dir (expand-file-name "notes" +personal-dir)
  "The root for all notes, calendars, agendas, todos, attachments, and bibliographies.

(defvar +papers-dir (expand-file-name "academic-papers" +info-dir)
  "Location of academic papers downloaded by BibDesk")
```

```
(setq org-directory      (expand-file-name "content" +info-dir)
  org-clock-persist-file (expand-file-name "org-clock-save.el" org-directory)
  +papers-notes-dir      (expand-file-name "papers" org-directory)
  org-download-image-dir (expand-file-name "image-downloads" org-directory)) ;; +dragnd

;; roam notes
(setq org-roam-directory     (expand-file-name "roam" org-directory)
  org-roam-dailies-directory "journal/"
  org-roam-db-location       (expand-file-name ".org-roam.db" org-directory ))

;; AI library
(setq +kb-dir (expand-file-name "kb" org-roam-directory))
(setq +prompts-dir (cr/mkdirp (expand-file-name "prompts" +kb-dir)))
(setq +context-dir (cr/mkdirp (expand-file-name "context" +kb-dir)))

;; agenda
(setq org-agenda-file-regexp "\\`[^.].*\\.org\\(\\.gpg\\)?\\'"
  org-agenda-files           (directory-files-recursively org-directory "\\.org$"))

(after! org
  (add-hook 'org-agenda-mode-hook
    (lambda ()
      (setq org-agenda-files
        (directory-files-recursively org-directory "\\.org$")))))

;; capture
(setq +org-capture-changelog-file "changelog.org"
  +org-capture-notes-file     "notes.org"
  +org-capture-projects-file  "projects.org"
  +org-capture-todo-file      "todo.org"
  +org-capture-journal-file   "journal.org")


(message "  ...org directories and files...")
```

1.9.2 Markup Functions

These commands let me markup org words quickly.

```
(defun cr/markup-word (markup-char)
  "Wraps the active region or the word at point with MARKUP-CHAR."
  (cl-destructuring-bind (text start end)
      (if (use-region-p)
          (list
           (buffer-substring-no-properties (region-beginning) (region-end))
           (region-beginning)
           (region-end))
        (let ((bounds (bounds-of-thing-at-point 'word)))
          (list (thing-at-point 'word)
                (car bounds)
                (cdr bounds))))
    (save-excursion
      (replace-region-contents
       start end
       (lambda ()
         (s-wrap text
                 (char-to-string markup-char)
                 (char-to-string markup-char)))))))

(defun cr/org-italicize-word ()
  (interactive)
  (cr/markup-word #x00002F))

(defun cr/org-bold-word ()
  (interactive)
  (cr/markup-word #x00002A))

(defun cr/org-code-word ()
  (interactive)
  (cr/markup-word #x00007E))

(defun cr/org-underline-word ()
  (interactive)
  (cr/markup-word #x00005F))

(defun cr/org-verbatim-word ()
  (interactive)
  (cr/markup-word #x00003D))
```

```
(defun cr/org-strike-word ()
  (interactive)
  (cr/markup-word #x00002B))

(message "  ...org custom markup functions...")
```

1.9.3 Core Settings

Basic org-mode configuration and startup behavior. Configuration for agenda, capture, appearance, tags, todos, and refiling.

```
;; which modules to load when org starts
;; org-habit
;; org-eval
;; org-expiry
;; org-interactive-query
;; org-collector
;; org-panel
(setq org-modules
  '(ol-bibtex
     ol-bookmark
     org-checklist
     ol-docview
     ol-doi
     org-expiry
     org-id
     org-tempo))

(after! org
  ;; startup configuration
  (setq org-startup-with-inline-images t
    org-startup-with-latex-preview t
    org-M-RET-may-split-line       t)

  ;; behaviors
  (setq
    org-export-html-postamble          nil
    org-export-with-footnotes          t
    org-export-with-latex              t
    org-export-with-smart-quotes       nil
```

```
  org-export-with-planning           nil
  org-export-with-toc                2
  org-hide-emphasis-markers          t
  org-html-validation-link           nil
  org-log-done                       nil
  org-outline-path-complete-in-steps nil
  org-return-follows-link            t
  org-use-fast-todo-selection        t
  org-src-window-setup               'current-window
  org-preview-latex-default-process  'dvipng ; 'dvisvgm
  org-use-sub-superscripts           "{}")

;; agenda
(setq
  org-agenda-tags-column            0
  org-agenda-block-separator        ?
  org-agenda-window-setup           'current-window
  org-agenda-include-diary          t
  org-agenda-show-log               t
  org-agenda-skip-deadline-if-done  t
  org-agenda-skip-scheduled-if-done t
  org-agenda-skip-timestamp-if-done t
  org-agenda-start-on-weekday       1
  org-agenda-todo-ignore-deadlines  t
  org-agenda-todo-ignore-scheduled  t
  org-agenda-use-tag-inheritance    nil
  org-agenda-custom-commands
  '(("d" "Dashboard"
      ((agenda "" ((org-agenda-span 10)))
        (tags-todo "+PRIORITY=\"A\"")
        (tags-todo "work")
        (tags-todo "personal")))
     ("n" "Agenda and all TODOs"
       ((agenda "" ((org-agenda-span 10)))
         (alltodo "")))))
  org-agenda-time-grid
  '((daily today require-timed)
     (800 1000 1200 1400 1600 1800 2000)
     "  " "")
  org-agenda-current-time-string
```

```
    " now ")

  ;; refiling
  (setq
    org-refile-use-cache                   t ;; use C-0 C-c C-w to clear cache
    org-refile-use-outline-path            t
    org-refile-allow-creating-parent-nodes t
    org-refile-targets                     '((nil :maxlevel . 5)
                                             (org-agenda-files :maxlevel . 5)))
  ;; capture
  (setq
    org-capture-templates
    `(("t" "Todo" entry (file+headline "todo.org" "Todos")
        "* TODO %^{Task} %^G")))

  ;; todos
  (setq
    org-todo-keywords
    '((sequence "TODO(t)" "WIP(w)" "PAUSE(p)" "|" "DONE(d)" "KILL(k)" "ASSIGNED(a)")))

  ;; roam
  (add-to-list 'display-buffer-alist
    '("\\*org-roam\\*"
       (display-buffer-in-side-window)
       (side . right)
       (slot . 0)
       (window-width . 0.33)
       (window-parameters . ((no-other-window . t)
                             (no-delete-other-windows . t)))))

(setq org-tag-alist
      '(;; Top Level
        (:startgroup . "Primary")
        ("Business" . ?b)
        ("Personal" . ?p)
        ("Tech" . ?t)
        ("Thinking" . ?k)
        (:endgroup)

        ;; Tech Sub-tags
```

```
      (:startgroup . "Tech")
      ("AI" . ?a)
      ("Clojure" . ?c)
      ("Data" . ?d)
      ("Systems" . ?s)
      ("Security" . ?x)
      ("Design" . ?d)
      (:endgroup)

      ;; Business Sub-tags
      (:startgroup . "Business")
      ("Finance" . ?f)
      ("Leadership" . ?l)
      ("Product" . ?p)
      (:endgroup)

      ;; Personal Sub-tags
      (:startgroup . "Personal")
      ("Cooking" . ?c)
      ("Sailing" . ?s)
      (:endgroup)

      ;; Thinking Sub-tags
      (:startgroup . "Thinking")
      ("ProblemSolving" . ?p)
      ("Learning" . ?l)
      (:endgroup)
      ))

;; visual appearance
(setq
  org-ellipsis                    "»"
  org-fontify-done-headline          t
  org-fontify-emphasized-text        t
  org-fontify-quote-and-verse-blocks t
  org-fontify-whole-heading-line     t
  org-pretty-entities                t
  org-hide-emphasis-markers          t
  org-src-fontify-natively           t
  org-src-tab-acts-natively          t
```

```
    org-auto-align-tags                nil
    org-tags-column                    0
    org-catch-invisible-edits          'show-and-error
    org-special-ctrl-a/e               t
    org-insert-heading-respect-content t
    org-startup-folded                 t
    org-startup-indented               t)

  ;; keybindings
  (map!
    (:map org-mode-map
      :desc "org markup"
      :prefix ("C-, o" . "org markup word")
      :desc "bold"             "b" #'cr/org-bold-word
      :desc "code"             "c" #'cr/org-code-word
      :desc "italics"          "i" #'cr/org-italicize-word
      :desc "strikethrough"    "s" #'cr/org-strike-word
      :desc "underline"        "u" #'cr/org-underline-word
      :desc "verbatim"         "v" #'cr/org-verbatim-word

      )))
(message "  ...org startup, bindings, agenda, tags, todos...")
```

1.9.4 Org Roam

```
(use-package! org-roam
  :after org
  :config
  (setq org-roam-db-autosync-mode 1))

(defun patch/emacsql-close (connection &rest args)
  "Prevent calling emacsql-close if connection handle is nil."
  (when (oref connection handle)
    t))

(advice-add 'emacsql-close :before-while #'patch/emacsql-close)
(message "  ...org-roam...")
```

1.9.5 Modern Appearance

Make org mode more aesthetically pleasing.

```
(use-package! org-modern
  :hook (org-mode . org-modern-mode)
  :config
  (setq org-modern-star 'replace)
  (setq org-modern-block-fringe 4)
  (setq org-modern-replace-stars
    '("" "" "" "" "" "" "" "" "" ""))
  (setq org-modern-keyword
    '(("options" .  "")
       ("title" . "")
       (t . t))))

(use-package! org-side-tree
  :defer t
  :config
  (setq
    org-side-tree-display-side 'right
    org-side-tree-persistent t ;; re-use a single buffer
    org-side-tree-enable-folding t
    org-side-tree-fontify t))

(map! :desc "Tree" "C-c t t" #'org-side-tree)

(message "  ...org appearance...")
```

1.9.6 Calendar

Calendar preferences include holidays, week start, and geographical location.

```
(defface +calendar-holiday
  '((t . (:foreground "#8fb236")))
  "Face for holidays in calendar.")

(defface +calendar-today
  '((t . (:foreground "#e07875" :slant italic)))
  "Face for the current day in calendar.")
```

```
(defface +calendar-appointment
  '((t . (:foreground "white"  :background "#9d7cc7")))
  "Face for appointment diary entries in calendar.")

(defface +calendar-weekend-header
  '((t . (:foreground "#eb9250")))
  "Face for calender weekend days of the week")

(after! org
  (require 'brazilian-holidays)
  (setq calendar-week-start-day              0
    calendar-mark-holidays-flag          t
    calendar-mark-diary-entries-flag     t
    calendar-christian-all-holidays-flag nil
    calendar-holiday-marker              '+calendar-holiday
    calendar-today-marker                '+calendar-today
    calendar-weekend-header              '+calendar-weekend-header
    diary-entry-marker                   '+calendar-appointment
    cal-html-directory                   "~/Desktop"
    cal-html-holidays                    t
    diary-file
    (expand-file-name "appointment-diary" org-directory)

    calendar-holidays
    (append holiday-general-holidays
      holiday-local-holidays
      holiday-other-holidays
      holiday-christian-holidays
      holiday-solar-holidays
      brazilian-holidays--general-holidays
      brazilian-holidays-sp-holidays))
  (add-hook 'calendar-today-visible-hook #'calendar-mark-today))
(message "...org calendar...")
```

1.9.7 Glossary

The org-glossary package adds terms to a top-level `Glossary` heading and expands the definition in the minibuffer whenever the cursor is over a glossary term.

```
(use-package! org-glossary
  :defer t
  :hook (org-mode . org-glossary-mode)
  :init
  (defface org-glossary-term
    '((default :foreground "black" :background "#e8b15c"
       :weight normal))
    "Base face used for term references.")
  :config
  (setq org-glossary-fontify-types-differently nil)
  (map!
    (:map org-mode-map
      :prefix ("C-c y" . "glossary")
      :desc "define term"      "d" #'org-glossary-create-definition
      :desc "goto definition"  "g" #'org-glossary-goto-term-definition
      :desc "insert reference" "i" #'org-glossary-insert-term-reference)))

(message "  ...org glossary...")
```

1.9.8 Citations

Bibliography management and citation embedding via with Citar and Zotero (primarily for computer science paper references from my notes).

```
(use-package! citar
  :after org
  :if (modulep! :tools biblio)
  :config
  (let ((bib (list (expand-file-name "bibliography.bib" +info-dir)))
        (lib-path (list +papers-dir))
        (notes-path +papers-notes-dir))
    (setq!
      org-cite-global-bibliography bib
      reftex-default-bibliography bib
      bibtex-completion-bibliography bib
      bibtex-completion-library-path lib-path
      bibtex-completion-notes-path notes-path
      citar-bibliography bib
      citar-file-variable "Local-Url"
      citar-library-file-extensions (list "pdf")
      citar-library-paths lib-path
```

```
      citar-notes-paths (list notes-path)
      citar-notes-source 'citar-file
      citar-file-open-functions
      (list
        '("pdf"  . citar-file-open-external) ;; use preview
        '("html" . citar-file-open-external)
        '(t      . find-file))))
  (citar-capf-setup)
  (map! :map general-override-mode-map
    "C-c n b" #'citar-open))

(after! citar
  (citar-org-roam-mode -1)
  (setq! citar-indicators
    (list
      (citar-indicator-create
        :symbol (nerd-icons-faicon
                  "nf-fa-file_pdf_o"
                  :face 'nerd-icons-red)
        :function #'citar-has-files
        :padding " "
        :tag "has:files")
      (citar-indicator-create
        :symbol (nerd-icons-codicon
                  "nf-cod-link"
                  :face 'nerd-icons-cyan)
        :function #'citar-has-links
        :padding " "
        :tag "has:links")
      (citar-indicator-create
        :symbol (nerd-icons-codicon
                  "nf-cod-note"
                  :face 'nerd-icons-green)
        :function #'citar-has-notes
        :padding " "
        :tag "has:notes")
      (citar-indicator-create
        :symbol (nerd-icons-codicon
                  "nf-cod-references"
                  :face 'nerd-icons-yellow)
```

```
        :function #'citar-is-cited
        :padding "  "
        :tag "is:cited")))
  (setq! citar-templates
    '((main . "${author editor:10%sn} ${date year issued:4} ${title:64}")
      (suffix . "  ${=key= id:20}  ${=type=:8} ${tags keywords keywords:*}")
      (preview . "${author editor:%etal} (${year issued date}) ${title}, ${journal jou
      (note . "Notes on ${author editor:%etal}, ${title}"))))

(message "  ...org citations, citar...")
```

1.9.9 Literate Programming (org-babel)

Org-mode's Babel feature allows mixing of prose and language blocks (this configuration file is a prime example) for literate programming. Tangling exports code blocks into separate files which can be compiled or interpreted by the relevant program.

I find Graphviz and Plant UML useful for creating diagrams to supplement my notes. I enable all the languages I am likely to use. Auto-tangling keeps tangled code files in sync on save.

```
(use-package! graphviz-dot-mode
  :defer t
  :config
  (setq graphviz-dot-indent-width 2))

(use-package! mermaid-mode
  :defer t
  :config
  (setq ob-mermaid-cli-path "/opt/homebrew/bin/mmdc"))

(after! org
  (when (modulep! :lang plantuml)
    (setq plantuml-default-exec-mode 'jar))

  (org-babel-do-load-languages
   'org-babel-load-languages
   '((clojure    . t)
     (css        . t)
     (dot        . t)
```

```
    (emacs-lisp . t)
    (gnuplot    . t)
    (java       . t)
    (js         . t)
    (makefile   . t)
    (mermaid    . t)
    (plantuml   . t)
    (prolog     . t)
    (python     . t)
    (R          . t)
    (ruby       . t)
    (scheme     . t)
    (sed        . t)
    (shell      . t)
    (sql        . t))))

(message "  ...org babel...")
```

1.9.10 Export Settings

I most often export my org notes to PDF or org-re-reveal HTML presentation.

```
(after! org
  (setq reveal_inter_presentation_links    t
        org-re-reveal-center               t
        org-re-reveal-control              t
        org-re-reveal-default-frag-style   'appear
        org-re-reveal-defaulttiming        nil
        org-re-reveal-fragmentinurl        t
        org-re-reveal-history              nil
        org-re-reveal-hlevel               2
        org-re-reveal-keyboard             t
        org-re-reveal-klipsify-src         t
        org-re-reveal-mousewheel           nil
        org-re-reveal-overview             t
        org-re-reveal-pdfseparatefragments nil
        org-re-reveal-progress             t
        org-re-reveal-rolling-links        nil
        org-re-reveal-title-slide          "%t"
        org-re-reveal-root
```

```
        "https://cdnjs.cloudflare.com/ajax/libs/reveal.js/4.5.0/reveal.js"))

(message "  ...org reveal...")
```

1.10 Artificial Intelligence

Dedicated LLM modes inside Emacs. Proprietary flagship AIs require API keys in `~/.authinfo.gpg`:

```
machine api.openai.com login apikey password sk-secret-openai-api-key-goes-here
```

1.10.1 Core

Commonly specified variables for use across various packages.

```
(defvar gpt-default-model "gpt-4.1-nano-2025-04-14"
  "My preferred Open AI chat model.")

(defvar gpt-default-embedding "text-embedding-3-small"
  "My preferred Open AI embedding model.")

(defvar llm-local-chat-model "gemma3n"
  "Default local model to use for chat.")

(defvar llm-local-embedding-model "mxbai-embed-large"
  "Default local model to use for embeddings.")
```

1.10.2 Chatbot Packages

gpt.el is a general purpose LLM client. It supports local and remote models, tool use, and MCP servers.

```
(when (modulep! :tools llm)
  (use-package! gptel
    :defer t
    :hook
    ((gptel-mode . turn-off-auto-fill)
      (gptel-mode . visual-line-fill-column-mode)
      (gptel-post-stream . gptel-auto-scroll))
    :bind (("C-c m s"    . gptel-send)
           ("C-c m g"   . gptel)
```

```
        ("C-c m r"   . gptel-rewrite)
        ("C-c m a"   . gptel-add)
        ("C-c m k"   . gptel-abort)
        ("C-c m f"   . gptel-add-file)
        ("C-c m t"   . gptel-tools)
        ("C-c m m"   . gptel-menu)
        ("C-c m q"   . gptel-quick)
        ("C-c m p"   . gptel-system-prompt)
        ("C-c m o t" . gptel-org-set-topic)
        ("C-c m o p" . gptel-org-set-properties))
:config
(require 'gptel-integrations)

(gptel-make-preset 'research
  :description "Preset for research tasks"
  :backend "Ollama"
  :model 'phi4-mini
  :tools '("fetch" "get_current_time" "convert_time")
  :temperature 0.7
  :use-context 'system)

(setq
  gptel-model 'qwen3:latest
  gptel-default-mode 'org-mode
  gptel-track-media t
  gptel-use-header-line t
  gptel-org-branching-context t
  gptel-include-reasoning "*gptel-inner-monologue*"
  gptel-prompt-prefix-alist
  '((markdown-mode . "# ")
     (org-mode . "*Prompt*: ")
     (text-mode . "# "))
  gptel-backend
  (gptel-make-ollama "Ollama"
    :host "localhost:11434"
    :stream t
    :models '(aya:latest
              deepcoder:latest
              deepseek-r1:latest
              devstral:latest
```

```
                    gemma3:12b
                    gemma3n:latest
                    llama3.2:latest
                    magistral:latest
                    phi4-mini:latest
                    phi4-reasoning:plus
                    qwen2.5-coder:latest
                    qwen3:latest)))
    (gptel-make-anthropic "Claude" :stream t)
    (gptel-make-gemini "Gemini" :stream t))

  (use-package! gptel-prompts
    :after (gptel)
    :config
    (setq gptel-prompts-directory +prompts-dir)
    (gptel-prompts-update)
    ;; Ensure prompts are updated if prompt files change
    (gptel-prompts-add-update-watchers)))
```

MCP support requires the mcp.el package.

```
(when (modulep! :tools llm)
  (use-package! mcp
    :defer t
    :bind (("C-c m m" . mcp-hub))
    :after gptel
    :custom
    (mcp-hub-servers
      `(("basic-memory" . (:command "uvx" :args ("basic-memory" "mcp")))
        ("context7"     . (:command "npx" :args ("-y" "@upstash/context7-mcp")))
        ("fetch"        . (:command "uvx" :args ("mcp-server-fetch")))
        ("filesystem"   . (:command "npx"
                            :args ("-y" "@modelcontextprotocol/server-filesystem" ,org
        ("playwright"   . (:command "npx" :args ("@playwright/mcp@latest")))
        ("time"         . (:command "uvx" :args ("mcp-server-time")))))
    :config
    (require 'mcp-hub)
    (advice-add 'save-buffers-kill-terminal :before #'mcp-hub-close-all-server)
    :hook (gptel-mode . mcp-hub-start-all-server)))
```

1.10.3 Coding with Aider EXPERIMENTAL

Uses aider and aidermacs for AI-assisted development using local LLMs via Ollama. My aider configuration in `$HOME/.aider.conf.yml` instructs aider to read a `CONVENTIONS.md` for coding conventions and other instructions.

Cogito requires the text "Enable deep thinking subroutine." to be part of the system prompt in order to unlock deeper reasoning. aider's FAQ says to use CONVENTIONS.md to load context for the model.

```
(use-package! aidermacs
  :bind (("C-*" . aidermacs-transient-menu))
  :init
  ;; I prefer local LLMs
  (setenv "OLLAMA_API_BASE" "http://127.0.0.1:11434")
  :config
  (set-popup-rule! "\\*aidermacs.*\\*" :side 'bottom :size 12)
  (require 'aidermacs-backend-vterm)
  (setq aidermacs-backend 'vterm)
  :custom
  (aidermacs-use-architect-mode t)
  ;; for basic question answering
  (aidermacs-default-model "ollama_chat/deepseek-r1:latest")
  ;; for "deeper reasoning"
  (aidermacs-architect-model "ollama_chat/magistral:latest")
  ;; for code changes
  (aidermacs-editor-model "ollama_chat/devstral:latest")
  ;; for commit messages
  (aidermacs-weak-model "ollama_chat/gemma3n:latest"))
```

1.10.4 Coding with Claude Code EXPERIMENTAL

```
;; https://github.com/stevemolitor/claude-code.el
(use-package! claude-code
  :defer t
  :bind-keymap
  ("C-c d" . claude-code-command-map)
  :config
  (setq claude-code-terminal-backend 'vterm))
```

1.11 Speech

1.11.1 Text-To-Speech (TTS)

Greader sends buffer text to a speech engine, like Mac's native speech utility (`say`). The Siri (Voice 4) voice is the most natural default option on the Mac. It's also possible to use your "Personal Voice" with the (say) command by running the `authorize_terminal` command from iTerm. That same shell command can be executed from Emacs with `M-!` to authorize Emacs to use the Personal Voice as well.

```
;; TTS
(use-package! greader
  :defer t
  :custom
  (greader-current-backend (if IS-MAC'greader-mac 'greader-espeak))
  (greader-mac-voice "Christian")
  :config
  (message "  ...greader..."))

(map! :desc "Greader TTS" "C-c 0" #'greader-mode)
```

1.11.2 Speech-To-Text (STT)

For speech-to-text to work, Emacs needs access to the microphone. emacs-plus has merged my PR to enable it by default. If you're using a different version of Emacs for Mac OS, update Emacs' Info.plist manually:

```
<key>NSMicrophoneUsageDescription</key>
<string>Emacs needs permission to access the microphone.</string>
```

Whisper uses the open-source whisper.cpp from Open AI to convert speech to text.

```
(use-package! whisper
  :defer t
  :commands (whisper-run)
  :config
  (setq whisper-install-directory
    (cr/mkdirp (expand-file-name "whisper" doom-cache-dir))
    whisper-model "small"
    whisper-language "en"
```

```
    whisper-translate nil)
  (when IS-MAC
    (let ((mic (cr/microphone-name
                (cl-some #'identity
                  (list (cr/re-find-microphone "rode")
                    (cr/re-find-microphone "mac"))))))
      (message (format " using microphone: %s" mic))
      (rk/select-default-audio-device mic))

    (when rk/default-audio-device
      (setq whisper--ffmpeg-input-device (format ":%s" rk/default-audio-device))))
  (message "  ...whisper..."))

(map! :desc "Whisper" "C-s-\\" #'whisper-run)
```

1.12 Programming Modes

Configuration for additional programming modes.

1.12.1 Indentation

Always 2 spaces for every language I use.

```
(let ((n 2))
  (setq standard-indent n
    python-indent-offset n
    lisp-indent-offset n
    fish-indent-offset n ;; some autoformatter on save is not respecting this
    smie-indent-basic n
    sh-indentation n
    markdown-list-indent-width n))
```

1.12.2 Paren Matching

Highlight and blink matching parentheses.

```
(setq blink-matching-paren t
      show-paren-mode t
      show-paren-style 'parenthesis
      show-paren-delay 0)
```

1.12.3 Smartparens

Smartparens doesn't play nicely with org-mode. This is one of the places where Doom is uncharacteristically heavy-handed with its defaults. I remove the global hook and enable smartparens (strict mode) where I want it, especially in Lisp buffers. I also don't like smartparens' default rules.

```
(pcase-dolist (`(,open . ,close) '(("(" . ")")
                                   ("[" . "]")
                                   ("{" . "}")))
    ;; remove all default rules
    (sp-pair open close :post-handlers nil :unless nil)
    ;; add sole exception
    (sp-pair open close :unless '(:add sp-in-string-p)))

(remove-hook! 'doom-first-buffer-hook #'smartparens-global-mode)
(add-hook! 'doom-first-buffer-hook #'smartparens-global-strict-mode)

(message "  ...smartparens...")
```

1.12.4 Diff / Merge

Configure ediff to have better defaults

```
(use-package! ediff
  :defer t
  :config
  (setq ediff-split-window-function 'split-window-horizontally
        ediff-window-setup-function 'ediff-setup-windows-plain)
  (setq ediff-keep-variants nil
        ediff-make-buffers-readonly-at-startup nil
        ediff-merge-revisions-with-ancestor t
        ediff-show-clashes-only t))
```

1.12.5 Projects

Have projectile save things where I want them.

```
(after! projectile
  (cr/mkdirp (expand-file-name "projectile" doom-cache-dir))
```

```
  (setq projectile-cache-file
        (expand-file-name "projectile/projectile.cache" doom-cache-dir)
        projectile-known-projects-file
        (expand-file-name "projectile/projectile.projects" doom-cache-dir)
        projectile-project-search-path '("~/src/"))

  (pushnew! projectile-project-root-files "project.clj" "deps.edn"))

(message "  ...projectile...")
```

1.12.6 Git

I use source control for everything, and enjoy a few extras for Magit. Also, Doom dropped the `gist` tool, so I grab it directly from Github.

```
(use-package! magit
  :bind ("C-x g" . magit-status)
  :custom
  (magit-git-executable "/opt/homebrew/bin/git"))

(after! magit
  (setq magit-revision-show-gravatars t
    forge-database-file
    (expand-file-name "forge/forge-database.sqlite" doom-cache-dir)
    magit-no-confirm '(stage-all-changes unstage-all-changes)))

(use-package igist
  :bind (("M-G" . igist-dispatch))
  :config
  (setq igist-auth-marker 'igist))

(message "  ...magit...")
```

1.12.7 Python

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

1.12.8 Clojure

Something weird is going on with org-mode

1. Clojure mode w/ LSP

```
(use-package! clojure-mode
  :defer t
  :hook ((clojure-mode . rainbow-delimiters-mode)
         (clojure-mode . subword-mode))
  :config
  (setq cider-enable-nrepl-jvmti-agent t
        cider-enrich-classpath t)
  (when (modulep! :tools lsp)
    (map! :map clojure-mode-map
      "C-c j u d"    #'lsp-ui-doc-glance
      "C-c j u m"    #'lsp-ui-imenu)
    (after! lsp-clojure
      (dolist (dir '("[/\\\\]\\.clj-kondo\\'"
                     "[/\\\\]\\.cp-cache\\'"
                     "[/\\\\]\\.lsp\\'"
                     "[/\\\\]\\.shadow-cljs\\'"
                     "[/\\\\]\\target\\'"))
        (add-to-list 'lsp-file-watch-ignored dir)))
    (setq
      lsp-lens-enable          t       ;; enable LSP code lens for inline referenc
      lsp-file-watch-threshold 2000
      lsp-enable-snippet       t)))

(message "  ...clojure editing...")
```

1.13 YouTube Playlists

I want to be able to find videos from my YouTube playlists and to add new ones.

```
;; (cr/keychain-api-token-for-host "www.googleapis.com")
```

1.14 Elfeed

Feed reader for emacs.

```
(use-package! elfeed
  :defer t
```

```
  :config
  (setq-default
    elfeed-search-filter "#50 @1-week-ago +unread "
    elfeed-save-multiple-enclosures-without-asking t
    elfeed-search-clipboard-type 'CLIPBOARD
    elfeed-search-filter "+unread "
    elfeed-search-date-format '("%Y-%m-%d" 10 :left) ;;'("%b %d" 6 :left)
    elfeed-search-title-min-width 45)
  (setq elfeed-feeds
    '(("https://news.ycombinator.com/news" tech)
       ("https://planet.emacslife.com/atom.xml" tech emacs)
       ( "https://simonwillison.net/atom/everything/" tech ai)
       ("https://huggingface.co/blog/feed.xml" tech ai))))

(map!
  :desc "Elfeed"        "C-x F v" #'elfeed
  :desc "Elfeed Update" "C-x F u" #'elfeed-update)

(use-package! elfeed-tube
  :after elfeed
  :defer t
  :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))
  :config
  (setq
    elfeed-tube-use-ytdlp-p       t       ;; use yt-dlp
    elfeed-tube-auto-save-p       nil     ;; enable auto-save
    elfeed-tube-captions-sblock-p t ;; diminish sponsorship text
    elfeed-tube-save-indicator    t
    elfeed-tube-thumbnail-size    'medium
    elfeed-log-level              'debug
    elfeed-tube-captions-languages
    '("en" "english" "english (auto generated)"))
  (add-hook 'elfeed-new-entry-hook
    (elfeed-make-tagger :feed-url "youtube\\.com"
      :add '(video youtube)))
```

```
  (elfeed-tube-setup))

;; (use-package! elfeed-tube-mpv
;;   :after elfeed-tube)

(map! :map elfeed-show-mode-map
  "C-c C-f" #'elfeed-tube-mpv-follow-mode
  "C-c C-w" #'elfeed-tube-mpv-where)
```

1.15 Miscellaneous

Every Emacs configuration contains a few little odds and ends.

```
(add-to-list 'auto-mode-alist (cons "\\.adoc\\'" 'adoc-mode))
```

1.16 Global Key Bindings

My global keybinding preferences.

```
(message "  ...global keybindings...")
(map!
  "<s-left>"  #'sp-forward-barf-sexp
  "<s-right>" #'sp-forward-slurp-sexp
  "C-$"       #'+spell/add-word
  "C-'"       #'avy-goto-line
  "C-:"       #'avy-goto-char
  "C-M-%"     #'anzu-query-replace-regexp
  "C-c M-t"   #'transpose-sentences
  "C-c a"     #'org-agenda
  "C-c g"     #'google-this
  "C-e"       #'move-end-of-line
  "C-x M-s"   #'transpose-sexps
  "C-x M-t"   #'transpose-paragraphs
  "C-x P"     #'print-buffer
  "C-x k"     #'doom/save-and-kill-buffer
  "C-x \\"    #'align-regexp
  "C-x g"     #'magit-status
  "C-x r I"   #'string-insert-rectangle
  "M-%"       #'anzu-query-replace
  "M-/"       #'hippie-expand
  "M-SPC"     #'cr/just-one-space
```

```
  "M-\\"      #'cr/delete-horizontal-space
  "M-o"       #'other-window
  "M-p"       #'fill-paragraph
  "C-,"       #'browse-url)
```

1.17 Conclusion

If this message appears in the `*Messages*` buffer, then all configuration loaded successfully.

```
(message "> Emacs initialization complete.")
```

1.17.1 Doom Config Instructions

Whenever you reconfigure a package, make sure to wrap your config in an 'after!' block, otherwise Doom's defaults may override your settings. E.g.

```
(after! PACKAGE
  (setq x y))
```

The exceptions to this rule:

- Setting file/directory variables (like 'org-directory')
- Setting variables which explicitly tell you to set them before their package is loaded (see 'C-h v VARIABLE' to look up their documentation).
- Setting doom variables (which start with 'doom-' or '+').

Here are some additional functions/macros that will 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 and hit 'C-c c k'.

This will open documentation for it, including demos of how they are used. Alternatively, use 'C-h o' to look up a symbol (functions, variables, faces, etc).

You can also try 'C-c c d' to jump to their definition and see how they are implemented.* My Doom Emacs Configuration