My Literate Emacs Configuration

Daniel Pinkston

December 27, 2025

Contents

1 Startup

1.1 Early initialization

The `early-init.el` removes all the visual fluff like scrollbars and sets up very fundamental Emacs settings that need to be loaded before everything else.

```
(setq inhibit-startup-message t)
(setq inhibit-startup-screen t)
(setq frame-resize-pixelwise t
      frame-inhibit-implied-resize t
      use-dialog-box t ; only for mouse events, which I seldom use
      use-file-dialog nil
      use-short-answers t
      inhibit-x-resources t
      inhibit-startup-echo-area-message user-login-name
      inhibit-startup-buffer-menu t)
(setq mode-line-misc-info
      (delete (assoc 'minor-mode-alist mode-line-misc-info)
   mode-line-misc-info))
(add-hook 'after-init-hook (lambda () (set-frame-name "home")))
;; Modes
(scroll-bar-mode -1)
(tool-bar-mode -1)
(tooltip-mode -1)
(set-fringe-mode 10)
(menu-bar-mode -1)
(tool-bar-mode 0)
;; settings for windows
(setq focus-follows-mouse t)
```

```
(add-to-list 'default-frame-alist '(alpha . (95 . 95)))
;; garbage collection
(setq gc-cons-threshold most-positive-fixnum
      gc-cons-percentage 0.5)
(defvar bard-emacs--file-name-handler-alist file-name-handler-alist)
(defvar bard-emacs--vc-handled-backends vc-handled-backends)
(add-hook 'emacs-startup-hook
          (lambda ()
            (setq gc-cons-threshold (* 1000 1000 8)
                  gc-cons-percentage 0.1
                  file-name-handler-alist
   bard-emacs--file-name-handler-alist
                  vc-handled-backends
   bard-emacs--vc-handled-backends)))
;; Package cache
(setq package-enable-at-startup t)
```

1.2 Initialization

```
;;; init.el --- init.el -*- lexical-binding: t -*-
;; Author: BardofSprites
;; Maintainer: BardofSprites
;; Version: 0.1.0
;; Package-Requires: ((emacs "28.2"))
;; This file is not part of GNU Emacs
;; 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:
;;; Code:
```

```
;; fix keymap
(repeat-mode t)
(global-set-key (kbd "C-z") nil)
(global-set-key (kbd "M-z") nil)
(global-set-key (kbd "C-z C-z") #'suspend-emacs)
;; Add the directories to the load path
(mapc
 (lambda (string)
   (add-to-list 'load-path (locate-user-emacs-file string)))
 '("bard-elisp" "bard-emacs-modules" "old-ada"))
;; Clipboard saving
(setq select-enable-clipboard t)
;; custom file is evil
(setq custom-file (make-temp-file "emacs-custom-"))
;; No Backups
(setq auto-save-default nil)
(setq make-backup-files nil)
(setq confirm-kill-emacs 'y-or-n-p)
;; native comp enabled
(when (native-comp-available-p)
  (setq native-compile-prune-cache t))
;; enable/disable commands
(mapc
 (lambda (command)
   (put command 'disabled nil))
 '(list-timers narrow-to-region narrow-to-page upcase-region
   downcase-region))
(mapc
 (lambda (command)
   (put command 'disabled t))
 '(eshell project-eshell overwrite-mode iconify-frame diary))
;;; Packages
(setq package-archives '(("elpa" . "https://elpa.gnu.org/packages/")
                         ("nongnu" . "https://elpa.nongnu.org/nongnu
   /")
                         ("melpa" . "https://melpa.org/packages/")))
;;; MACROS
(defmacro bard/make-abbrev (table &rest definitions)
  "Expand abbrev DEFINITIONS for the given TABLE.
DEFINITIONS is a sequence of (i) string pairs mapping the
abbreviation to its expansion or (ii) a string and symbol pair
making an abbreviation to a function."
  (declare (indent 1))
  (unless (zerop (% (length definitions) 2))
    (error "Uneven number of key+command pairs"))
  `(if (abbrev-table-p ,table)
       (progn
         ,@(mapcar
            (lambda (pair)
              (let ((abbrev (nth 0 pair))
                    (expansion (nth 1 pair)))
                (if (stringp expansion)
                    `(define-abbrev ,table ,abbrev ,expansion)
                  `(define-abbrev ,table ,abbrev "" ,expansion))))
            (seq-split definitions 2)))
     (error "%s is not an abbrev table" ,table)))
(require 'bard-emacs-anki)
(require 'bard-emacs-calendar)
(require 'bard-emacs-completion)
(require 'bard-emacs-dired)
;; (require 'bard-emacs-email)
(require 'bard-emacs-media)
(require 'bard-emacs-eshell)
(require 'bard-emacs-essentials)
;; (require 'bard-emacs-keyboard)
(require 'bard-emacs-modeline)
(require 'bard-emacs-org)
(require 'bard-emacs-prog)
(require 'bard-emacs-theme)
(require 'bard-emacs-ui)
(require 'bard-emacs-web)
(require 'bard-emacs-window)
(require 'bard-emacs-writing)
(provide 'init)
(put 'eshell 'disabled nil)
```

2 Modules

2.1 bard-emacs-anki

Anki is a flashcard program that helps you spend more time on challenging material, and less on what you already know.

This is an emacs package that allows me to write flashcards in org mode. I talked about it in my EmacsConf 2024 presentation titled: A example of a cohesive student workflow in Emacs.

```
(use-package anki-editor
  :ensure t
  :after org
  :bind (:map org-mode-map
              ("C-c M-i" . bard/anki-editor-cloze-region-auto-incr)
              ("C-c M-I" . bard/anki-editor-cloze-region-dont-incr)
              ("C-c M-r" . bard/anki-editor-reset-cloze-number)
              ("C-c M-p" . anki-editor-push-new-notes)
              ("C-c M-P" . anki-editor-push-notes))
  :hook (org-capture-after-finalize . bard/
   anki-editor-reset-cloze-number) ; Reset cloze-number after each
   capture.
  :config
  (setq anki-editor-create-decks t ;; Allow anki-editor to create a
   new deck if it doesn't exist
        anki-editor-org-tags-as-anki-tags t)
  (defun bard/anki-editor-cloze-region-auto-incr (&optional arg)
    "Cloze region without hint and increase card number."
    (interactive)
    (anki-editor-cloze-region my-anki-editor-cloze-number "")
    (setq my-anki-editor-cloze-number (1+
   my-anki-editor-cloze-number))
    (forward-sexp))
  (defun bard/anki-editor-cloze-region-dont-incr (&optional arg)
    "Cloze region without hint using the previous card number."
    (interactive)
    (anki-editor-cloze-region (1- my-anki-editor-cloze-number) "")
    (forward-sexp))
  (defun bard/anki-editor-reset-cloze-number (&optional arg)
    "Reset cloze number to ARG or 1"
    (interactive)
    (setq my-anki-editor-cloze-number (or arg 1)))
  ;; Initialize
  (bard/anki-editor-reset-cloze-number)
  )
(provide 'bard-emacs-anki)
```

2.2 bard-emacs-calendar

2.2.1 Load required libraries

```
(require 'bard-calendar)
(require 'hi-lock)
```

2.2.2 Org directory and agenda files

```
(setq org-directory "~/Notes/denote/")
;; symlinked file to shorten denote file name in agenda buffers
(setq org-agenda-files (list "~/Notes/denote/todo.org" "~/Notes/
   denote/uni.org"))
```

2.2.3 Org Calendar

```
;; Calendar
(use-package calendar-mode
  :config
  (setq calendar-holidays (append calendar-holidays russian-holidays
   ))
  :hook
  (calendar-today-visible . calendar-mark-today)
  (calendar-mode . denote-journal-calendar-mode)
  (calendar-today-visible . calendar-mark-holidays))
```

1. Orthodox Christian Holidays

```
(use-package orthodox-christian-new-calendar-holidays
  :ensure t
  :config
  (setq holiday-other-holidays (append holiday-other-holidays
   orthodox-christian-new-calendar-holidays))
  (setq holiday-bahai-holidays nil
        holiday-christian-holidays nil
        holiday-islamic-holidays nil))
```

2.2.4 Org todo keywords

```
;; Org todo keywords - changed to using hl-todo faces fixed by modus
   /ef themes
(setq org-todo-keywords
      '((sequence "TODO(t)" "EXTRA(e)" "INPROG(i)" "|" "DONE(d)" "
   KILLED(k)")
        (sequence "MEET(m)" "TENT(T)" "|" "MET(M)" "NOGO(n)")))
(setq org-todo-keyword-faces
      '(("EXTRA" . (:inherit warning))
        ("TENT" . (:inherit warning))
        ("MEET" . (:inherit warning underline bold))
        ("INPROG" . (:inherit hi-yellow :weight bold))))
(setq org-enforce-todo-dependencies t)
;; Automatically clock in
(add-hook 'org-after-todo-state-change-hook #'bard/auto-clock-in)
```

2.2.5 Org clocking

```
;; clock tables
(setq org-clock-clocktable-default-properties '(:maxlevel 7 :scope
   agenda)
      org-clock-persist 'history
      org-clock-mode-line-total 'current)
(org-clock-persistence-insinuate)
(use-package org
  ;; not really show what this does anymore
  :demand t
  :hook
  ((org-clock-out . bard/org-clock-update-mode-line)))
```

2.2.6 Org agenda settings

```
(global-set-key (kbd "<f1>") 'bard/default-agenda)
(global-set-key (kbd "M-<f1>") 'bard/choose-agenda)
(setq org-agenda-include-diary t)
(setq org-agenda-custom-commands
      `(("D" "Daily agenda and top priority tasks"
         ((tags-todo "!TODO"
                     ((org-agenda-overriding-header "Unscheduled
   Tasks \n")
                      (org-agenda-skip-function '(
   org-agenda-skip-entry-if 'timestamp))))
          (agenda "" ((org-agenda-span 1)
                      (org-agenda-start-day nil)
                      (org-deadline-warning-days 14)
```

```
                    ;; (org-scheduled-past-days 0)
                    (org-agenda-day-face-function (lambda (date) '
    org-agenda-date))
                    (org-agenda-format-date "%A %-e %B %Y")
                    (org-agenda-overriding-header "Today's agenda
    \n")))
          (agenda "" ((org-agenda-span 8)
                    (org-calendar-holiday)
                    (org-deadline-warning-days 0)
                    (org-agenda-skip-function '(
    org-agenda-skip-entry-if 'todo 'done))
                    (org-agenda-overriding-header "Upcoming this
    week \n")))))
        ("Y" "Yearly view for all tasks"
         ((agenda "" ((org-agenda-span 365)
                    (org-deadline-warning-days 2)
                    (org-agenda-skip-function '(
    org-agenda-skip-entry-if 'todo 'done))
                    (org-agenda-overriding-header "Upcoming this
    Year\n")))))
        ("M" "Monthly view for all tasks"
         ((agenda "" ((org-agenda-span 31)
                    (org-deadline-warning-days 2)
                    (org-agenda-skip-function '(
    org-agenda-skip-entry-if 'todo 'done))
                    (org-agenda-overriding-header "Upcoming this
    month\n")))))))
```

2.2.7 Org habit

```
(use-package org-habit
  :after org-agenda
  :config
  (setq org-habit-show-done-always-green t
        org-habit-show-habits t
        org-habit-show-all-today t))
```

2.2.8 Provide module

```
(provide 'bard-emacs-calendar)
```

2.3 bard-emacs-completion

2.3.1 Vertico completion framework

```
;; minibuffer completion
(use-package vertico
  :ensure t
  :config
  (setq vertico-scroll-margin 0)
  (setq vertico-cycle nil)
  (setq vertico-resize nil)
  (vertico-mode t)
  (with-eval-after-load 'rfn-eshadow
    ;; This works with `file-name-shadow-mode' enabled.  When you
   are in
    ;; a sub-directory and use, say, `find-file' to go to your home
   '~/'
    ;; or root '/' directory, Vertico will clear the old path to
   keep
    ;; only your current input.
    (add-hook 'rfn-eshadow-update-overlay-hook #'
   vertico-directory-tidy)))
```

2.3.2 Rfn eshadow

```
(use-package rfn-eshadow
  :ensure nil
  :hook (minibuffer-setup . cursor-intangible-mode)
  :config
  (setq resize-mini-windows t)
  (setq read-answer-short t) ; also check `use-short-answers' for
   Emacs28
  (setq echo-keystrokes 0.25)
  ;; Do not allow the cursor to move inside the minibuffer prompt.
   I
  ;; got this from the documentation of Daniel Mendler's Vertico
  ;; package: <https://github.com/minad/vertico>.
  (setq minibuffer-prompt-properties
        '(read-only t cursor-intangible t face minibuffer-prompt))
  ;; MCT has a variant of this built-in.
  (defun crm-indicator (args)
    (cons (format "[`completing-read-multiple': %s]  %s"
                  (propertize
                   (replace-regexp-in-string
                    "\\`\\[.*?]\\*\\|\\[.*?]\\*\\'" ""
                    crm-separator)
                   'face 'error)
                  (car args))
          (cdr args)))
  (advice-add #'completing-read-multiple :filter-args #'
   crm-indicator)
  (file-name-shadow-mode 1))
```

2.3.3 Marginalia

```
(use-package marginalia
  :ensure t
  :init
  (marginalia-mode 1))
```

2.3.4 Orderless

```
(use-package orderless
  :ensure t
  :config
  (setq completion-styles '(orderless basic)))
```

2.3.5 Tab completion

For a long time I really struggled with Emacs tab completion. It still only kind of works most of the time, but I don't want to tweak it to make it work. It works really well with LSP and using Lisp REPL's for languages like Clojure, Common Lisp, and Scheme. However for some languages like Haskell, without the LSP (which hasn't worked for me in the past), it does not work to the extent that I believe it should. Completion is something I'm kind of jealous of when it comes to Neovim vs Emacs.

1. UI for tab completion

   ```
   (use-package corfu
     :ensure t
     :init (global-corfu-mode)
     ;; I also have (setq tab-always-indent 'complete) for TAB to
      complete
   ```

```
  ;; when it does not need to perform an indentation change.
  :bind (:map corfu-map ("<tab>" . corfu-complete))
  :config
  (setq corfu-preview-current nil)
  (setq corfu-min-width 20)
  (setq corfu-popupinfo-delay nil)
  (corfu-popupinfo-mode 1) ; shows documentation after `
   corfu-popupinfo-delay'
  ;; Sort by input history (no need to modify `
   corfu-sort-function').
  (with-eval-after-load 'savehist
    (corfu-history-mode 1)
    (add-to-list 'savehist-additional-variables 'corfu-history))
   )
```

2. Compilation at point functions (CAPF)

```
(use-package cape
  :ensure t
  :init
  (add-to-list 'completion-at-point-functions #'cape-file)
  (add-to-list 'completion-at-point-functions #'cape-dabbrev)
  (add-to-list 'completion-at-point-functions #'cape-keyword)
  (add-to-list 'completion-at-point-functions #'cape-abbrev))
(use-package clang-capf
  :ensure t
  :config
  (defun bard/clang-capf-init ()
    "Add `clang-capf' to `completion-at-point-functions'."
    (add-hook 'completion-at-point-functions #'clang-capf nil t)
   )
  (add-hook 'c-mode-hook #'bard/clang-capf-init))
```

3. Completion styles

```
(use-package minibuffer
  :config
;;;; Completion styles
  (setq completion-styles '(basic substring initials flex
   orderless))
  (setq completion-category-defaults nil)
  (setq completion-category-overrides
        '((file (styles . (basic partial-completion orderless)))
          (bookmark (styles . (basic substring)))
          (library (styles . (basic substring)))
          (embark-keybinding (styles . (basic substring)))
          (imenu (styles . (basic substring orderless)))
          (consult-location (styles . (basic substring orderless
   )))
          (kill-ring (styles . (emacs22 orderless)))
          (eglot (styles . (emacs22 substring orderless))))))
```

2.3.6 Minibuffer enhancements

1. Consult

```
(use-package consult
  :ensure t
  :defer 2
  :bind*
  ("C-x r b" . consult-bookmark)
  ("M-g M-g" . consult-goto-line)
  ("C-x b" . consult-buffer)
  ("M-s M-f" . consult-find)
  ("M-s M-g" . consult-grep)
  ("M-s M-h" . consult-history)
  ("M-s M-y" . consult-yank-pop)
  ("M-s M-o" . consult-outline)
  ("M-s M-l" . consult-line)
  ("M-s M-k" . consult-kmacro)
  ("M-s M-r" . consult-register)
  :config
  (setq consult-find-args
        (concat "find . -not ( "
                "-path */.git* -prune "
                "-or -path */.cache* -prune )")))
```

2. Embark

```
(use-package embark
  :ensure t
  :bind*
  (("C-," . bard-embark-act-no-quit)
   ("C-." . bard-embark-act-quit))
  :config
```

```
(require 'bard-embark)
(setq embark-keymap-alist
      '((buffer bard-embark-buffer-map)
        (command bard-embark-command-map)
        (expression bard-embark-expression-map)
        (file bard-embark-file-map)
        (function bard-embark-function-map)
        (identifier bard-embark-identifier-map)
        (package bard-embark-package-map)
        (region bard-embark-region-map)
        (symbol bard-embark-symbol-map)
        (url bard-embark-url-map)
        (variable bard-embark-variable-map)
        (t embark-general-map)))
(defun bard-embark-act-no-quit ()
  "Call `embark-act' but do not quit after the action."
  (interactive)
  (let ((embark-quit-after-action nil))
    (call-interactively #'embark-act)))
(defun bard-embark-act-quit ()
  "Call `embark-act' and quit after the action."
  (interactive)
  (let ((embark-quit-after-action t))
    (call-interactively #'embark-act))
  (when (and (> (minibuffer-depth) 0)
             (derived-mode-p 'completion-list-mode))
    (abort-recursive-edit)))
(setq embark-confirm-act-all nil)
;; The prot-embark.el has an advice to further simplify the
;; minimal indicator.  It shows cycling, which I never want to
  see
;; or do.
(setq embark-mixed-indicator-both nil)
(setq embark-mixed-indicator-delay 0.1)
(setq embark-indicators '(embark-mixed-indicator
 embark-highlight-indicator))
(setq embark-verbose-indicator-nested nil) ; I think I don't
 have them, but I do not want them either
(setq embark-verbose-indicator-buffer-sections '(bindings))
(setq embark-verbose-indicator-excluded-actions
      '(embark-cycle embark-act-all embark-collect
    embark-export embark-insert)))
```

2.3.7 Save minibuffer history (`savehist`)

```
(setq savehist-file (locate-user-emacs-file "savehist"))
(setq history-length 100)
(setq history-delete-duplicates t)
(setq savehist-save-minibuffer-history t)
(setq savehist-additional-variables '(register-alist kill-ring))
(savehist-mode t)
```

2.3.8 Recent files (`recentf`)

```
(recentf-mode t)
(setq recentf-save-file (locate-user-emacs-file "recentf"))
;; save cursor place for code files
(add-hook 'prog-mode-hook #'save-place-local-mode)
```

2.3.9 Abbreviations (`(d)abbrev-mode`)

```
(setq abbrev-file-name (locate-user-emacs-file "abbrevs"))
(setq only-global-abbrevs nil)
(bard/make-abbrev global-abbrev-table
  "meweb" "https://bardman.dev"
  "megit" "https://github.com/BardofSprites"
  "meyt" "https://www.youtube.com/@bardmandev"
  "protweb" "https://protesilaos.com/")
(bard/make-abbrev text-mode-abbrev-table
  "asciidoc"       "AsciiDoc"
  "auctex"         "AUCTeX"
  "cafe"           "écaf"
  "cliche"         "éclich"
  "clojurescript"  "ClojureScript"
  "emacsconf"      "EmacsConf"
  "github"         "GitHub"
  "gitlab"         "GitLab"
  "javascript"     "JavaScript"
  "latex"          "LaTeX"
  "libreplanet"    "LibrePlanet"
  "linkedin"       "LinkedIn"
  "paypal"         "PayPal"
  "sourcehut"       "SourceHut"
  "texmacs"         "TeXmacs"
  "typescript"      "TypeScript"
  "visavis"         "àvis--vis"
  "vscode"          "Visual Studio Code"
  "youtube"         "YouTube"
  "Результат"       "Результат= Сегодняшний Битвый="
  "asf"             "and so on and so forth"
  "paragraph"       "¶"
  "em"              "—"
  "ua"              "↑"
  "da"              "↓"
  "ra"              "→"
  "la"              "←"
  "iff"             "⍰"
  "imp"             "⇒"
  "tf"              "⍰"
  "xmonad"          "XMonad"
  "xmobar"          "XMobar")
(dolist (hook '(text-mode-hook prog-mode-hook git-commit-mode-hook))
  (add-hook hook #'abbrev-mode))
(remove-hook 'save-some-buffers-functions #'abbrev--possibly-save)
```

2.3.10 Provide module

```
(provide 'bard-emacs-completion)
;;; bard-emacs-completion.el ends here
```

2.4 **bard-emacs-dired**

2.4.1 Dired subtree

```
(use-package dired-subtree
  :ensure t
  :config
  (setq dired-subtree-use-backgrounds nil))
```

2.4.2 Dired file preview

```
(use-package dired-preview
  :ensure t
  :bind
  (:map dired-mode-map
        ("P" . dired-preview-mode))
  :config
  (setq dired-preview-delay 0.1))
```

2.4.3 General dired settings

```
(use-package dired
  :bind*
  (("C-j" . dired-jump))
  :bind (:map dired-mode-map
              (("E" . emms-add-dired)
               ("<tab>" . dired-subtree-toggle)
               ("<backtab>" . dired-subtree-cycle)))
  :config
  (setq dired-guess-shell-alist-user ; those are the suggestions for
    ! and & in Dired
        '(("\\.\\(png\\|jpe?g\\|tiff\\)" "nsxiv" "feh" "xdg-open")
          ("\\.\\(mp[34]\\|m4a\\|ogg\\|flac\\|webm\\|mkv\\)" "mpv" "
   xdg-open")
          (".gif" "mpv --loop=inf")
          (".*" "xdg-open")))
  (setq dired-dwim-target t)
  (setq dired-listing-switches
        "-AgGFhlv --group-directories-first --time-style=long-iso")
  :hook
  ((dired-mode . dired-hide-details-mode)
   ;; attachments for email through dired
   (dired-mode . turn-on-gnus-dired-mode)))
```

2.4.4 Image dired

```
(use-package image-dired
  :bind
  (:map dired-mode-map
        ((")" . image-dired-dired-display-external)
         ("B" . bard/dired-set-background-with-feh)))
  :bind
  ("C-x C-d" . image-dired)
  :bind
```

```
(:map image-dired-thumbnail-mode-map ("B" . bard/
 image-dired-set-background-with-feh))
:config
(define-advice image-dired-display-image (:override (file &
 optional _ignored))
  (setq file (expand-file-name file))
  (when (not (file-exists-p file))
    (error "No such file: %s" file))
  (let ((buf (get-buffer image-dired-display-image-buffer))
        (cur-win (selected-window)))
    (when buf
      (kill-buffer buf))
    (when-let ((buf (find-file-noselect file nil t)))
      (with-current-buffer buf
        (rename-buffer image-dired-display-image-buffer)
        (if (string-match (image-file-name-regexp) file)
            (image-dired-image-mode)
          ;; Support visiting PDF files.
          (normal-mode))
        (display-buffer buf))
      (select-window cur-win))))
(setq image-dired-thumbnail-storage 'standard)
(setq image-dired-external-viewer "nsxiv")
(setq image-dired-thumb-size 80)
(setq image-dired-thumb-margin 2)
(setq image-dired-thumb-relief 0)
(setq image-dired-thumbs-per-row 4)
(defun bard/dired-set-background-with-feh ()
  "Set the selected image as the background using feh."
  (interactive)
  (let ((image-file (dired-get-file-for-visit)))
    (start-process "feh" nil "feh" "--bg-fill" image-file)
    (message "Background set to %s" image-file)))
(defun bard/image-dired-set-background-with-feh ()
  "Set the selected image as the background using feh."
  (interactive)
  (let ((image-file (image-dired-original-file-name)))
    (start-process "feh" nil "feh" "--bg-fill" image-file)
    (message "Background set to %s" image-file))))
```

2.4.5 Running commands on dired files

```
;; Taken from https://superuser.com/a/176629
(defun bard/dired-do-command (command)
  "Run COMMAND on marked files. Any files not already open will be
   opened.
After this command has been run, any buffers it's modified will
   remain
open and unsaved."
  (interactive "CRun on marked files M-x ")
  (save-window-excursion
    (mapc (lambda (filename)
            (find-file filename)
            (call-interactively command))
          (dired-get-marked-files))))
```

2.4.6 Dired video thumbnails

```
(use-package dired-video-thumbnail
  :ensure t
  :vc (:url "https://github.com/captainflasmr/dired-video-thumbnail"
            :rev :newest)
  :bind (:map dired-mode-map
              ("C-t v" . dired-video-thumbnail)))
```

2.4.7 Provide module

```
(provide 'bard-emacs-dired)
```

2.5 **bard-emacs-email**

2.5.1 Require email library

```
(require 'bard-email)
```

2.5.2 Notmuch mail client settings

```
(use-package notmuch
  :ensure t
  :config
  (define-key global-map (kbd "C-c m") #'notmuch)
  (setq notmuch-show-logo t
        notmuch-column-control 1.0
        notmuch-hello-auto-refresh t
        notmuch-hello-recent-searches-max 20
        notmuch-hello-thousands-separator ""
        notmuch-hello-sections '(notmuch-hello-insert-header
   notmuch-hello-insert-saved-searches notmuch-hello-insert-search
   notmuch-hello-insert-alltags)
        notmuch-show-all-tags-list t)
  (setq notmuch-search-oldest-first nil)
  (setq notmuch-show-seen-current-message t)
  (defun bard/notmuch-mua-empty-subject-check ()
    "Request confirmation before sending a message with empty
   subject."
    (when (and (null (message-field-value "Subject"))
               (not (y-or-n-p "Subject is empty, send anyway? ")))
      (error "Sending message cancelled: empty subject")))
  (add-hook 'message-send-hook 'bard/notmuch-mua-empty-subject-check
   ))
```

1. Notmuch saved searches

```
(setq notmuch-show-empty-saved-searches t)
(setq notmuch-saved-searches
      `(( :name "⊠ inbox (all-mail)"
          :query "tag:inbox"
          :sort-order newest-first
          :key ,(kbd "i"))
        ( :name "⊠ unread (inbox)"
          :query "tag:unread and tag:inbox"
          :sort-order newest-first
          :key ,(kbd "u"))
        ( :name "⊠ To Do"
          :query "tag:todo"
          :sort-order oldest-first
          :key ,(kbd "t"))
        ( :name "⊠ flagged"
          :query "tag:flag"
          :sort-order newest-first
          :key ,(kbd "f"))
        ( :name "⊠ contributions"
          :query "tag:unread and tag:contrib"
```

```
            :sort-order newest-first
            :key ,(kbd "c"))
          ( :name "⊠ linux-related"
            :query "tag:unread and tag:linux"
            :sort-order newest-first
            :key ,(kbd "l"))
          ( :name "⊠ emacs developement"
            :query "tag:unread and tag:contrib"
            :sort-order newest-first
            :key ,(kbd "ed"))
          ( :name "⊠ emacs humanities"
            :query "tag:unread and tag:emacs-humanities"
            :sort-order newest-first
            :key ,(kbd "eh"))
          ( :name "⊠ emacs org-mode"
            :query "tag:unread and tag:emacs-org"
            :sort-order newest-first
            :key ,(kbd "eo"))))
```

2. Tagging keys

```
(setq notmuch-tagging-keys
      `((,(kbd "d") prot-notmuch-mark-delete-tags "⊠ Mark for
   deletion")
        (,(kbd "f") prot-notmuch-mark-flag-tags "⊠ Flag as
   important")
        (,(kbd "s") prot-notmuch-mark-spam-tags "⊠ Mark as spam"
   )
        (,(kbd "r") ("-unread") "⊠⊠ Mark as read")
        (,(kbd "u") ("+unread") "⊠ Mark as unread")))
(setq notmuch-archive-tags '("+archive")
      notmuch-message-replied-tags '("+replied")
      notmuch-message-forwarded-tags '("+forwarded")
      notmuch-show-mark-read-tags '("-unread")
      notmuch-draft-tags '("+draft")
      notmuch-draft-folder "drafts"
      notmuch-draft-save-plaintext 'ask)
```

2.5.3 Notmuch modeline indicator

```
(use-package notmuch-indicator
  :ensure t
  :after notmuch
  :config
  (setq notmuch-indicator-args
        '(( :terms "tag:unread and tag:inbox"
            :label "[U] "
            :label-face prot-modeline-indicator-green
            :counter-face prot-modeline-indicator-green)
          ( :terms "tag:unread and tag:linux"
            :label "[L] "
            :label-face prot-modeline-indicator-cyan
            :counter-face prot-modeline-indicator-cyan)
          ( :terms "tag:unread and tag:emacs"
            :label "[E] "
            :label-face prot-modeline-indicator-blue
            :counter-face prot-modeline-indicator-blue))
        notmuch-indicator-refresh-count (* 60 3)
        notmuch-indicator-hide-empty-counters t
        notmuch-indicator-force-refresh-commands '(
   notmuch-refresh-this-buffer))
  (setq notmuch-indicator-add-to-mode-line-misc-info nil)
  (notmuch-indicator-mode t))
```

2.5.4 Org links for notmuch

```
(use-package ol-notmuch
  :ensure t
  :after notmuch)
```

2.5.5 Send mail and smtp settings

```
;; use msmtp
(setq sendmail-program "/usr/bin/msmtp"
      message-send-mail-function 'message-send-mail-with-sendmail
      message-sendmail-f-is-evil nil
      mail-specify-envelope-from t
      message-sendmail-envelope-from 'header
      mail-envelope-from t)
```

2.5.6 Provide module

```
(provide 'bard-emacs-email)
```

2.6 bard-emacs-eshell

```
(use-package eshell
  :ensure nil
  :bind
  (("C-z e" . eshell-switcher))
  :config
  (require 'bard-eshell)
  (require 'ffap)
  ;; (setq eshell-banner-message "Time for another recreational
   programming session.\n\n")
  (setq eshell-banner-message
        '(format "%s %s\n %s\n"
                 (propertize (format " %s " (string-trim (
   buffer-name)))
                             'face 'mode-line-highlight)
                 (propertize (current-time-string)
                             'face 'font-lock-keyword-face)
         (propertize "Time for another recreational programming
   session."
                     'face 'warning)))
  (setq bard/eshell-aliases
        '((g   . magit)
          (gl  . magit-log)
          (d   . dired)
          (o   . find-file)
          (oo . find-file-other-window)
          (vim . find-file)
          (l  . (lambda () (eshell/ls '-la)))
          (eshell/clear . eshell/clear-scrollback)))
  (mapc (lambda (alias)
          (defalias (car alias) (cdr alias)))
        bard/eshell-aliases))
(use-package eshell
  :ensure nil
  :after esh-mode
  :bind
  (:map eshell-mode-map
        ("C-c C-e" . prot-eshell-export)
        ("M-k"     . eshell-kill-input)
        ("C-c C-d"   . prot-eshell-complete-recent-dir)
        ("C-c C-h"   . prot-eshell-narrow-output-highlight-regexp)
        ("C-c C-f"   . bard/eshell-find-file-at-point)))
(provide 'bard-emacs-eshell)
```

2.7 bard-emacs-essentials

2.7.1 Writable grep buffers (wgrep)

```
(use-package wgrep
  :ensure t
  :bind
  (:map wgrep-mode-map
        ("C-x C-s" . wgrep-save-all-buffers)
        ("C-x C-q" . wgrep-change-to-wgrep-mode)
        ("C-c C-c" . wgrep-finish-edit))
  :bind
  (:map grep-mode-map
        ("e" . wgrep-change-to-wgrep-mode)))
```

```
(use-package multiple-cursors
  :ensure t
  :config
  (setq mc/always-run-for-all t)
  :bind
  (("C-S-c C-S-c" . mc/edit-lines)
   ("C->" . mc/mark-next-like-this)
   ("C-<" . mc/mark-previous-like-this)
   ("C-c C" . mc/mark-all-like-this)
   ("C-\"". mc/skip-to-next-like-this)
   ("C-;" . mc/skip-to-previous-like-this)))
```

```
(use-package expand-region
  :ensure t
  :bind ("C-=" . er/expand-region))
```

```
(use-package substitute
  :ensure t
  :bind
  (("C-c s b" . substitute-target-below-point)
   ("C-c s a" . substitute-target-above-point)
   ("C-c s d" . substitute-target-in-defun)
   ("C-c s s" . substitute-target-in-buffer)))
```

```
;; Desktop mode/session saving
(setq desktop-path '("~/.emacs.d/desktop")
      desktop-dirname "~/.emacs.d/desktop/"
      desktop-base-file-name "emacs-desktop"
      desktop-save t
      desktop-restore-eager t
      desktop-restore-=frams t
      desktop-restory-in-current-display t
      desktop-files-not-to-save "\(^$\\|\\*scratch\\*\\|\\*Messages
   \\*\\|\\*dashboard\\*\\|\\*Async-native-compile-log\\*|\\*Music
   \\*)")
;; (desktop-save-mode t)
(global-set-key (kbd "C-z s") 'desktop-save-in-desktop-dir)
(global-set-key (kbd "C-z r") 'desktop-read)
```

2.7.2 Scratch buffers

```
;; Text Scratch buffers
(defun bard/new-org-buffer ()
  (interactive)
  (let ((xbuf (generate-new-buffer "*org*")))
    (switch-to-buffer xbuf)
    (funcall (quote org-mode))
    (text-scale-increase 1.5)
    xbuf))
(define-key global-map (kbd "M-=") #'bard/new-org-buffer)
(defun bard/new-plain-buffer ()
  (interactive)
  (let ((xbuf (generate-new-buffer "*plain*")))
    (switch-to-buffer xbuf)
    (text-scale-increase 1.5)
    xbuf))
(define-key global-map (kbd "M--") #'bard/new-plain-buffer)
;; elisp scratch buffer
(defun bard/new-elisp-buffer ()
  (interactive)
  (let ((xbuf (generate-new-buffer "*elisp*")))
    (switch-to-buffer xbuf)
    (funcall (quote emacs-lisp-mode))
    (text-scale-increase 1.5)
    xbuf))
```

2.7.3 Terminals

```
(defun bard/open-terminal-in-current-directory ()
  "Open a terminal in the current working directory."
  (interactive)
  (let ((default-directory default-directory))
    (term "/usr/bin/env bash")))
;; (define-key global-map (kbd "C-t") #'bard/
   open-terminal-in-current-directory)
(define-key global-map (kbd "C-z t") #'bard/
   open-terminal-in-current-directory)
(defun bard/open-terminal-emulator ()
  "Open a terminal in the current working directory."
  (interactive)
  (let ((default-directory default-directory))
    (start-process "st terminal" nil "st")))
(define-key global-map (kbd "C-z C-t") 'bard/open-terminal-emulator)
(define-key global-map (kbd "C-z C-s") #'bard/new-elisp-buffer)
```

2.7.4 Time Management

```
;; Modeline
(setq display-time-format "%Y-%m-%d (%a) %H:%M")
(setq display-time-interval 60)
(setq display-time-default-load-average nil)
(setq display-time-mail-directory nil)
(setq display-time-mail-function nil)
(setq display-time-use-mail-icon nil)
(setq display-time-mail-string nil)
(setq display-time-mail-face nil)
(setq display-time-string-forms
      '((propertize
         (format-time-string display-time-format now)
         'face 'display-time-date-and-time
         'help-echo (format-time-string "%a %b %e, %Y" now))
        " "))
(display-time-mode 1)
;; world clock
```

```
(setq world-clock-list
      '(("America/New_York" "New York")
        ("Europe/Moscow" "Moscow")
        ("Europe/London" "London")
        ("Asia/Tokyo" "Tokyo")))
(setq world-clock-time-format "%Y-%m-%d %B (%A) %R %Z")
;; timer package
(use-package tmr
  :ensure t
  :config
  (setq tmr-notification-urgency 'normal)
  (setq tmr-sound-file nil)
  (setq tmr-timer-finished-functions '(tmr-notification-notify

   tmr-print-message-for-finished-timer
                                       tmr-acknowledge-minibuffer))
  (setq tmr-descriptions-list 'tmr-description-history)
  (define-key global-map (kbd "C-c t l") 'tmr-tabulated-view)
  (define-key global-map (kbd "C-c t t") #'tmr)
  (define-key global-map (kbd "C-c t T") #'tmr-with-description)
  (define-key global-map (kbd "C-c t l") #'tmr-tabulated-view)
  (define-key global-map (kbd "C-c t c") #'tmr-clone)
  (define-key global-map (kbd "C-c t k") #'tmr-cancel)
  (define-key global-map (kbd "C-c t s") #'tmr-reschedule)
  (define-key global-map (kbd "C-c t e") #'tmr-edit-description)
  (define-key global-map (kbd "C-c t r") #'tmr-remove)
  (define-key global-map (kbd "C-c t R") #'tmr-remove-finished))
```

2.7.5 Running emacs as server

```
(require 'server)
(setq server-client-instructions nil)
(unless (server-running-p)
  (server-start))
```

2.7.6 Provide module

```
(provide 'bard-emacs-essentials)
```

2.8 bard-emacs-media

2.8.1 EMMS music player

```
(use-package emms
  :ensure t
  ;; :demand t
  :bind
  (:map emms-playlist-mode-map
        ("A" . emms-add-directory)
        ("l" . emms-add-playlist)
        ("T" . emms-add-directory-tree)
        ("F" . emms-add-file)
        ("U" . emms-add-url)
        ("L" . emms-toggle-repeat-track)
        ("<mouse-3>" . emms-pause)
        ("<SPC>" . emms-pause)
        ("c" . bard/emms-recenter)
        ("P" . emms-playlist-mode-shift-track-up)
        ("N" . emms-playlist-mode-shift-track-down)
        ("Z" . bard/save-emms-watch-later)
        ("Y" . bard/emms-download-current-video))
  :bind (("<f8>" . emms)
         ("M-<f8>" . emms-browser))
  :hook
  (emms-playlist-mode . hl-line-mode)
  :config
  (emms-all)
  (emms-default-players)
  (emms-mpris-enable)
  (setq emms-player-list '(emms-player-mpv))
  ;; emms-info-functions '(emms-info-native)
  ;; (setq emms-browser-covers 'emms-browser-cache-thumbnail)
  (setq emms-volume-amixer-card 0)
  ;; center line function
  (defun bard/emms-recenter ()
    (interactive)
    (recenter-top-bottom)
    (emms-playlist-mode-center-current))
  ;; modeline
  (emms-mode-line-disable)
  (emms-playing-time-disable-display)
  ;; playlist saving
  (setq bard/emms-playlist-format 'm3u)
  (setq bard/watch-later-file "~/Videos/watch-later.m3u")
  )
```

2.8.2 PDF viewer (`pdf-tools`)

```
(use-package pdf-tools
  :ensure t
  :config
  (pdf-tools-install)
  (add-to-list 'pdf-tools-enabled-modes #'pdf-view-themed-minor-mode
   ))
```

2.8.3 Custom functions

```
(use-package bard-media
  :ensure nil
  :config
  (require 'bard-media)
  :bind
  (("C-c o p" . bard/play-youtube-video)
   ("C-c o i" . bard/image-browser))
  )
```

2.8.4 Provide module

```
(provide 'bard-emacs-media)
```

2.9 **bard-emacs-modeline**

```
(require 'bard-modeline)
;;; Mode line
(setq mode-line-compact nil) ; Emacs 28
(setq mode-line-right-align-edge 'right-margin)
(setq-default mode-line-format
              '("%e"
                prot-modeline-kbd-macro
                prot-modeline-narrow
                bard-modeline-centered-cursor
                prot-modeline-input-method
                prot-modeline-buffer-status
                prot-modeline-window-dedicated-status
                bard-evil-state-indicator
                " "
                prot-modeline-buffer-identification
                "  "
                prot-modeline-major-mode
                prot-modeline-process
                "  "
                prot-modeline-vc-branch
                "  "
                prot-modeline-flymake
                prot-modeline-eglot
                "  "
                mode-line-format-right-align
                prot-modeline-notmuch-indicator
                " "
                prot-modeline-misc-info
                " "))
(with-eval-after-load 'spacious-padding
  (defun prot/modeline-spacious-indicators ()
    "Set box attribute to `'prot-modeline-indicator-button' if
   spacious-padding is enabled."
    (if (bound-and-true-p spacious-padding-mode)
        (set-face-attribute 'prot-modeline-indicator-button nil :box
    t)
      (set-face-attribute 'prot-modeline-indicator-button nil :box '
   unspecified)))
  ;; Run it at startup and then afterwards whenever
  ;; `spacious-padding-mode' is toggled on/off.
  (prot/modeline-spacious-indicators)
  (add-hook 'spacious-padding-mode-hook #'prot/
   modeline-spacious-indicators))
(setq mode-line-right-align-edge 'window)
(provide 'bard-emacs-modeline)
;;; bard-emacs-modeline.el ends here
```

2.10 **bard-emacs-org**

2.10.1 Include required libraries

```
(require 'org)
(require 'ox)
(require 'org-habit)
```

2.10.2 Main Configuration

```
(use-package org
  :defer nil
  :bind
  (:map org-mode-map
        ("C-M-a" . backward-paragraph)
        ("C-M-e" . forward-paragraph)
        ("C-c M-c" . count-words-region)
        ("C-c C-M-c" . count-words)
        ("C-c l" . org-id-get-create)
        ("C-c j" . org-goto)
        )
  :bind
  (("C-c c" . org-capture))
  :config
  (setq org-goto-interface 'outline-path-completion)
  (setq org-special-ctrl-a/e t)
  (setq safe-local-variable-values '((org-refile-targets (nil :
   maxlevel . 3)))))
```

```
(setq org-archive-location "~/Notes/denote/20240328
   T215840--archive__self.org::* Archive")
(setq org-log-done 'time)
(setq org-icalendar-include-todo t
      org-icalendar-include-body t
      org-icalendar-with-timestamps t
      org-icalendar-use-scheduled '(event-if-todo-not-done)
      org-icalendar-scheduled-summary-prefix "SCHEDULED: "
      org-icalendar-use-deadline '(event-if-todo-not-done)
      org-icalendar-deadline-summary-prefix "DEADLINE: ")
```

```
(setq org-structure-template-alist
      '(("c" . "center")
        ("x" . "example")
        ("d" . "definition")
        ("t" . "theorem")
        ("q" . "quote")
        ("v" . "verse")
        ("s" . "src")
        ("E" . "src emacs-lisp :results value code :lexical t") ;
   for code examples in notes
        ("z" . "src emacs-lisp :tangle FILENAME") ; tangle without
   making dir, below makes dir
        ("Z" . "src emacs-lisp :tangle FILENAME :mkdirp yes")))
(setq org-ellipsis " ⊠")
```

2.10.3 Making org mode look nice

```
(setq org-startup-indented t
      org-startup-folded 'showeverything
      org-hide-emphasis-markers t
      org-startup-with-inline-images t
      org-image-actual-width '(600)
      org-list-allow-alphabetical t
      org-insert-heading-respect-content t)
(use-package org-bullets
  :ensure t
  :hook (org-mode . org-bullets-mode)
  :config
  (setq org-bullets-bullet-list '("◉" "○" "●" "⊠" "⊠" "◆")))
```

2.10.4 Org export + latex

```
(setq org-format-latex-options (plist-put org-format-latex-options :
   scale 1.5))
(defun bard/org-export-on-save ()
  "Export current Org buffer to PDF and open it with auto-revert
   enabled."
  (when (derived-mode-p 'org-mode)
    (org-latex-export-to-pdf)))
(define-minor-mode bard/org-auto-export-pdf-mode
  "Automatically export Org buffer to PDF on save."
  :lighter " AutoPDF"
  :group 'org
  (if bard/org-auto-export-pdf-mode
      (add-hook 'after-save-hook #'bard/org-export-on-save)
    (remove-hook 'after-save-hook #'bard/org-export-on-save)))
(use-package auctex
  :ensure t)
(use-package cdlatex
  :ensure t
  )
;; latex editing niceness
(use-package org-fragtog
  :ensure t)
(with-eval-after-load 'ox-latex
  (add-to-list 'org-latex-classes
               '("org-plain-latex"
                 "\\documentclass{article}
           [NO-DEFAULT-PACKAGES]
           [PACKAGES]
           [EXTRA]"
                 ("\\section{%s}" . "\\section*{%s}")
                 ("\\subsection{%s}" . "\\subsection*{%s}")
                 ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                 ("\\paragraph{%s}" . "\\paragraph*{%s}")
                 ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
(setq org-latex-listings t)
(setq org-latex-listings-options
      '(("basicstyle" "\\ttfamily")
        ("breakatwhitespace" "false")
        ("breakautoindent" "true")
        ("breaklines" "true")
        ("columns" "[c]fullflexible")
        ("commentstyle" "")
        ("emptylines" "*")
        ("extendedchars" "false")
        ("fancyvrb" "true")
        ("firstnumber" "auto")
        ("flexiblecolumns" "false")
        ("frame" "single")
        ("frameround" "tttt")
        ("identifierstyle" "")
        ("keepspaces" "true")
        ("keywordstyle" "")
        ("mathescape" "false")
        ("numbers" "left")
        ("numbers" "none")
        ("numbersep" "5pt")
        ("numberstyle" "\\tiny")
        ("resetmargins" "false")
        ("showlines" "true")
        ("showspaces" "false")
        ("showstringspaces" "false")
        ("showtabs" "true")
        ("stepnumber" "2")
        ("stringstyle" "")
        ("tab" "⊠")
        ("tabsize" "4")
        ("texcl" "false")
        ("upquote" "false")))
```

2.10.5 Org capture

```
(setq org-capture-bookmark nil
      org-id-link-to-org-use-id t)
(require 'org-protocol)
(setq org-capture-templates
      '(("t" "task" entry
         (file+olp
          "~/Notes/denote/20240328T215727--todo.org"
          "Inbox" "General tasks")
         "* TODO %?")
        ;; ("s" "Basic Statistics" entry
        ;;  (file+headline
        ;;   "~/Notes/denote/20240830
   T215644--statistics-flashcards__anki_stats.org" "Unsorted")
        ;;  "** %U %^g\n:PROPERTIES:\n:ANKI_NOTE_TYPE: Basic\n:
   ANKI_DECK: Statistics\n:END:\n*** Front\n %?\n*** Back\n\n")
        ;; ("S" "Cloze Statistics" entry
        ;;  (file+headline
        ;;   "~/Notes/denote/20240830
   T215644--statistics-flashcards__anki_stats.org" "Unsorted")
        ;;  "** %U %^g\n:PROPERTIES:\n:ANKI_NOTE_TYPE: Cloze\n:
   ANKI_DECK: Statistics\n:END:\n*** Text\n %?\n*** Hooray\n\n")
        ("c" "Basic Chemistry" entry
         (file+headline
          "~/Notes/denote/20251019
   T175402--chemistry-flashcards__anki_chem.org" "Unsorted")
         "** %U %^g\n:PROPERTIES:\n:ANKI_NOTE_TYPE: Basic\n:
   ANKI_DECK: Chemistry\n:END:\n*** Front\n%?\n*** Back\n\n")
        ("n" "common place note" entry
         (file "~/Notes/denote/20251023
   T182240--common-place-notes__topic.org")
         "* %^{Source}\n#+BEGIN_QUOTE\n%?\n#+END_QUOTE")
        ("z" "Protocol" entry
         (file+olp
          "~/Notes/denote/20240328
   T220037--media-tracker__media_topic.org" "Quotes")
         "* Source: [[%:link][%:description]]\n#+BEGIN_QUOTE\n%i\n#+
   END_QUOTE\n%?")
        ("Z" "Protocol Link" entry
         (file+olp
          "~/Notes/denote/20240328
   T220037--media-tracker__media_topic.org" "Watch/Read List")
         "* [[%:link][%:description]] \nCaptured On: %U \n%?")
        ("w" "Class outline" entry
         (file
          "~/Notes/denote/20240328T215727--todo.org")
         (file
          "~/Notes/denote/templates/class-template.org"))
        ("p" "project idea" entry
         (file
          "~/Notes/denote/20250201
   T165619--project-ideas__idea_programming.org")
         "* %^{Project description}\n%?")))
```

2.10.6 Managing media

```
;; copy/paste images
(use-package org-download
  :after org
  :defer nil
  :ensure t
  :custom
  (org-download-method 'directory)
  (org-download-image-dir "~/Notes/denote/Images")
  (org-download-heading-lvl 0)
  (org-download-timestamp "org_%Y%m%d-%H%M%S_")
  (org-download-screenshot-method "xclip -selection clipboard -t
   image/png -o > '%s'")
  :bind
  ("C-M-y" . org-download-screenshot)
  :config
  (require 'org-download))
;; pdf notes
(use-package org-noter
  :ensure t)
;; links
(use-package org-cliplink
  :ensure t
  :bind
  ("C-c p" . org-cliplink))
(provide 'bard-emacs-org)
```

2.11 **bard-emacs-prog**

2.11.1 General input setup

```
(use-package electric
  :hook
  (prog-mode . electric-indent-local-mode)
  (prog-mode . electric-pair-local-mode))
(use-package paren
  :hook (prog-mode . show-paren-local-mode)
  :config
  (setq show-paren-style 'parenthesis)
  (setq show-paren-when-point-in-periphery nil)
  (setq show-paren-when-point-inside-paren nil)
  (setq show-paren-context-when-offscreen 'overlay))
```

2.11.2 Haskell

```
(use-package haskell-mode
  :ensure t
  :config
  (setq haskell-interactive-popup-errors nil))
```

2.11.3 C/C++

```
(use-package c-mode
  :config
  (setq-default c-basic-offset 4)
  (setq c-default-style '((c-mode . "gnu")
                          (java-mode . "java")
                          (awk-mode . "awk"))))
(use-package ggtags
  :ensure t
  :config
  (add-hook 'c-mode-common-hook
            (lambda ()
              (when (derived-mode-p 'c-mode 'c++-mode 'java-mode)
                (ggtags-mode 1)
                (setq-local imenu-create-index-function #'
   ggtags-build-imenu-index)))))
(use-package compile
  :ensure nil
  :defer 2
  :config
  (require 'bard-compile)
  (setq compilation-scroll-output t
        compilation-auto-jump-to-first-error nil))
```

2.11.4 Lisp development

```
(use-package clojure-mode
  :ensure t)
(use-package cider
  :ensure t)
(use-package sly
  :ensure t
  :config
  (setq inferior-lisp-program (executable-find "sbcl")))
(use-package geiser
  :ensure t)
(use-package geiser-gauche
  :ensure t)
;; parens packages
```

2.11.5 Catching errors

```
(use-package flycheck
  :ensure t
  :config
  (global-flycheck-mode t))
```

2.11.6 Version control

```
;; Version control
(use-package magit
  :ensure t
  :config
  (setq magit-repository-directories
        '(("~/Code"                    . 1)
          ("~/Repositories"            . 1)
          ("~/dotfiles-stow"           . 0)
          ("~/.emacs.d"                . 0)
          ("~/Pictures/wallpaper"      . 0)))
  :bind ("C-c g" . magit-status))
```

2.11.7 Ada-mode

```
;; (use-package ada-mode
;;   :load-path "~/.emacs.d/old-ada"
;;   :bind
;;   (:map ada-mode-map
;; ⊠("C-j" . dired-jump)))
```

2.11.8 Provide module

```
(provide 'bard-emacs-prog)
```

2.12 **bard-emacs-theme**

2.12.1 Load required libraries

```
(require 'bard-theme)
```

2.12.2 Set all themes to safe

```
;; declare all themes as safe (i trust developers)
(setq custom-safe-themes t)
```

2.12.3 Color theme

```
(use-package doom-themes
  :ensure t
  :config
  (setq doom-gruvbox-dark-variant "hard")
  (bard/select-theme 'doom-gruvbox))
```

2.12.4 Fonts

```
(use-package fontaine
  :ensure nil
  :config
  ;; save file
  (setq fontaine-latest-state-file
        (locate-user-emacs-file "fontaine-latest-state.eld"))
  ;; Set last preset or fall back to desired style from `
   fontaine-presets'.
  (fontaine-set-preset (or (fontaine-restore-latest-preset) 'default
   ))
  ;; The other side of `fontaine-restore-latest-preset'.
  (add-hook 'kill-emacs-hook #'fontaine-store-latest-preset)
  ;; preserve fonts when switching themes
  (dolist (hook '(modus-themes-after-load-theme-hook
   ef-themes-post-load-hook))
    (add-hook hook #'fontaine-apply-current-preset))
  (define-key global-map (kbd "C-c f") #'fontaine-set-preset))
```

```
(use-package rainbow-mode
  :ensure t)
```

2.12.5 Binding for custom select theme function

```
(global-set-key (kbd "M-<f6>") #'bard/select-theme)
```

2.12.6 Mixed pitch faces for writing

```
(use-package mixed-pitch
  :ensure t
  :hook
  (org-mode . mixed-pitch-mode))
```

2.12.7 Spacious padding

```
(use-package spacious-padding
  :ensure t
  :config
  (setq spacious-padding-widths
        '( :internal-border-width 10
           :header-line-width 4
           :mode-line-width 6
           :tab-width 4
           :right-divider-width 1
           :left-fringe-width 0
           :right-fringe-width 0
           :scroll-bar-width 0))
  (spacious-padding-mode t))
```

2.12.8 Provide module

```
(provide 'bard-emacs-theme)
;;; bard-emacs-theme.el ends here
```

2.13 **bard-emacs-ui**

```
(use-package whitespace
  :ensure nil
  :demand t
  :config
  (setq whitespace-style '(face
                           tabs
                           spaces
                           trailing
                           space-before-tab
                           newline indentation
                           empty space-after-tab
                           space-mark tab-mark))
  :hook
  (prog-mode . whitespace-mode))
(use-package display-line-numbers
  :ensure nil
  :demand t
  :bind
  (("<f12>" . display-line-numbers-mode))
  :hook (prog-mode . display-line-numbers-mode)
  :config
  (setq display-line-numbers-type 'relative))
```

2.13.1 Todo keywords

```
(use-package hl-todo
  :ensure t
  :hook
  (prog-mode . hl-todo-mode)
  :config
  (setq hl-todo-highlight-punctuation ":"))
```

2.13.2 Keycasting

```
(use-package keycast
  :ensure t
  :commands (keycast-mode-line-mode keycast-header-line-mode
   keycast-tab-bar-mode keycast-log-mode)
  :init
  (setq keycast-mode-line-format "%2s%k%c%R")
  (setq keycast-mode-line-insert-after 'prot-modeline-vc-branch)
  (setq keycast-mode-line-window-predicate '
   mode-line-window-selected-p)
  (setq keycast-mode-line-remove-tail-elements nil)
  :config
  (dolist (input '(self-insert-command org-self-insert-command))
    (add-to-list 'keycast-substitute-alist `(,input "." "…Typing")))
  (dolist (event '("<mouse-event>" "<mouse-movement>" "<mouse-2>" "<
   drag-mouse-1>" "<wheel-up>" "<wheel-down>" "<double-wheel-up>" "<
   double-wheel-down>" "<triple-wheel-up>" "<triple-wheel-down>" "<
   wheel-left>" "<wheel-right>" handle-select-window mouse-set-point
     mouse-drag-region))
    (add-to-list 'keycast-substitute-alist `(,event nil nil))))
```

2.13.3 Provide module

```
(provide 'bard-emacs-ui)
;;; bard-emacs-ui.el ends here
```

2.14 bard-emacs-web

2.14.1 IRC

```
;;; IRC
(use-package circe
  :ensure t
  :config
  (setq auth-sources '("~/.authinfo.gpg"))
  (defun my-fetch-password (&rest params)
    (require 'auth-source)
    (let ((match (car (apply 'auth-source-search params))))
      (if match
          (let ((secret (plist-get match :secret)))
            (if (functionp secret)
                (funcall secret)
              secret))
        (error "Password not found for %S" params))))
  (defun my-nickserv-password (server)
    (my-fetch-password :user "bardman" :machine "irc.libera.chat"))
  (setq circe-network-options
        '(("Libera Chat"
           :nick "bardman"
           :channels ("#emacs" "##anime" "#gentoo")
           :nickserv-password my-nickserv-password))))
```

2.14.2 RSS Feeds

```
(use-package elfeed
  :ensure t
  :config
  (require 'bard-web)
  (global-set-key (kbd "C-c r") 'elfeed)
  (setq elfeed-search-filter "+unread")
  :bind
  (:map elfeed-search-mode-map
        ;; C-p for play now
        ("C-c C-p" . bard/play-elfeed-video)
        ;; C-e for EMMS
        ("C-c C-e" . bard/add-video-emms-queue)
        ;; C-w for watch later
        ("C-c C-w" . bard/add-video-watch-later)
        ;; F is for fetch
        ("F"       . elfeed-update)))
(use-package elfeed-org
  :ensure t
  :init
  (elfeed-org)
  :config
  (setq rmh-elfeed-org-files (list "~/Notes/denote/feeds.org"
                                   "~/Notes/denote/youtube.org")))
```

2.14.3 Web Browsing (EWW and external)

```
(use-package eww
  :defer t
  :config
  (setq browse-url-handlers
        '(("wikipedia\\.org" . eww-browse-url)
          ("github" . browse-url-default-browser)
          ("youtube.com" . browse-url-default-browser)
          ("reddit.com" . browse-url-default-browser)))
  ;; shr optimizations
  (setq shr-use-colors nil)
  (setq shr-use-fonts nil)
  (setq shr-max-image-proportion 0.6)
  (setq shr-image-animate nil)
  (setq shr-width fill-column)
  (setq shr-max-width fill-column)
  (setq shr-discard-aria-hidden t)
  (setq shr-cookie-policy nil)
  ;; eww
  (setq eww-search-prefix "https://duckduckgo.com/html/?q=")
  (setq eww-history-limit 150)
  (setq eww-use-external-browser-for-content-type
        "\\`\\(video/\\|audio\\)")
  :bind
  ("C-c w" . eww))
(provide 'bard-emacs-web)
;;; bard-emacs-web.el ends here
```

2.15 bard-emacs-window

2.15.1 Load required libraries

```
(require 'bard-window)
```

```
(use-package emacs
  ;; configuration for window splits/window sizes
  :config
  (setq focus-follows-mouse t)
  (setq mouse-autoselect-window t)
  (setq window-combination-resize t)
  (setq even-window-sizes 'height-only)
  (setq window-sides-vertical nil)
  (setq switch-to-buffer-in-dedicated-window 'pop)
  (setq split-height-threshold 80
        split-width-threshold 125)
  (setq window-min-height 3
        window-min-width 30))
```

2.15.2 Moving windos

```
(use-package windmove
  :bind*
  (("C-M-<up>" . windmove-up)
   ("C-M-<right>" . windmove-right)
   ("C-M-<down>" . windmove-down)
   ("C-M-<left>" . windmove-left)
   ("C-M-S-<up>" . windmove-swap-states-up)
   ("C-M-S-<right>" . windmove-swap-states-right)
   ("C-M-S-<down>" . windmove-swap-states-down)
   ("C-M-S-<left>" . windmove-swap-states-left)))
```

2.15.3 Window display rules (`display-buffer-alist`)

Watch Protesilaos Stavrou's video about this to understand it. My configuration Just Works™, so I don't really get how to change it that much anymore. He can explain it a lot better than I can.

```
(use-package emacs
  :config
  (setq display-buffer-alist
        `(;; no window
          ("\\`\\*Async Shell Command\\*\\'"
           (display-buffer-no-window))
          ("\\`\\*\\(Warnings\\|Compile-Log\\|Org Links\\)\\*\\'"
           (display-buffer-no-window)
           (allow-no-window . t))
          ;; bottom side window
          ("\\*Org \\(Select\\|Note\\)\\*" ; the `org-capture' key
   selection and `org-add-log-note'
           (display-buffer-in-side-window)
           (dedicated . t)
           (side . bottom)
           (slot . 0)
           (window-parameters . ((mode-line-format . none))))
          ;; bottom buffer (NOT side window)
          ((or . ((derived-mode . flymake-diagnostics-buffer-mode)
                  (derived-mode . flymake-project-diagnostics-mode)
                  (derived-mode . messages-buffer-mode)
                  (derived-mode . backtrace-mode)))
           (display-buffer-reuse-mode-window
   display-buffer-at-bottom)
           (window-height . 0.3)
           (dedicated . t)
           (preserve-size . (t . t)))
          ;; terminal popups
          (prot-window-shell-or-term-p
           (display-buffer-reuse-mode-window
   display-buffer-at-bottom)
           (mode . (shell-mode eshell-mode comint-mode))
           (body-function . prot-window-select-fit-size))
          ("\\magit: .*"
           (display-buffer-same-window)
           (inhibit-same-window . nil)
           (dedicated . t))
          ("\\*Org Agenda\\*"
           (display-buffer-same-window)
           (inhibit-same-window . nil)
           (dedicated . t))
          ("\\*cfw-calendar\\*"
           (display-buffer-same-window)
           (inhibit-same-window . nil)
           (dedicated . t))
          ("\\*image-dired\\*"
           (display-buffer-reuse-mode-window
   display-buffer-in-side-window)
           (side . bottom)
           (window-height . 0.5))
          ("\\*image-dired-display-image\\*"
           (display-buffer-reuse-mode-window
   display-buffer-in-side-window)
           (side . right)
           (window-width . 0.35))
          ;; ("\\*Embark Actions\\*"
          ;;  (display-buffer-reuse-mode-window
   display-buffer-below-selected)
          ;;  (window-height . fit-window-to-buffer)
          ;;  (window-parameters . ((no-other-window . t)
          ;;                        (mode-line-format . none))))
          ("\\*\\(Output\\|Register Preview\\).*"
           (display-buffer-reuse-mode-window
   display-buffer-at-bottom))
          ;; below current window
          ("\\(\\*Capture\\*\\|CAPTURE-.*\\)"
           (display-buffer-in-side-window)
           (dedicated . t)
           (side . bottom)
           (slot . 0)
           (window-parameters . ((mode-line-format . none))))
          ("\\*\\vc-\\(incoming\\|outgoing\\|git : \\).*"
           (display-buffer-reuse-mode-window
   display-buffer-below-selected)
           (window-height . 0.1)
           (dedicated . t)
           (preserve-size . (t . t)))
          ((derived-mode . reb-mode) ; M-x re-builder
           (display-buffer-reuse-mode-window
   display-buffer-below-selected)
           (window-height . 4) ; note this is literal lines, not
   relative
           (dedicated . t)
           (preserve-size . (t . t)))
          ((or . ((derived-mode . occur-mode)
                  (derived-mode . grep-mode)
```

```
               (derived-mode . Buffer-menu-mode)
               (derived-mode . log-view-mode)
               (derived-mode . help-mode) ; See the hooks for `
visual-line-mode'
               "\\*\\(|Buffer List\\|Occur\\|vc-change-log\\|
eldoc.*\\).*"
               prot-window-shell-or-term-p
               ;; ,world-clock-buffer-name
               ))
        (prot-window-display-buffer-below-or-pop)
        (body-function . prot-window-select-fit-size))
       ("\\*\\(Calendar\\|Bookmark Annotation\\|ert\\).*"
        (display-buffer-reuse-mode-window
display-buffer-below-selected)
        (dedicated . t)
        (window-height . fit-window-to-buffer))
       ("\\*ispell-top-choices\\*.*"
        (display-buffer-reuse-mode-window
display-buffer-below-selected)
        (window-height . fit-window-to-buffer))
       )))
```

2.15.4 Frame

1. Keybinds

```
(use-package frame
  :ensure nil
  :bind (("C-x u" . undelete-frame)
         ("C-x f" . other-frame-prefix)
         ("C-x w t" . tear-off-window)
         ("C-x w c" . clone-indirect-buffer-other-window))
  :hook (after-init . undelete-frame-mode))
```

2. Beframe

```
(use-package beframe
  :ensure t
  :config
  (setq beframe-functions-in-frames '(project-prompt-project-dir
                                      notmuch))
  (setq beframe-create-frame-scratch-buffer nil)
  (setq beframe-global-buffers '("*scratch*" "*Messages*" "*
   Backtrace*"))
  (beframe-mode 1)
  (define-key global-map (kbd "C-x f") #'other-frame-prefix)
  (define-key global-map (kbd "C-c b") beframe-prefix-map)
  (define-key global-map (kbd "C-x C-b") #'beframe-buffer-menu)
  (define-key global-map (kbd "C-x B") #'select-frame-by-name)
  (define-key global-map (kbd "C-c b u") #'
   beframe-unassume-current-frame-buffers-selectively)
  (define-key global-map (kbd "C-c b a") #'
   beframe-assume-buffers-selectively-all-frames)
  ;; Consult integration
  (defvar consult-buffer-sources)
  (declare-function consult--buffer-state "consult")
  (with-eval-after-load 'consult
    (defface beframe-buffer
      '((t :inherit font-lock-string-face))
      "Face for `consult' framed buffers.")
    (defun my-beframe-buffer-names-sorted (&optional frame)
      "Return the list of buffers from `beframe-buffer-names'
   sorted by visibility.
    With optional argument FRAME, return the list of buffers of
    FRAME."
      (beframe-buffer-names frame :sort #'
   beframe-buffer-sort-visibility))
    (defvar beframe-consult-source
      `( :name     "Frame-specific buffers (current frame)"
         :narrow   ?F
         :category buffer
         :face     beframe-buffer
         :history  beframe-history
         :items    ,#'my-beframe-buffer-names-sorted
         :action   ,#'switch-to-buffer
         :state    ,#'consult--buffer-state))
    (add-to-list 'consult-buffer-sources 'beframe-consult-source
   )))
(provide 'bard-emacs-window)
;;; bard-emacs-window.el ends here
```

2.15.5 Undoing window actions (`winner-mode`)

```
(use-package winner-mode
  :init
  (winner-mode 1)
  :bind
  (("C-x <right>" . winner-redo)
   ("C-x <left>" . winner-undo)
   ("C-x C-n" . next-buffer)
   ("C-x C-p" . previous-buffer)
   ("C-x <up>" . next-buffer)
   ("C-x <down>" . previous-buffer)
   ("C-x w w" . bard/toggle-window-split)))
```

2.15.6 Ibuffer

```
(use-package ibuffer
  :ensure nil
  :config
  (setq ibuffer-default-sorting-mode 'major-mode)
  (ibuffer-auto-mode t))
```

2.16 bard-emacs-writing

```
(use-package emacs
  :ensure nil
  :demand t
  :bind
  (("C-x i" . insert-char)
   ("M-z"   . zap-to-char)
   ("<f10>"  . toggle-input-method))
  :config
  ;; Sentence size
  (setq sentence-end-double-space nil)
  ;; Keyboard things
  (setq default-input-method "cyrillic-yawerty")
  (setq default-transient-input-method "cyrillic-yawerty"))
;; spell check
(use-package text-mode
  :ensure nil
  :hook
  (text-mode . flyspell-mode))
;; Tab settings
```

```
(use-package emacs
  :config
  (setq tab-always-indent 'complete)
  (setq tab-first-completion 'word-or-paren-or-punct)
  (setq-default tab-width 4
                indent-tabs-mode nil))
```

2.16.1 Snippets

```
(use-package yasnippet
  :ensure t
  :config
  (setq yas-snippet-dirs '("~/.emacs.d/snippets"))
  (yas-global-mode t)
  )
(use-package yasnippet-capf
  :ensure t
  :after cape
  :config
  (add-to-list 'completion-at-point-functions #'yasnippet-capf))
```

2.16.2 Note taking

```
(use-package denote
  :ensure t
  :config
  (require 'bard-writing)
  (setq denote-directory "~/Notes/denote/")
  (setq denote-buffer-name-prefix "[Note] "
        denote-rename-buffer-format "%t %b")
  (setq denote-known-keywords
        '("emacs"
          "linux"
          "programming"
          "org"
          "school"
          "language"
          "history"
          "biology"
          ))
  (setq denote-templates
        '((default . "Related to — ")
          (todo . bard/denote-todo-template)))
  (setq denote-save-buffers t)
  (setq denote-prompts '(title keywords))
  (setq denote-sort-dired-extra-prompts nil)
  (setq denote-sort-dired-default-sort-component 'identifier)
  (setq denote-sort-dired-default-reverse-sort nil)
  ;; backlinks sidebar
  (setq denote-backlinks-display-buffer-action
        '((display-buffer-in-direction)
          (direction . right)
          (window-width . 0.33)
          (window-height . fit-window-to-buffer)
          (dedicated . t)))
  (denote-rename-buffer-mode 1)
  (require 'bard-writing)
  :hook
  (dired-mode . denote-dired-mode)
  :bind
  (("C-c n n" . denote)
   ("C-c n d" . denote-sort-dired)
   ("C-c n r" . denote-rename-file-using-front-matter)
   ("C-c n k" . denote-rename-file-keywords)
   ("C-c n I" . denote-add-links)
   ("C-c n b" . bard/consult-buffer-notes)   ; notes buffer
   ("C-c n B" . bard/ibuffer-notes)          ; notes buffer but more
   ("C-c n f" . bard/find-notes-file)        ; notes-find
   ("C-c n g" . bard/search-notes-directory))) ; notes-grep
(use-package denote-org
  :ensure t
  )
(use-package denote-silo
  :ensure t
  :config
  (setq denote-silo-directories '("~/Notes/denote"
                                  "~/Notes/Old Notes/")))
(use-package denote-sequence
  :ensure t
  :config
  (require 'bard-writing)
  :bind
  ("C-c n N" . denote-sequence)
  ("C-c n D" . denote-sequence-dired)
  ("C-c n <SPC>" . denote-sequence-region))
(use-package denote-journal
  :ensure t
  :bind
  ("C-c n j" . denote-journal-new-or-existing-entry)
  :config
  (setq denote-journal-directory "~/Notes/denote/journal/")
  (setq denote-journal-title-format "Daily Tasks and Notes"))
```

2.16.3 Denote roam

Denote-roam is my Emacs package that blends org roam and denote into one workflow.

```
(use-package denote-roam
  :ensure nil
  :load-path "~/Code/denote-roam/"
  :bind
  ("C-c n i" . denote-roam-insert-or-create-node)  ; node insert
  ("C-c n o" . denote-roam-find-or-create-node)    ; node open
  :custom
  (denote-roam-include-journal nil)
  (denote-roam-directory "~/Notes/denote")
  :config
  (denote-roam-mode t))
```

2.16.4 Org roam

```
(use-package org-roam
  :ensure t
  :custom
  (org-roam-directory (file-truename "~/Notes/denote"))
  :bind (("C-c n l" . org-roam-buffer-toggle))
  :config
  (setq org-roam-db-node-include-function
        (lambda ()
          (not (member "ATTACH" (org-get-tags)))))
  (org-roam-db-autosync-mode 1))
(use-package org-roam-ui
  :ensure t
  :bind
  ("C-c n u" . org-roam-ui-open)
  :custom
  (org-roam-ui-open-on-start nil))
```

2.16.5 Focused writing environment

```
;; Center line scrolling for focused writing
(use-package emacs
  :config
  (define-minor-mode bard/scroll-center-cursor-mode
    "Toggle centered cursor scrolling behavior."
    :init-value nil
    :lighter " S="
    :global nil
    (if bard/scroll-center-cursor-mode
        (setq-local scroll-margin (* (frame-height) 2)
                    scroll-conservatively 0
                    maximum-scroll-margin 0.5)
      (dolist (local '(scroll-preserve-screen-position
                       scroll-conservatively
                       maximum-scroll-margin
                       scroll-margin))
        (kill-local-variable `,local))))
  (defun bard/cursor-centered-p ()
    "Check if `bard/scroll-center-cursor-mode` is currently active."
    (bound-and-true-p bard/scroll-center-cursor-mode))
  :bind
  (("C-c L" . bard/scroll-center-cursor-mode)))
(use-package olivetti
  :ensure t
  :config
  (setq olivetti-minimum-body-width 90)
  (setq olivetti-recall-visual-line-mode-entry-state t)
  :hook
  ((olivetti-mode-on . (lambda () (olivetti-set-width 90)))
   ))
;; narrowing and focus mode
(use-package logos
  :ensure t
  :config
  (defun logos-reveal-entry ()
    "Reveal Org or Outline entry."
    (cond
     ((and (eq major-mode 'org-mode)
           (org-at-heading-p))
      (org-show-subtree))
     ((or (eq major-mode 'outline-mode)
          (bound-and-true-p outline-minor-mode))
      (outline-show-subtree))))
  (setq logos-outlines-are-pages t)
  (setq logos-outline-regexp-alist
        `((emacs-lisp-mode . "^;;;+ ")
          (org-mode . "^\\* +")
          (t . ,(or outline-regexp logos--page-delimiter))))
  (setq-default logos-hide-cursor nil
                logos-hide-mode-line nil
                logos-hide-header-line t
                logos-hide-buffer-boundaries t
                logos-hide-fringe t
                logos-variable-pitch t
                logos-olivetti t)
  (defun bard/logos--recenter-top ()
    "Use `recenter' to reposition the view at the top."
    (unless (derived-mode-p 'prog-mode)
      (recenter 1))) ; Use 0 for the absolute top
  :hook
  ((logos-page-motion . bard/logos--recenter-top))
  :hook
  ((org-mode . logos-focus-mode)
   (markdown-mode . logos-focus-mode))
  :bind
  (("M-]" . logos-forward-page-dwim)
   ("M-[" . logos-backward-page-dwim)
   ("<f9>" . logos-focus-mode)
   ("C-x n n" . logos-narrow-dwim)))
```

2.16.6 Citations and bibliography

```
(use-package citar
  :ensure t
  :bind
  ("C-c n c" . citar-open)
  :config
  (setq citar-bibliography '("~/Documents/bib/references.bib"))
  (setq org-cite-global-bibliography citar-bibliography
        org-cite-insert-processor 'citar
        org-cite-follow-processor 'citar
        org-cite-activate-processor 'citar)
  (setq citar-notes-paths '("~/Notes/denote"))
  (setq citar-library-paths '("~/Documents/Research Articles/"))
  ;; (setq citar-file-open-functions 'find-file)
  :hook
  (org-mode . citar-capf-setup)
  :bind (("C-c i" . citar-insert-citation)))
(use-package citar-denote
  :ensure t
  :config
  (citar-denote-mode t)
  )
(use-package citar-embark
  :ensure t
  )
```

2.16.7 Provide module

```
(provide 'bard-emacs-writing)
```

3 Libraries

3.1 bard-calendar

```
(require 'org)
```

3.1.1 Study session program spawning

I use a timer to study sometimes when I really don't lock in. It is a common lisp script that manages process of a timer called sowon.

```
(defun bard/auto-clock-in ()
  "Automatically clock in when task marked in progress (INPROG),
 and start study session."
  (when (equal (org-get-todo-state) "INPROG")
    (org-clock-in)
    (bard/study-session)))
```

```
(defun bard/study-session ()
  "Prompt for study parameters, run study session, and clock out
   when done."
  (interactive)
  (let* ((study-time (read-string "Study time (minutes): "))
         (break-time (read-string "Break time (minutes): "))
         (sessions (read-string "Number of sessions: "))
         (command (format "study %s %s %s" study-time break-time
   sessions))
         (process (start-process-shell-command "study-session" "*
   study*" command)))
    (set-process-sentinel
     process
     (lambda (_proc event)
       (when (string= event "finished\n")
         (progn (org-clock-out)
                (pop-to-buffer-same-window "todo.org")))))))
```

3.1.2 Org clock

```
  (defun bard/org-clock-report ()
    "Generate an org clock report in a separate org buffer."
    (interactive)
    (bard/new-org-buffer)
    (org-clock-report))
  (defun bard/org-clock-update-mode-line ()
    "Update the modeline, not sure why I have this."
    (interactive)
    (setq org-mode-line-string nil)
    (force-mode-line-update))
  (defun bard/org-clock-task-string ()
    "Return a simplified org clock task string.
Used in FVWM3 configuration to show clocked task in FVWMbuttons."
    (if (and (boundp 'org-mode-line-string)
             (not (string-equal "" org-mode-line-string))
             org-mode-line-string)
        (substring-no-properties org-mode-line-string)
      "No task clocked in"))
```

3.1.3 Open the calendar in its own window

I use this in window manager status bar to open a popup window to view the dates on the calendar.

```
(defun bard/open-calendar ()
  "Opens calendar as only window"
  (interactive)
  (calendar)
  (delete-other-windows))
```

3.1.4 Org agenda

```
(defun bard/choose-agenda ()
  "For viewing my custom agenda"
  (interactive)
  (let ((agenda-views '("Default" "Monthly" "Yearly")))
    (setq chosen-view (completing-read "Choose an agenda view: "
   agenda-views))
    (cond
     ((string= chosen-view "Yearly")
      (org-agenda nil "Y"))
     ((string= chosen-view "Monthly")
      (org-agenda nil "M"))
     ((string= chosen-view "Default")
      (org-agenda nil "D")))))
(defun bard/default-agenda ()
  "For viewing my custom agenda"
  (interactive)
  (org-agenda nil "D"))
```

3.1.5 Provide library

```
(provide 'bard-calendar)
;;; bard-calendar.el ends here
```

3.2 bard-compile

Code taken from here. I need this to solve a problem I had with pros-cli back when I did robotics in high school. The program had a bunch of color codes that weren't properly displayed in the compliation output. I really like using `M-x compile` for C/C++ development specifically.

```
(require 'ansi-color)
```

```
(defun endless/colorize-compilation ()
  "Colorize from `compilation-filter-start' to `point'."
  (let ((inhibit-read-only t))
    (ansi-color-apply-on-region
     compilation-filter-start (point))))
(add-hook 'compilation-filter-hook
          #'endless/colorize-compilation)
;; Stolen from (https://oleksandrmanzyuk.wordpress.com/2011/11/05/
   better-emacs-shell-part-i/)
(defun regexp-alternatives (regexps)
  "Return the alternation of a list of regexps."
  (mapconcat (lambda (regexp)
               (concat "\\(?:" regexp "\\)"))
             regexps "\\|"))
(defvar non-sgr-control-sequence-regexp nil
  "Regexp that matches non-SGR control sequences.")
(setq non-sgr-control-sequence-regexp
      (regexp-alternatives
       '(;; icon name escape sequences
         "\033\\][0-2];.*?\007"
         ;; non-SGR CSI escape sequences
         "\033\\[\\??[0-9;]*[^0-9;m]"
         ;; noop
         "\012\033\\[2K\033\\[1F"
         )))
(defun filter-non-sgr-control-sequences-in-region (begin end)
  (save-excursion
    (goto-char begin)
    (while (re-search-forward
            non-sgr-control-sequence-regexp end t)
      (replace-match ""))))
(defun filter-non-sgr-control-sequences-in-output (ignored)
  (let ((start-marker
         (or comint-last-output-start
             (point-min-marker)))
        (end-marker
         (process-mark
          (get-buffer-process (current-buffer)))))
    (filter-non-sgr-control-sequences-in-region
     start-marker
     end-marker)))
```

```
(add-hook 'comint-output-filter-functions
          'filter-non-sgr-control-sequences-in-output)
(provide 'bard-compile)
```

3.3 bard-email

Word for word copy of Protesilaos notmuch library. Email in Emacs is already complicated enough, and I had enough of a hard time without his help.

```
(require 'prot-common)
(eval-when-compile (require 'cl-lib))
(defgroup prot-notmuch ()
  "Extensions for notmuch.el."
  :group 'notmuch)
(defcustom prot-notmuch-delete-tag "del"
  "Single tag that applies to mail marked for deletion.
This is used by `prot-notmuch-delete-mail'."
  :type 'string
  :group 'prot-notmuch)
(defcustom prot-notmuch-mark-delete-tags
  `(,(format "+%s" prot-notmuch-delete-tag) "-inbox" "-unread")
  "List of tags to mark for deletion.
To actually delete email, refer to `prot-notmuch-delete-mail'."
  :type '(repeat string)
  :group 'prot-notmuch)
(defcustom prot-notmuch-mark-flag-tags '("+flag" "-unread")
  "List of tags to mark as important (flagged).
This gets the `notmuch-tag-flagged' face, if that is specified in
`notmuch-tag-formats'."
  :type '(repeat string)
  :group 'prot-notmuch)
(defcustom prot-notmuch-mark-spam-tags '("+spam" "-inbox" "-unread")
  "List of tags to mark as spam."
  :type '(repeat string)
  :group 'prot-notmuch)
(autoload 'notmuch-interactive-region "notmuch")
(autoload 'notmuch-tag-change-list "notmuch")
(autoload 'notmuch-search-next-thread "notmuch")
(autoload 'notmuch-search-tag "notmuch")
(defmacro prot-notmuch-search-tag-thread (name tags)
  "Produce NAME function parsing TAGS."
  (declare (indent defun))
  `(defun ,name (&optional untag beg end)
     ,(format
       "Mark with `%s' the currently selected thread.
Operate on each message in the currently selected thread.  With
optional BEG and END as points delimiting a region that
encompasses multiple threads, operate on all those messages
instead.
With optional prefix argument (\\[universal-argument]) as UNTAG,
reverse the application of the tags.
This function advances to the next thread when finished."
       tags)
     (interactive (cons current-prefix-arg (
   notmuch-interactive-region)))
     (when ,tags
       (notmuch-search-tag
        (notmuch-tag-change-list ,tags untag) beg end))
     (when (eq beg end)
       (notmuch-search-next-thread))))
(prot-notmuch-search-tag-thread
  prot-notmuch-search-delete-thread
  prot-notmuch-mark-delete-tags)
(prot-notmuch-search-tag-thread
  prot-notmuch-search-flag-thread
  prot-notmuch-mark-flag-tags)
(prot-notmuch-search-tag-thread
  prot-notmuch-search-spam-thread
  prot-notmuch-mark-spam-tags)
(defmacro prot-notmuch-show-tag-message (name tags)
  "Produce NAME function parsing TAGS."
  (declare (indent defun))
  `(defun ,name (&optional untag)
     ,(format
       "Apply `%s' to message.
With optional prefix argument (\\[universal-argument]) as UNTAG,
reverse the application of the tags."
       tags)
     (interactive "P")
     (when ,tags
       (apply 'notmuch-show-tag-message
              (notmuch-tag-change-list ,tags untag)))))
(prot-notmuch-show-tag-message
  prot-notmuch-show-delete-message
  prot-notmuch-mark-delete-tags)
(prot-notmuch-show-tag-message
  prot-notmuch-show-flag-message
  prot-notmuch-mark-flag-tags)
(prot-notmuch-show-tag-message
  prot-notmuch-show-spam-message
  prot-notmuch-mark-spam-tags)
(defun prot-notmuch-delete-mail ()
  "Permanently delete mail marked as `prot-notmuch-delete-mail'.
Prompt for confirmation before carrying out the operation.
Do not attempt to refresh the index.  This will be done upon the
next invocation of 'notmuch new'."
  (interactive)
  (let* ((del-tag prot-notmuch-delete-tag)
         (count
          (string-to-number
           (with-temp-buffer
             (shell-command
              (format "notmuch count tag:%s" prot-notmuch-delete-tag
   ) t)
             (buffer-substring-no-properties (point-min) (1- (
   point-max))))))
         (mail (if (> count 1) "mails" "mail")))
    (unless (> count 0)
      (user-error "No mail marked as `%s'" del-tag))
    (when (yes-or-no-p
           (format "Delete %d %s marked as `%s'?" count mail del-tag
   ))
      (shell-command
       (format "notmuch search --output=files --format=text0 tag:%s
   | xargs -r0 rm" del-tag)
       t))))
(provide 'bard-email)
```

3.4 bard-embark

Main embark configuration found here. This is another one of the files I borrowed from Protesilaos, but renamed/modified so I can remember.

```
(require 'embark)
(defvar-keymap bard-embark-general-map
  :parent embark-general-map
  "i" #'embark-insert
  "w" #'embark-copy-as-kill
  "E" #'embark-export
  "S" #'embark-collect
  "A" #'embark-act-all
  "DEL" #'delete-region)
(defvar-keymap bard-embark-url-map
  :parent embark-general-map
  "b" #'browse-url
  "d" #'embark-download-url
  "e" #'eww)
(defvar-keymap bard-embark-buffer-map
  :parent embark-general-map
  "k" #'bard-simple-kill-buffer
  "o" #'switch-to-buffer-other-window
  "e" #'ediff-buffers)
(add-to-list 'embark-post-action-hooks (list '
   bard-simple-kill-buffer 'embark--restart))
(defvar-keymap bard-embark-file-map
  :parent embark-general-map
  "f" #'find-file
  "j" #'embark-dired-jump
  "c" #'copy-file
  "e" #'ediff-files)
(defvar-keymap bard-embark-identifier-map
  :parent embark-general-map
  "h" #'display-local-help
  "." #'xref-find-definitions
  "o" #'occur)
(defvar-keymap bard-embark-command-map
  :parent embark-general-map
  "h" #'describe-command
  "." #'embark-find-definition)
(defvar-keymap bard-embark-expression-map
  :parent embark-general-map
  "e" #'pp-eval-expression
  "m" #'pp-macroexpand-expression)
(defvar-keymap bard-embark-function-map
  :parent embark-general-map
  "h" #'describe-function
  "." #'embark-find-definition)
(defvar-keymap bard-embark-package-map
  :parent embark-general-map
  "h" #'describe-package
  "i" #'package-install
  "d" #'package-delete
  "r" #'package-reinstall
  "b" #'embark-browse-package-url
  "w" #'embark-save-package-url)
(defvar-keymap bard-embark-symbol-map
  :parent embark-general-map
  "h" #'describe-symbol
  "." #'embark-find-definition)
(defvar-keymap bard-embark-variable-map
  :parent embark-general-map
  "h" #'describe-variable
  "." #'embark-find-definition)
(defvar-keymap bard-embark-region-map
  :parent embark-general-map
  "a" #'align-regexp
  "D" #'delete-duplicate-lines
  "f" #'flush-lines
  "i" #'epa-import-keys-region
  "d" #'epa-decrypt-armor-in-region
  "r" #'repunctuate-sentences
  "s" #'sort-lines
  "u" #'untabify)
;; The minimal indicator shows cycling options, but I have no use
;; for those.  I want it to be silent.
(defun bard-embark-no-minimal-indicator ())
(advice-add #'embark-minimal-indicator :override #'
   bard-embark-no-minimal-indicator)
(provide 'bard-embark)
```

3.5 bard-eshell

```
(require 'cl-lib)
(require 'eshell)
```

3.5.1 Eshell aliases

```
;; aliases
(setq bard/eshell-aliases
      '((g  . magit)
        (gl . magit-log)
        (d  . dired)
        (o  . find-file)
        (oo . find-file-other-window)
        (l  . (lambda () (eshell/ls '-la)))
        (eshell/clear . eshell/clear-scrollback)))
(mapc (lambda (alias)
        (defalias (car alias) (cdr alias)))
      bard/eshell-aliases)
```

3.5.2 Prot-eshell code

I believe Protesilaos has stopped using eshell, but I went back in his git history to the time when he used it in a video and took some custom elisp from there.

```
(defun prot-eshell--cd (dir)
  "Routine to cd into DIR."
  (delete-region eshell-last-output-end (point-max))
  (when (> eshell-last-output-end (point))
    (goto-char eshell-last-output-end))
  (insert-and-inherit "cd " (eshell-quote-argument dir))
  (eshell-send-input))
(defun prot-eshell-complete-recent-dir (dir &optional arg)
  "Switch to a recent Eshell directory.
When called interactively, DIR is selected with completion from
the elements of `eshell-last-dir-ring'.
With optional ARG prefix argument (\\[universal-argument]) also
open the directory in a `dired' buffer."
  (interactive
   (list
    (if-let ((dirs (ring-elements eshell-last-dir-ring)))
        (completing-read "Switch to recent dir: " dirs nil t)
```

```
      (user-error "There is no Eshell history for recent directories
   "))
    current-prefix-arg))
  (prot-eshell--cd dir)
  ;; UPDATE 2022-01-04 10:48 +0200: The idea for `dired-other-window
   '
  ;; was taken from Sean Whitton's `spw/eshell-cd-recent-dir'.
   Check
  ;; Sean's dotfiles: <https://git.spwhitton.name/dotfiles>.
  (when arg
    (dired-other-window dir)))
(defun bard/eshell-find-file-at-point ()
  "Run `find-file` to find file"
  (interactive)
  (let ((file (ffap-file-at-point)))
    (if file
        (find-file file)
      (user-error "No file at point"))))
(defcustom prot-eshell-output-buffer "*Exported Eshell output*"
  "Name of buffer with the last output of Eshell command.
Used by `prot-eshell-export'."
  :type 'string
  :group 'prot-eshell)
(defcustom prot-eshell-output-delimiter "* * *"
  "Delimiter for successive `prot-eshell-export' outputs.
This is formatted internally to have newline characters before
and after it."
  :type 'string
  :group 'prot-eshell)
(defun prot-eshell--command-prompt-output ()
  "Capture last command prompt and its output."
  (let ((beg (save-excursion
               (goto-char (eshell-beginning-of-input))
               (goto-char (point-at-bol)))))
    (when (derived-mode-p 'eshell-mode)
      (buffer-substring-no-properties beg (eshell-end-of-output)))))
;;;###autoload
(defun prot-eshell-export ()
  "Produce a buffer with output of the last Eshell command.
If `prot-eshell-output-buffer' does not exist, create it.  Else
append to it, while separating multiple outputs with
```

```
`prot-eshell-output-delimiter'."
  (interactive)
  (let ((eshell-output (prot-eshell--command-prompt-output)))
    (with-current-buffer (get-buffer-create
   prot-eshell-output-buffer)
      (let ((inhibit-read-only t))
        (goto-char (point-max))
        (unless (eq (point-min) (point-max))
          (insert (format "\n%s\n\n" prot-eshell-output-delimiter)))
        (goto-char (point-at-bol))
        (insert eshell-output)
        (switch-to-buffer-other-window (current-buffer))))))
(defgroup bard-eshell-faces nil
  "Faces for my custom modeline."
  :group 'prot-eshell-faces)
(defface bard-eshell-highlight-yellow-bg
  '((default :inherit (bold prot-modeline-indicator-button))
    (((class color) (min-colors 88) (background light))
     :background "#805000" :foreground "white")
    (((class color) (min-colors 88) (background dark))
     :background "#ffc800" :foreground "black")
    (t :background "yellow" :foreground "black"))
  "Face for modeline indicators with a background."
  :group 'bard-eshell-faces)
(defun prot-eshell-narrow-output-highlight-regexp (regexp)
  "Narrow to last command output and highlight REGEXP."
  (interactive
   (list (read-regexp "Regexp to highlight" nil '
   prot-eshell--output-highlight-history)))
  (narrow-to-region (eshell-beginning-of-output)
                    (eshell-end-of-output))
  (goto-char (point-min))
  (highlight-regexp regexp 'prot-eshell-highlight-yellow-bg)
  (message "%s to last output and highlighted '%s'"
           (propertize "Narrowed" 'face 'bold)
           (propertize regexp 'face 'italic)))
(defun select-or-create (arg)
  "Commentary ARG."
  (if (string= arg "New eshell")
      (eshell t)
    (switch-to-buffer arg)))
```

```
(defun eshell-switcher (&optional arg)
  "Commentary ARG."
  (interactive)
  (let* (
         (buffers (cl-remove-if-not (lambda (n) (eq (
   buffer-local-value 'major-mode n) 'eshell-mode)) (buffer-list)) )
         (names (mapcar (lambda (n) (buffer-name n)) buffers))
         (num-buffers (length buffers) )
         (in-eshellp (eq major-mode 'eshell-mode)))
    (cond ((eq num-buffers 0) (eshell (or arg t)))
          ((not in-eshellp) (switch-to-buffer (car buffers)))
          (t (select-or-create (completing-read "Select Shell:" (
   cons "New eshell" names)))))))
```

3.5.3 Prompt configuration

```
;; taken from https://github.com/karthink/.emacs.d/blob/master/lisp/
   setup-shells.el
(use-package eshell
  :defer
  :config
  (setq eshell-prompt-regexp "^.* λ "
        eshell-prompt-function #'bard/eshell-default-prompt-fn)
  (defun bard/eshell-default-prompt-fn ()
    "Generate the prompt string for eshell. Use for `
   eshell-prompt-function'."
    (concat (if (bobp) "" "\n")
            (let ((pwd (eshell/pwd)))
              (propertize (if (equal pwd "~")
                               pwd
                             (abbreviate-file-name pwd))
                          'face 'bard/eshell-prompt-pwd))
            (propertize (bard/eshell--current-git-branch)
                        'face 'bard/eshell-prompt-git-branch)
            (propertize " λ" 'face (if (zerop
   eshell-last-command-status) 'success 'error))
            " "))
  (defsubst bard/eshell--current-git-branch ()
    ;; TODO Refactor me
    (cl-destructuring-bind (status . output)
        (with-temp-buffer (cons
                             (or (call-process "git" nil t nil "
   symbolic-ref" "-q" "--short" "HEAD")
                                 (call-process "git" nil t nil "
   describe" "--all" "--always" "HEAD")
                                 -1)
                             (string-trim (buffer-string))))
      (if (equal status 0)
          (format " [%s]" output)
        "")))
  (defface bard/eshell-prompt-pwd '((t (:inherit
   font-lock-keyword-face)))
    "TODO"
    :group 'eshell)
  (defface bard/eshell-prompt-git-branch '((t (:inherit
   font-lock-builtin-face)))
    "TODO"
    :group 'eshell))
```

3.5.4 Provide library

```
(provide 'bard-eshell)
```

3.6 **bard-media**

3.6.1 Load required libraries

```
(require 'cl-lib)
(require 'seq)
(require 'emms)
(require 'image-dired)
(require 'dired-x)
```

3.6.2 EMMS Convenience functions

Somewhat related to Elfeed/EMMS Youtube Code.

```
(defun bard/play-youtube-video ()
  "Play the YouTube URL at point or prompt for one if none is found.
   "
  (interactive)
  (let* ((url-at-point (thing-at-point 'url t))
         (url (if (and url-at-point
                      (string-match-p "https?://\\(www\\.\\)?\\(
   youtube\\.com\\|youtu\\.be\\)" url-at-point))
                 url-at-point
               (read-string "Enter YouTube URL: "))))
    (if (and url (string-match-p "https?://\\(www\\.\\)?\\(youtube
   \\.com\\|youtu\\.be\\)" url))
        (async-shell-command (format "mpv '%s'" url))
      (message "The URL is not a valid YouTube link: %s" url))))
(defun bard/save-emms-watch-later ()
  "Save the current EMMS playlist to `bard/watch-later-file` using `
   bard/emms-playlist-format`."
  (interactive)
  (when (and bard/watch-later-file bard/emms-playlist-format)
    (emms-playlist-save bard/emms-playlist-format bard/
   watch-later-file)
    (message "Playlist saved to %s" bard/watch-later-file)))
(defun bard/emms-download-current-video (destination)
  "Download the currently playing EMMS video and move it to
   DESTINATION."
  (interactive "DSelect destination directory: ")
  (require 'emms)
  (let* ((track (emms-playlist-current-selected-track))
         (url (emms-track-get track 'name))
         (default-directory (file-name-as-directory
   temporary-file-directory))
         (downloader (executable-find "yt-dlp"))
         (output-template "%(title)s.%(ext)s"))
    (unless downloader
      (error "yt-dlp or youtube-dl is not installed or not in PATH")
   )
    (unless (string-match-p "^https?://" url)
      (error "Current track is not a valid video URL"))
    (let ((cmd (format "%s -o \"%s\" \"%s\""
                      downloader output-template url)))
      (message "Downloading video from: %s" url)
      (let ((exit-code (shell-command cmd)))
        (if (not (eq exit-code 0))
            (error "Download failed, see *Messages* for details")
          ;; Move the downloaded file
          (let* ((downloaded-file (car (directory-files
   default-directory t ".*\\(mp4\\|mkv\\|webm\\)$" 'time)))
```

```
                (target-path (expand-file-name (
   file-name-nondirectory downloaded-file) destination)))
           (rename-file downloaded-file target-path t)
           (message "Video saved to: %s" target-path)))))))
```

3.6.3 Interacting with `nsxiv` image browser through elisp

I made a video about this workflow here.

```
(defun bard/image-browser-choose (directory)
  "Open nsxiv in thumbnail mode on DIRECTORY.
Asks the user whether to enable recursive mode and whether to output
    marked files to a buffer."
  (interactive "DSelect directory: ")
  (let* ((recursive (if (y-or-n-p "Recursive searching? ") "-r" ""))
         (stdout (if (y-or-n-p "Output marked files to buffer? ") "-
   o" ""))
         (full-dir (expand-file-name directory))
         (args (remove "" (list "nsxiv" "-t" stdout recursive
   full-dir))))
    ;; Pre-clear the output buffer if needed
    (when (string= stdout "-o")
      (with-current-buffer (get-buffer-create "*nsxiv*")
        (read-only-mode 0)
        (erase-buffer)))
    (message "Running: %s" (string-join args " "))
    (let ((process (apply #'start-process "nsxiv" "*nsxiv*" args)))
      (when (string= stdout "-o")
        (set-process-sentinel
         process
         (lambda (proc event)
           (when (string= event "finished\n")
             (with-current-buffer "*nsxiv*"
               (read-only-mode nil)
               (goto-char (point-min)))
             ;; Read marked files
             (let ((files (with-current-buffer "*nsxiv*"
                             (split-string (buffer-string) "\n" t))))
               (bard/open-marked-in-dired files)))))
        (pop-to-buffer "*nsxiv*")))))
(defun bard/open-marked-in-dired (files)
  "Open a list of FILES in an interactive Dired buffer."
  (if (and files (listp files))
      (dired (cons "*nsxiv-marked*" files))
    (message "No valid files to show in Dired.")))
(defun bard/image-browser-marked ()
  "Open nsxiv on the marked files in Dired.
Assumes that files have already been validated."
  (let ((files (dired-get-marked-files)))
    (message "Opening marked files: %s" (string-join files ", "))
    (apply #'start-process "nsxiv" "*nsxiv*" "nsxiv" "-t" files)))
(defun bard/image-browser ()
  "Open nsxiv in a context-sensitive way:
- If in Dired with marked files, open those with nsxiv.
- If in Dired with no marked files, prompt for a directory.
- If not in Dired, prompt for a directory."
  (interactive)
  (cond
   ;; In Dired and files are marked
   ((and (derived-mode-p 'dired-mode)
         (< 1 (length (dired-get-marked-files))))
    (message "Opening marked files from Dired...")
    (bard/image-browser-marked))
   ;; In Dired but no marked files
   ((derived-mode-p 'dired-mode)
    (message "No files marked in Dired. Prompting for directory...")
    (call-interactively #'bard/image-browser-choose))
   ;; Not in Dired
   (t
    (message "Not in Dired. Prompting for directory...")
    (call-interactively #'bard/image-browser-choose))))
(provide 'bard-media.el)
```

3.7 bard-modeline

This is another one of prot's libraries that I copied and modified a long time ago. The modifications this time were quite insignificant, but I did add some custom modules that I wanted.

```
(require 'prot-common)
(defgroup prot-modeline nil
  "Custom modeline that is stylistically close to the default."
  :group 'mode-line)
```

```
(defgroup prot-modeline-faces nil
  "Faces for my custom modeline."
  :group 'prot-modeline)
(defcustom prot-modeline-string-truncate-length 9
  "String length after which truncation should be done in small
   windows."
  :type 'natnum)
(defun mode-line-window-selected-p ()
  "Return non-nil if we're updating the mode line for the selected
   window.
This function is meant to be called in `:eval' mode line
constructs to allow altering the look of the mode line depending
on whether the mode line belongs to the currently selected window
or not."
  (let ((window (selected-window)))
    (or (eq window (old-selected-window))
    (and (minibuffer-window-active-p (minibuffer-window))
         (with-selected-window (minibuffer-window)
           (eq window (minibuffer-selected-window)))))))
;;;; Faces
(defface prot-modeline-indicator-button nil
  "Generic face used for indicators that have a background.
Modify this face to, for example, add a :box attribute to all
relevant indicators (combines nicely with my `spacious-padding'
package).")
(defface prot-modeline-indicator-red
  '((default :inherit bold)
    (((class color) (min-colors 88) (background light))
     :foreground "#880000")
    (((class color) (min-colors 88) (background dark))
     :foreground "#ff9f9f")
    (t :foreground "red"))
  "Face for modeline indicators (e.g. see my `notmuch-indicator')."
  :group 'prot-modeline-faces)
(defface prot-modeline-indicator-red-bg
  '((default :inherit (bold prot-modeline-indicator-button))
    (((class color) (min-colors 88) (background light))
     :background "#aa1111" :foreground "white")
    (((class color) (min-colors 88) (background dark))
     :background "#ff9090" :foreground "black")
    (t :background "red" :foreground "black"))
  "Face for modeline indicators with a background."
  :group 'prot-modeline-faces)
(defface prot-modeline-indicator-green
  '((default :inherit bold)
    (((class color) (min-colors 88) (background light))
     :foreground "#005f00")
    (((class color) (min-colors 88) (background dark))
     :foreground "#73fa7f")
    (t :foreground "green"))
  "Face for modeline indicators (e.g. see my `notmuch-indicator')."
  :group 'prot-modeline-faces)
(defface prot-modeline-indicator-green-bg
  '((default :inherit (bold prot-modeline-indicator-button))
    (((class color) (min-colors 88) (background light))
     :background "#207b20" :foreground "white")
    (((class color) (min-colors 88) (background dark))
     :background "#77d077" :foreground "black")
    (t :background "green" :foreground "black"))
  "Face for modeline indicators with a background."
  :group 'prot-modeline-faces)
(defface prot-modeline-indicator-yellow
  '((default :inherit bold)
    (((class color) (min-colors 88) (background light))
     :foreground "#6f4000")
    (((class color) (min-colors 88) (background dark))
     :foreground "#f0c526")
    (t :foreground "yellow"))
  "Face for modeline indicators (e.g. see my `notmuch-indicator')."
  :group 'prot-modeline-faces)
(defface prot-modeline-indicator-yellow-bg
  '((default :inherit (bold prot-modeline-indicator-button))
    (((class color) (min-colors 88) (background light))
     :background "#805000" :foreground "white")
    (((class color) (min-colors 88) (background dark))
     :background "#ffc800" :foreground "black")
    (t :background "yellow" :foreground "black"))
  "Face for modeline indicators with a background."
  :group 'prot-modeline-faces)
(defface prot-modeline-indicator-blue
  '((default :inherit bold)
    (((class color) (min-colors 88) (background light))
     :foreground "#00228a")
    (((class color) (min-colors 88) (background dark))
     :foreground "#88bfff")
    (t :foreground "blue"))
  "Face for modeline indicators (e.g. see my `notmuch-indicator')."
  :group 'prot-modeline-faces)
(defface prot-modeline-indicator-blue-bg
  '((default :inherit (bold prot-modeline-indicator-button))
    (((class color) (min-colors 88) (background light))
     :background "#0000aa" :foreground "white")
    (((class color) (min-colors 88) (background dark))
     :background "#77aaff" :foreground "black")
    (t :background "blue" :foreground "black"))
  "Face for modeline indicators with a background."
  :group 'prot-modeline-faces)
(defface prot-modeline-indicator-magenta
  '((default :inherit bold)
    (((class color) (min-colors 88) (background light))
     :foreground "#6a1aaf")
    (((class color) (min-colors 88) (background dark))
     :foreground "#e0a0ff")
    (t :foreground "magenta"))
  "Face for modeline indicators (e.g. see my `notmuch-indicator')."
  :group 'prot-modeline-faces)
(defface prot-modeline-indicator-magenta-bg
  '((default :inherit (bold prot-modeline-indicator-button))
    (((class color) (min-colors 88) (background light))
     :background "#6f0f9f" :foreground "white")
    (((class color) (min-colors 88) (background dark))
     :background "#e3a2ff" :foreground "black")
    (t :background "magenta" :foreground "black"))
  "Face for modeline indicators with a background."
  :group 'prot-modeline-faces)
(defface prot-modeline-indicator-cyan
  '((default :inherit bold)
    (((class color) (min-colors 88) (background light))
     :foreground "#004060")
    (((class color) (min-colors 88) (background dark))
     :foreground "#30b7cc")
    (t :foreground "cyan"))
  "Face for modeline indicators (e.g. see my `notmuch-indicator')."
  :group 'prot-modeline-faces)
(defface prot-modeline-indicator-cyan-bg
  '((default :inherit (bold prot-modeline-indicator-button))
    (((class color) (min-colors 88) (background light))
     :background "#006080" :foreground "white")
    (((class color) (min-colors 88) (background dark))
     :background "#40c0e0" :foreground "black")
    (t :background "cyan" :foreground "black"))
  "Face for modeline indicators with a background."
  :group 'prot-modeline-faces)
;;;; Common helper functions
(defun prot-modeline--string-truncate-p (str)
  "Return non-nil if STR should be truncated."
  (and (prot-common-window-small-p)
       (> (length str) prot-modeline-string-truncate-length)
       (not (one-window-p :no-minibuffer))))
(defun prot-modeline--truncate-p ()
  "Return non-nil if truncation should happen.
This is a more general and less stringent variant of
`prot-modeline--string-truncate-p'."
  (and (prot-common-window-small-p)
       (not (one-window-p :no-minibuffer))))
(defun prot-modeline-string-truncate (str)
  "Return truncated STR, if appropriate, else return STR.
Truncation is done up to `prot-modeline-string-truncate-length'."
  (if (prot-modeline--string-truncate-p str)
      (concat (substring str 0 prot-modeline-string-truncate-length)
    "...")
    str))
(defun prot-modeline-string-truncate-end (str)
  "Like `prot-modeline-string-truncate' but truncate from STR
   beginning."
  (if (prot-modeline--string-truncate-p str)
      (concat "..." (substring str (-
   prot-modeline-string-truncate-length)))
    str))
(defun prot-modeline--first-char (str)
  "Return first character from STR."
  (substring str 0 1))
(defun prot-modeline-string-abbreviate (str)
  "Abbreviate STR individual hyphen or underscore separated words.
```

```
Also see `prot-modeline-string-abbreviate-but-last'."
  (if (prot-modeline--string-truncate-p str)
      (mapconcat #'prot-modeline--first-char (split-string str "[_-]
   ") "-")
    str))
(defun prot-modeline-string-abbreviate-but-last (str nthlast)
  "Abbreviate STR, keeping NTHLAST words intact.
Also see `prot-modeline-string-abbreviate'."
  (if (prot-modeline--string-truncate-p str)
      (let* ((all-strings (split-string str "[_-]"))
             (nbutlast-strings (nbutlast (copy-sequence all-strings)
    nthlast))
             (last-strings (nreverse (ntake nthlast (nreverse (
   copy-sequence all-strings)))))
             (first-component (mapconcat #'prot-modeline--first-char
    nbutlast-strings "-"))
             (last-component (mapconcat #'identity last-strings "-")
   ))
        (if (string-empty-p first-component)
            last-component
          (concat first-component "-" last-component)))
    str))
;;;; Keyboard macro indicator
(defvar-local prot-modeline-kbd-macro
    '(:eval
      (when (and (mode-line-window-selected-p) defining-kbd-macro)
        (propertize " KMacro " 'face '
   prot-modeline-indicator-blue-bg)))
  "Mode line construct displaying `mode-line-defining-kbd-macro'.
Specific to the current window's mode line.")
;;;; Narrow indicator
(defvar-local prot-modeline-narrow
    '(:eval
      (when (and (mode-line-window-selected-p)
                 (buffer-narrowed-p)
                 (not (derived-mode-p 'Info-mode 'help-mode '
   special-mode 'message-mode)))
        (propertize " Narrow " 'face '
   prot-modeline-indicator-cyan-bg)))
  "Mode line construct to report the multilingual environment.")
;;;; Centered cursor indicator
```

```
(defvar-local bard-modeline-centered-cursor
    '(:eval
      (when (and (mode-line-window-selected-p)
                 (bard/cursor-centered-p)
                 (not (derived-mode-p 'Info-mode 'help-mode '
   special-mode 'message-mode)))
        (propertize " Center " 'face '
   prot-modeline-indicator-yellow-bg)))
  "Mode line construct to report the multilingual environment.")
;; FIXME: Combine these two functions one day...
(defvar-local bard-modeline-ryo-modal-normal
    '(:eval
      (when (and (mode-line-window-selected-p)
                 (not (bard/ryo-insert-p))
                 (not (derived-mode-p 'Info-mode 'help-mode '
   special-mode 'message-mode)))
        (propertize "<N>" 'face 'prot-modeline-indicator-magenta-bg)
   )
      )
  "Mode line construct to show normal mode for ryo-modal.")
(defvar-local bard-modeline-ryo-modal-insert
    '(:eval
      (when (and (mode-line-window-selected-p)
                 (bard/ryo-insert-p)
                 (not (derived-mode-p 'Info-mode 'help-mode '
   special-mode 'message-mode)))
        (propertize "<I>" 'face 'prot-modeline-indicator-blue-bg))
      )
  "Mode line construct to show insert mode for ryo-modal.")
;;;; Input method
(defvar-local prot-modeline-input-method
    '(:eval
      (when current-input-method-title
        (propertize (format " %s " current-input-method-title)
                    'face 'prot-modeline-indicator-green-bg
                    'mouse-face 'mode-line-highlight)))
  "Mode line construct to report the multilingual environment.")
;;;; Buffer status
(defvar-local prot-modeline-buffer-status
    '(:eval
      (when (file-remote-p default-directory)
        (propertize " @ "
                    'face 'prot-modeline-indicator-red-bg
                    'mouse-face 'mode-line-highlight)))
  "Mode line construct for showing remote file name.")
;;;; Dedicated window
(defvar-local prot-modeline-window-dedicated-status
    '(:eval
      (when (window-dedicated-p)
        (propertize " = "
                    'face 'prot-modeline-indicator-magenta-bg
                    'mouse-face 'mode-line-highlight)))
  "Mode line construct for dedicated window indicator.")
;;;; Buffer name and modified status
(defun prot-modeline-buffer-identification-face ()
  "Return appropriate face or face list for `
   prot-modeline-buffer-identification'."
  (let ((file (buffer-file-name)))
    (cond
     ((and (mode-line-window-selected-p)
           file
           (buffer-modified-p))
      '(error italic mode-line-buffer-id))
     ((and file (buffer-modified-p))
      'italic)
     ((mode-line-window-selected-p)
      'mode-line-buffer-id))))
(defun prot-modeline--buffer-name ()
  "Return `buffer-name', truncating it if necessary.
See `prot-modeline-string-truncate'."
  (when-let ((name (buffer-name)))
    (prot-modeline-string-truncate name)))
(defun prot-modeline-buffer-name ()
  "Return buffer name, with read-only indicator if relevant."
  (let ((name (prot-modeline--buffer-name)))
    (if buffer-read-only
        (format "%s %s" (char-to-string #xE0A2) name)
      name)))
(defun prot-modeline-buffer-name-help-echo ()
  "Return `help-echo' value for `prot-modeline-buffer-identification
   '."
  (concat
   (propertize (buffer-name) 'face 'mode-line-buffer-id)
   "\n"
   (propertize
    (or (buffer-file-name)
        (format "No underlying file.\nDirectory is: %s"
   default-directory))
    'face 'font-lock-doc-face)))
(defvar-local prot-modeline-buffer-identification
    '(:eval
      (propertize (prot-modeline-buffer-name)
                  'face (prot-modeline-buffer-identification-face)
                  'mouse-face 'mode-line-highlight
                  'help-echo (prot-modeline-buffer-name-help-echo)))
  "Mode line construct for identifying the buffer being displayed.
Propertize the current buffer with the `mode-line-buffer-id'
face.  Let other buffers have no face.")
;;;; Major mode
(defun prot-modeline-major-mode-indicator ()
  "Return appropriate propertized mode line indicator for the major
   mode."
  (let ((indicator (cond
                    ((derived-mode-p 'text-mode) "§")
                    ((derived-mode-p 'prog-mode) "λ")
                    ((derived-mode-p 'term-mode) ">_")
                    ((derived-mode-p 'emms-playlist-mode) "♪")
                    (t "◦"))))
    (propertize indicator 'face 'shadow)))
(defun prot-modeline-major-mode-name ()
  "Return capitalized `major-mode' without the -mode suffix."
  (capitalize (string-replace "-mode" "" (symbol-name major-mode))))
(defun prot-modeline-major-mode-help-echo ()
  "Return `help-echo' value for `prot-modeline-major-mode'."
  (if-let ((parent (get major-mode 'derived-mode-parent)))
      (format "Symbol: `%s'.  Derived from: `%s'" major-mode parent)
    (format "Symbol: `%s'." major-mode)))
(defvar-local prot-modeline-major-mode
    (list
     (propertize "%[" 'face 'prot-modeline-indicator-red)
     '(:eval
       (concat
        (prot-modeline-major-mode-indicator)
```

```
         " "
         (propertize
          (prot-modeline-string-abbreviate-but-last
           (prot-modeline-major-mode-name)
           2)
          'mouse-face 'mode-line-highlight
          'help-echo (prot-modeline-major-mode-help-echo))))
     (propertize "%]" 'face 'prot-modeline-indicator-red))
  "Mode line construct for displaying major modes.")
(defvar-local prot-modeline-process
    (list '("" mode-line-process))
  "Mode line construct for the running process indicator.")
;;;; Git branch and diffstat
(declare-function vc-git--symbolic-ref "vc-git" (file))
(defun prot-modeline--vc-branch-name (file backend)
  "Return capitalized VC branch name for FILE with BACKEND."
  (when-let ((rev (vc-working-revision file backend))
             (branch (or (vc-git--symbolic-ref file)
                         (substring rev 0 7))))
    (capitalize branch)))
(declare-function vc-git-working-revision "vc-git" (file))
(defvar prot-modeline-vc-map
  (let ((map (make-sparse-keymap)))
    (define-key map [mode-line down-mouse-1] 'vc-diff)
    (define-key map [mode-line down-mouse-3] 'vc-root-diff)
    map)
  "Keymap to display on VC indicator.")
(defun prot-modeline--vc-help-echo (file)
  "Return `help-echo' message for FILE tracked by VC."
  (format "Revision: %s\nmouse-1: `vc-diff'\nmouse-3: `vc-root-diff'
   "
          (vc-working-revision file)))
(defun prot-modeline--vc-text (file branch &optional face)
  "Prepare text for Git controlled FILE, given BRANCH.
With optional FACE, use it to propertize the BRANCH."
  (concat
   (propertize (char-to-string #xE0A0) 'face 'shadow)
   " "
   (propertize branch
               'face face
               'mouse-face 'mode-line-highlight
               'help-echo (prot-modeline--vc-help-echo file)
               'local-map prot-modeline-vc-map)
   ;; " "
   ;; (prot-modeline-diffstat file)
   ))
(defun prot-modeline--vc-details (file branch &optional face)
  "Return Git BRANCH details for FILE, truncating it if necessary.
The string is truncated if the width of the window is smaller
than `split-width-threshold'."
  (prot-modeline-string-truncate
   (prot-modeline--vc-text file branch face)))
(defvar prot-modeline--vc-faces
  '((added . vc-locally-added-state)
    (edited . vc-edited-state)
    (removed . vc-removed-state)
    (missing . vc-missing-state)
    (conflict . vc-conflict-state)
    (locked . vc-locked-state)
    (up-to-date . vc-up-to-date-state))
  "VC state faces.")
(defun prot-modeline--vc-get-face (key)
  "Get face from KEY in `prot-modeline--vc-faces'."
   (alist-get key prot-modeline--vc-faces 'up-to-date))
(defun prot-modeline--vc-face (file backend)
  "Return VC state face for FILE with BACKEND."
  (prot-modeline--vc-get-face (vc-state file backend)))
(defvar-local prot-modeline-vc-branch
    '(:eval
      (when-let* (((mode-line-window-selected-p))
                  (file (buffer-file-name))
                  (backend (vc-backend file))
                  ;; ((vc-git-registered file))
                  (branch (prot-modeline--vc-branch-name file
   backend))
                  (face (prot-modeline--vc-face file backend)))
        (prot-modeline--vc-details file branch face)))
  "Mode line construct to return propertized VC branch.")
;;;; Flymake errors, warnings, notes
(declare-function flymake--severity "flymake" (type))
(declare-function flymake-diagnostic-type "flymake" (diag))
;; Based on `flymake--mode-line-counter'.
```

```
(defun prot-modeline-flymake-counter (type)
  "Compute number of diagnostics in buffer with TYPE's severity.
TYPE is usually keyword `:error', `:warning' or `:note'."
  (let ((count 0))
    (dolist (d (flymake-diagnostics))
      (when (= (flymake--severity type)
               (flymake--severity (flymake-diagnostic-type d)))
        (cl-incf count)))
    (when (cl-plusp count)
      (number-to-string count))))
(defvar prot-modeline-flymake-map
  (let ((map (make-sparse-keymap)))
    (define-key map [mode-line down-mouse-1] '
   flymake-show-buffer-diagnostics)
    (define-key map [mode-line down-mouse-3] '
   flymake-show-project-diagnostics)
    map)
  "Keymap to display on Flymake indicator.")
(defmacro prot-modeline-flymake-type (type indicator &optional face)
  "Return function that handles Flymake TYPE with stylistic
   INDICATOR and FACE."
  `(defun ,(intern (format "prot-modeline-flymake-%s" type)) ()
     (when-let ((count (prot-modeline-flymake-counter
                        ,(intern (format ":%s" type)))))
       (concat
        (propertize ,indicator 'face 'shadow)
        (propertize count
                    'face ',(or face type)
                     'mouse-face 'mode-line-highlight
                     ;; FIXME 2023-07-03: Clicking on the text with
                     ;; this buffer and a single warning present,
   the
                     ;; diagnostics take up the entire frame.  Why?
                     'local-map prot-modeline-flymake-map
                     'help-echo "mouse-1: buffer diagnostics\
   nmouse-3: project diagnostics")))))
(prot-modeline-flymake-type error "☒")
(prot-modeline-flymake-type warning "!")
(prot-modeline-flymake-type note "·" success)
(defvar-local prot-modeline-flymake
    `(:eval
      (when (and (bound-and-true-p flymake-mode)
                 (mode-line-window-selected-p))
        (list
         ;; See the calls to the macro `prot-modeline-flymake-type'
         '(:eval (prot-modeline-flymake-error))
         '(:eval (prot-modeline-flymake-warning))
         '(:eval (prot-modeline-flymake-note)))))
  "Mode line construct displaying `flymake-mode-line-format'.
Specific to the current window's mode line.")
(with-eval-after-load 'eglot
  (setq mode-line-misc-info
        (delete '(eglot--managed-mode (" [" eglot--mode-line-format
   "] ")) mode-line-misc-info)))
(defvar-local prot-modeline-eglot
    `(:eval
      (when (and (featurep 'eglot) (mode-line-window-selected-p))
        '(eglot--managed-mode eglot--mode-line-format)))
  "Mode line construct displaying Eglot information.
Specific to the current window's mode line.")
(defvar-local prot-modeline-notmuch-indicator
    '(notmuch-indicator-mode
      (" "
       (:eval (when (mode-line-window-selected-p)
                notmuch-indicator--counters))))
  "The equivalent of `notmuch-indicator-mode-line-construct'.
Display the indicator only on the focused window's mode line.")
(defvar-local bard-evil-state-indicator
  '(:eval
    (when (and (bound-and-true-p evil-local-mode)
               (mode-line-window-selected-p))
      (let ((state-label
             (pcase evil-state
               ('normal  (propertize " <N>" 'face '
   prot-modeline-indicator-green))
               ('insert  (propertize " <I> " 'face '
   prot-modeline-indicator-blue))
               ('visual  (propertize " <V> " 'face '
   prot-modeline-indicator-yellow))
               ('replace (propertize " <R> " 'face '
   prot-modeline-indicator-red))
               ('emacs   (propertize " <E> " 'face '
   prot-modeline-indicator-magenta))
               ('motion  (propertize " <V> " 'face '
   prot-modeline-indicator-cyan))
               (_        (propertize " <> " 'face 'shadow))))
        state-label)))
  "Modeline indicator for current Evil state.")
;;;; Miscellaneous
(defvar-local prot-modeline-misc-info
    '(:eval
      (when (mode-line-window-selected-p)
        mode-line-misc-info))
  "Mode line construct displaying `mode-line-misc-info'.
Specific to the current window's mode line.")
;;;; Risky local variables
;; NOTE 2023-04-28: The `risky-local-variable' is critical, as those
;; variables will not work without it.
(dolist (construct '(prot-modeline-kbd-macro
                     prot-modeline-narrow
                     bard-modeline-centered-cursor
                     bard-evil-state-indicator
                     prot-modeline-input-method
                     prot-modeline-buffer-status
                     prot-modeline-window-dedicated-status
                     prot-modeline-evil
                     prot-modeline-buffer-identification
                     prot-modeline-major-mode
                     prot-modeline-process
                     prot-modeline-vc-branch
                     prot-modeline-flymake
                     prot-modeline-eglot
                     prot-modeline-misc-info
                     prot-modeline-notmuch-indicator))
  (put construct 'risky-local-variable t))
(provide 'bard-modeline)
```

3.8 bard-package

I renamed the functions from this post because otherwise I would forget I had them in my system. I keep personal/custom functions prefixed with `bard/` so I can look at them through minibuffer if I forgot the name. This shows me that they are not part of default Emacs.

```
;; taken and renamed functions from
(defun bard/package-report ()
  "Report total package counts grouped by archive."
  (interactive)
  (package-refresh-contents)
  (bard/display-package-report
   (let* ((arch-pkgs (bard/archive-packages))
          (counts (seq-sort-by #'cdr #'> (bard/archive-counts
   arch-pkgs)))
          (by-arch (seq-group-by #'car arch-pkgs)))
     (concat
      (format "Total packages: %s\n\n" (apply #'+ (mapcar #'cdr
   counts)))
      (mapconcat
       (lambda (archive)
         (concat "• "
                 (format "%s (%s)" (car archive) (cdr archive))
                 ": "
                 (mapconcat (lambda (ap-pair) (cdr ap-pair))
                            (alist-get (car archive) by-arch)
                            ", ")))
       counts
       "\n\n")))))
(defun bard/display-package-report (output)
  "Display OUTPUT in a popup buffer."
  (let ((buffer-name "*package-report*"))
    (with-help-window buffer-name
      (with-current-buffer buffer-name
        (visual-line-mode 1)
        (erase-buffer)
        (insert output)
        (goto-char (point-min))))))
(defun bard/archive-packages ()
  "Return a list of (archive . package) cons cells."
  (seq-reduce
   (lambda (res package)
     (let ((archive (package-desc-archive
                     (cadr (assq package package-archive-contents)))
   )
           (pkg (symbol-name package)))
       (push (cons archive pkg) res)))
   (mapcar #'car package-alist)
   nil))
(defun bard/archive-counts (arch-pkgs)
  "Return a list of cons cells from alist ARCH-PKGS.
The cars are package archives, the cdrs are the number of
packages installed from each archive."
  (seq-reduce
   (lambda (counts key)
     (cons (cons key (+ 1 (or (cdr (assoc key counts)) 0)))
           (assoc-delete-all key counts)))
   (mapcar #'car arch-pkgs)
   nil))
(provide 'bard-package)
```

3.9 **bard-theme**

3.9.1 Fontaine font presets

The main configuration for fontaine is found at bard-emacs-theme::Fonts. These are the presets for different fonts that I can switch between using `C-c f`.

```
(setq fontaine-presets
      '((default
         :default-height 140
         :default-family "Iosevka Comfy"
         :variable-pitch-family "Iosevka Comfy"
         :variable-pitch-height 1.0
         :fixed-pitch-family "Iosevka Comfy"
         :fixed-pitch-height 1.0
         :bold-weight bold
         )
        (tiny
         :inherit default
         :default-height 135)
        (wide
         :default-height 135
         :default-family "Iosevka Comfy Wide"
         :fixed-pitch-family "Iosevka Comfy Wide"
         :variable-pitch-family "Iosevka Comfy Wide Motion Duo")
        (prot
         :default-family "Iosevka Comfy Wide Motion"
```

```
          :default-height 130
          :default-weight medium
          :fixed-pitch-family "Iosevka Comfy Wide Motion"
          :variable-pitch-family "Iosevka Comfy Wide Duo"
          :bold-weight extrabold)
         (mono
          :default-height 130
          :default-family "monospace"
          :fixed-pitch-family "monospace"
          :variable-pitch-family "Baskerville"
          :variable-pitch-height 140)
         (mono-large
          :inherit mono
          :default-height 150
          :variable-pitch-height 160)
         (mac
          :default-height 130
          :default-family "Monaco"
          :variable-pitch-family "Monaco"
          :fixed-pitch-family "Monaco")
         (large
          :inherit default
          :default-height 160)
         (huge
          :inherit default
          :default-height 180)
         (t
          :default-family "Monospace")))
```

3.9.2 Japanese CJK font

Explanation provided by AJATT — Japanese fonts. Essentially: Chinese fonts and Japanese fonts look different for certain Kanji. Since I don't want Chinese glyphs prioritized in Emacs over Japanese ones, I set the font to the Japanese (JP) version of Noto CJK font which is a very accessible and popular font for Asian languages.

```
(set-fontset-font
 t 'han
 (font-spec :family "Noto Serif CJK JP") nil 'prepend)
```

3.9.3 Switching themes

```
(defun bard/disable-all-themes ()
  "disable all active themes."
  (interactive)
  (dolist (i custom-enabled-themes)
    (disable-theme i)))
(defvar bard/after-theme-load-hook nil
  "Hook that runs after a new theme is loaded using `bard/
   select-theme`.")
(dolist (hook '(bard/after-theme-load-hook))
  (add-hook hook #'fontaine-apply-current-preset)
  (add-hook hook #'logos-update-fringe-in-buffers))
(defun bard/select-theme (&optional theme)
  "Enable the specified THEME, or prompt the user to select one if
   THEME is nil."
  (interactive
   (list
    (completing-read "Select theme: "
                     (mapcar 'symbol-name (custom-available-themes))
   )))
  (let* ((theme-symbol (if (symbolp theme) theme (intern theme)))
         (theme-name (symbol-name theme-symbol))
         (display-theme-name (if (string-suffix-p "-theme"
   theme-name)
                                 (substring theme-name 0 -6)
                               theme-name))
         (colored-theme-name (propertize display-theme-name 'face
   '(:weight bold))))
    (bard/disable-all-themes)
    (load-theme theme-symbol t)
    (message "Loaded the %s theme" colored-theme-name)
    (run-hooks 'bard/after-theme-load-hook)))
```

3.9.4 Updating cursor for text mode

```
(defvar my-last-cursor-type nil)
(defun bard/update-cursor-type ()
  "Set cursor type to 'bar in text modes, 'box otherwise.
leave it alone in pdf-view-mode."
  (unless (derived-mode-p 'pdf-view-mode)
    (let ((new-cursor (if (derived-mode-p 'text-mode) 'bar 'box)))
      (unless (eq my-last-cursor-type new-cursor)
        (setq cursor-type new-cursor)
        (setq my-last-cursor-type new-cursor)))))
(add-hook 'post-command-hook #'bard/update-cursor-type)
```

3.9.5 Heading faces

```
(defun bard/outline-heading-faces ()
  (set-face-attribute 'org-document-title nil
                      :inherit '(outline-1 variable-pitch)
                      :weight 'light
                      :height 1.5)
  (set-face-attribute 'org-level-1 nil
                      :inherit '(outline-2 variable-pitch)
                      :weight 'light
                      :height 1.3)
  (set-face-attribute 'org-level-2 nil
                      :inherit 'outline-3
                      :height 1.2)
  (set-face-attribute 'org-level-3 nil
                      :inherit '(outline-4 variable-pitch)
                      :height 1.1)
  (set-face-attribute 'org-level-4 nil
                      :inherit '(outline-5 variable-pitch)
                      :height 1.1)
  (set-face-attribute 'org-level-5 nil
                      :inherit '(outline-6 variable-pitch)
                      :height 1.1)
  (set-face-attribute 'org-level-6 nil
                      :inherit '(outline-6 variable-pitch)
                      :height 1.1)
  (set-face-attribute 'org-agenda-date nil
                      :inherit 'variable-pitch
                      :weight 'bold
                      :height 1.3)
  (set-face-attribute 'org-agenda-structure nil
                      :inherit 'variable-pitch
                      :weight 'bold
                      :height 1.5))
(add-hook 'bard/after-theme-load-hook #'bard/outline-heading-faces)
```

3.9.6 Provide library

```
(provide 'bard-theme)
```

3.10 **bard-web**

3.10.1 Load required libraries

```
(require 'emms)
(require 'elfeed-search)
```

3.10.2 Elfeed/EMMS YouTube code

I made a video about this here: https://bardman.dev/technology/elfeed.

```
(defun bard/play-elfeed-video ()
  "Play the URL of the entry at point in mpv if it's a YouTube video
   ."
  (interactive)
  (let ((entry (elfeed-search-selected :single)))
    (if entry
        (let ((url (elfeed-entry-link entry)))
          (if (and url (string-match-p "https?://\\(www\\.\\)?
   youtube\\.com\\|youtu\\.be" url))
              (progn
                (async-shell-command (format "mpv '%s'" url))
                (elfeed-search-untag-all-unread))
            (message "The URL is not a YouTube link: %s" url)))
      (message "No entry selected in Elfeed."))))
(defun bard/add-video-emms-queue ()
  "Play the URL of the entry at point in mpv if it's a YouTube video
   . Add it to EMMS queue."
  (interactive)
  (let ((entry (elfeed-search-selected :single)))
    (if entry
        (let ((url (elfeed-entry-link entry)))
          (if (and url (string-match-p "https?://\\(www\\.\\)?
   youtube\\.com\\|youtu\\.be" url))
              (let* ((playlist-name "Watch Later")
                     (playlist-buffer (get-buffer (format " *%s*"
   playlist-name))))
                (unless playlist-buffer
                (setq playlist-buffer (emms-playlist-new (format "
    *%s*" playlist-name))))
               (emms-playlist-set-playlist-buffer playlist-buffer)
               (emms-add-url url)
               (elfeed-search-untag-all-unread)
               (message "Added YouTube video to EMMS playlist: %s"
   url))
           (message "The URL is not a YouTube link: %s" url)))
      (message "No entry selected in Elfeed."))))
(defun bard/add-video-watch-later ()
  "Add the current Elfeed YouTube entry URL to '~/Videos/watch-later
   .m3u' and mark it as read."
  (interactive)
  (let ((entry (elfeed-search-selected :single)))
    (if entry
        (let* ((url (elfeed-entry-link entry))
               (watch-later-file (expand-file-name "~/Videos/
   watch-later.m3u")))
          (if (and url (string-match-p "https?://\\(www\\.\\)?
   youtube\\.com\\|youtu\\.be" url))
              (progn
                (with-temp-buffer
                  (insert (concat url "\n"))
                  (append-to-file (point-min) (point-max)
   watch-later-file))
                ;; Remove the 'unread tag from the entry directly
                (setf (elfeed-entry-tags entry)
                      (remove 'unread (elfeed-entry-tags entry)))
                ;; Force UI update
                (when (derived-mode-p 'elfeed-search-mode)
                  (elfeed-search-update-entry entry))
                (message "Added video to watch later: %s" url))
            (message "The URL is not a YouTube link: %s" url)))
      (message "No entry selected in Elfeed."))))
```

3.10.3 Provide library

```
(provide 'bard-web)
```

3.11 bard-window

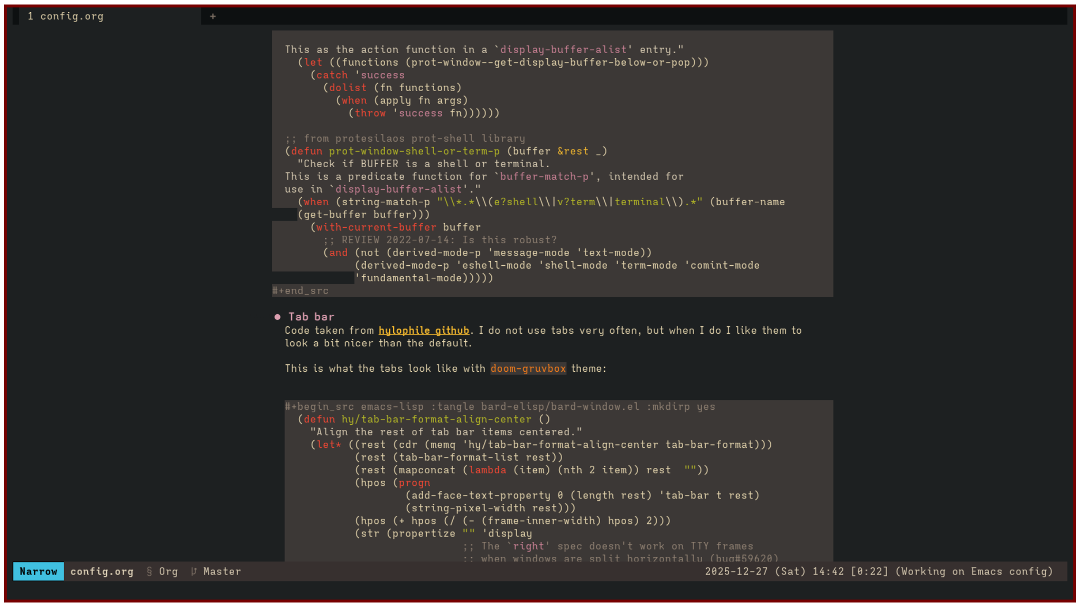
This module has some code from Protesilaos's dotemacs, especially the prot-window.el file. I copied most of this code a long time ago, so his library may have changed since then. But for me it works, and I never had to toy with it. No reason to reinvent the wheel with this one.

```
(require 'prot-common)
(defvar prot-window-window-sizes
  '( :max-height (lambda () (floor (frame-height) 3))
     :min-height 10
     :max-width (lambda () (floor (frame-width) 4))
     :min-width 20)
  "Property list of maximum and minimum window sizes.
The property keys are `:max-height', `:min-height', `:max-width',
and `:min-width'.  They all accept a value of either a
number (integer or floating point) or a function.")
(defun prot-window--get-window-size (key)
  "Extract the value of KEY from `prot-window-window-sizes'."
  (when-let ((value (plist-get prot-window-window-sizes key)))
    (cond
     ((functionp value)
      (funcall value))
     ((numberp value)
      value)
     (t
      (error "The value of `%s' is neither a number nor a function"
   key)))))
(defun prot-window-select-fit-size (window)
  "Select WINDOW and resize it.
The resize pertains to the maximum and minimum values for height
and width, per `prot-window-window-sizes'.
Use this as the `body-function' in a `display-buffer-alist' entry."
  (select-window window)
  (fit-window-to-buffer
   window
   (prot-window--get-window-size :max-height)
   (prot-window--get-window-size :min-height)
   (prot-window--get-window-size :max-width)
   (prot-window--get-window-size :min-width))
  ;; If we did not use `display-buffer-below-selected', then we must
  ;; be in a lateral window, which has more space.  Then we do not
  ;; want to dedicate the window to this buffer, because we will be
  ;; running out of space.
  (when (or (window-in-direction 'above) (window-in-direction 'below
   ))
    (set-window-dedicated-p window t)))
(defun prot-window--get-display-buffer-below-or-pop ()
  "Return list of functions for `
   prot-window-display-buffer-below-or-pop'."
  (list
   #'display-buffer-reuse-mode-window
   (if (or (prot-common-window-small-p)
           (prot-common-three-or-more-windows-p))
       #'display-buffer-below-selected
     #'display-buffer-pop-up-window)))
(defun prot-window-display-buffer-below-or-pop (&rest args)
  "Display buffer below current window or pop a new window.
The criterion for choosing to display the buffer below the
current one is a non-nil return value for
`prot-common-window-small-p'.
Apply ARGS expected by the underlying `display-buffer' functions.
This as the action function in a `display-buffer-alist' entry."
  (let ((functions (prot-window--get-display-buffer-below-or-pop)))
    (catch 'success
      (dolist (fn functions)
        (when (apply fn args)
          (throw 'success fn))))))
;; from protesilaos prot-shell library
(defun prot-window-shell-or-term-p (buffer &rest _)
  "Check if BUFFER is a shell or terminal.
This is a predicate function for `buffer-match-p', intended for
use in `display-buffer-alist'."
  (when (string-match-p "\\*.*\\(e?shell\\|v?term\\|terminal\\).*" (
   buffer-name (get-buffer buffer)))
    (with-current-buffer buffer
      ;; REVIEW 2022-07-14: Is this robust?
      (and (not (derived-mode-p 'message-mode 'text-mode))
           (derived-mode-p 'eshell-mode 'shell-mode 'term-mode '
   comint-mode 'fundamental-mode)))))
```

3.11.1 Tab bar

Code taken from hylophile github. I do not use tabs very often, but when I do I like them to look a bit nicer than the default.

This is what the tabs look like with `doom-gruvbox` theme:



```
1 config.org    +

This as the action function in a `display-buffer-alist' entry."
  (let ((functions (prot-window--get-display-buffer-below-or-pop)))
    (catch 'success
      (dolist (fn functions)
        (when (apply fn args)
          (throw 'success fn))))))

;; from protesilaos prot-shell library
(defun prot-window-shell-or-term-p (buffer &rest _)
  "Check if BUFFER is a shell or terminal.
This is a predicate function for `buffer-match-p', intended for
use in `display-buffer-alist'."
  (when (string-match-p "\\*.*\\(e?shell\\|v?term\\|terminal\\).*" (buffer-name
  (get-buffer buffer)))
    (with-current-buffer buffer
      ;; REVIEW 2022-07-14: Is this robust?
      (and (not (derived-mode-p 'message-mode 'text-mode))
           (derived-mode-p 'eshell-mode 'shell-mode 'term-mode 'comint-mode
           'fundamental-mode)))))
#+end_src

● Tab bar
  Code taken from hylophile github. I do not use tabs very often, but when I do I like them to
  look a bit nicer than the default.

  This is what the tabs look like with doom-gruvbox theme:


  #+begin_src emacs-lisp :tangle bard-elisp/bard-window.el :mkdirp yes
    (defun hy/tab-bar-format-align-center ()
      "Align the rest of tab bar items centered."
      (let* ((rest (cdr (memq 'hy/tab-bar-format-align-center tab-bar-format)))
             (rest (tab-bar-format-list rest))
             (rest (mapconcat (lambda (item) (nth 2 item)) rest  ""))
             (hpos (progn
                     (add-face-text-property 0 (length rest) 'tab-bar t rest)
                     (string-pixel-width rest)))
             (hpos (+ hpos (/ (- (frame-inner-width) hpos) 2)))
             (str (propertize "" 'display
                              ;; The `right' spec doesn't work on TTY frames
                              ;; when windows are split horizontally (bug#59620)
Narrow  config.org  § Org  ⑂ Master          2025-12-27 (Sat) 14:42 [0:22] (Working on Emacs config)
```

```
(defun hy/tab-bar-format-align-center ()
  "Align the rest of tab bar items centered."
  (let* ((rest (cdr (memq 'hy/tab-bar-format-align-center
   tab-bar-format)))
         (rest (tab-bar-format-list rest))
         (rest (mapconcat (lambda (item) (nth 2 item)) rest  ""))
         (hpos (progn
                 (add-face-text-property 0 (length rest) 'tab-bar t
   rest)
                 (string-pixel-width rest)))
         (hpos (+ hpos (/ (- (frame-inner-width) hpos) 2)))
         (str (propertize "" 'display
                          ;; The `right' spec doesn't work on TTY
   frames
                          ;; when windows are split horizontally (
   bug#59620)
                          (if (window-system)
                              `(space :align-to (- right (,hpos)))
```

```
                            `(space :align-to (,(- (
   frame-inner-width) hpos)))))))
    `((align-center menu-item ,str ignore))))
(setq tab-bar-tab-name-format-function #'hy/
   tab-bar-tab-name-format-default)
(defun hy/tab-bar-tab-name-format-default (tab i)
  (let* ((hint (format "%d" i))
         (name (alist-get 'name tab))
         (dir (concat "(" (alist-get 'dir tab "") ")"))
         (name-format (concat
                       " "
                       (propertize hint 'face 'tab-bar-hint)
                       " "
                       name
                       " ")))
    (add-face-text-property
     0 (length name-format)
     (funcall tab-bar-tab-face-function tab)
     'append name-format)
    name-format))
(setq tab-bar-tab-name-function #'hy/tab-bar-tab-name-current)
(defun hy/tab-bar-tab-name-current ()
  (hy/shorten-string
   (hy/abbreviate-tab-name
    (buffer-name (window-buffer (or (minibuffer-selected-window)
                                    (and (window-minibuffer-p)
                                         (get-mru-window))))))
   25))
(defun hy/set-tab-dir ()
  (setf (alist-get 'dir (cdr (tab-bar--current-tab-find)))
        (hy/tab-bar-dir)))
(defun hy/abbreviate-directory-path (path)
  "Turns `~/code/test/t` into `~/c/t/project`."
  (let* ((directories (seq-filter (lambda (s) (not (string= s "")))
   (split-string path "/")))
         (last-dir (car (last directories)))
         (abbreviated-dirs (mapcar (lambda (dir)
                                     (if (string= dir last-dir)
                                         dir
                                       (substring dir 0 (if (
   string-prefix-p "." dir) 2 1))))
```

```
                                   directories)))
    (mapconcat 'identity abbreviated-dirs "/")))
(defun hy/tab-bar-dir ()
  (hy/shorten-string (hy/abbreviate-directory-path
                      (abbreviate-file-name
                       (or (projectile-project-root)
   default-directory)))
                     10
                     t))
(defun hy/shorten-string (string max-length &optional at-start)
  (let ((len (length string)))
    (if (> len max-length)
        (if at-start
            (concat  "…" (substring string (- len max-length) len))
          (concat (substring string 0 max-length) "…"))
      string)))
(defun hy/abbreviate-tab-name (name)
  (string-trim (replace-regexp-in-string
                (rx (or "*" "helpful" "Org Src"))
                "" name)))
```

3.11.2 Toggle window split direction

```
(defun bard/toggle-window-split ()
  "Toggle between horizontal and vertical window splits, preserving
   buffer layout."
  (interactive)
  (let ((current-buffers (mapcar #'window-buffer (window-list))) ;
   List of buffers in current windows
        (split-direction (if (= (window-width) (frame-width))
                             'vertical
                           'horizontal)))
    (delete-other-windows)
    ;; Toggle the split direction
    (if (eq split-direction 'horizontal)
        (split-window-vertically)
      (split-window-horizontally))
    ;; Restore buffers to the new windows
    (let ((windows (window-list)))
      (cl-loop for buffer in current-buffers
               for window in windows
               do (set-window-buffer window buffer)))))
(provide 'bard-window)
```

3.12 bard-writing

3.12.1 Load required libraries

```
(require 'consult)
(require 'beframe)
(require 'calendar)
(require 'org-roam-node)
(require 'denote)
```

3.12.2 Note buffers

```
(defvar bard/consult--source-notes
  `(:name     "Note Buffers"
             :narrow   ?n
             :category buffer
             :face     consult-buffer
             :history  buffer-name-history
             :items    ,(lambda ()
                          (mapcar #'buffer-name
                                  (seq-filter
                                   (lambda (buf)
                                     (string-prefix-p "[Note]" (
   buffer-name buf)))
                                   (beframe-buffer-list))))
             :action   ,#'switch-to-buffer
             :state    ,#'consult--buffer-state)
  "Consult source for note buffers (limited to beframe buffers).")
(defun bard/consult-buffer-notes ()
  "Show `consult-buffer` limited to buffers starting with [Note]."
  (interactive)
  (consult-buffer '(bard/consult--source-notes)))
(defun bard/ibuffer-notes ()
  "Open `ibuffer` limited to buffers starting with [Note]."
  (interactive)
  (ibuffer nil "*Ibuffer-Notes*"
           '((name . "^\\[Note\\]"))))
```

3.12.3 Searching notes

```
(defun bard/find-notes-file ()
  (interactive)
  (consult-find "~/Notes/denote"))
(defun bard/search-notes-directory ()
  (interactive)
  (consult-grep "~/Notes/denote"))
```

3.12.4 Denote

```
(defvar bard/class-dirs
  '(("ANTH 204" . "~/Documents/Uni/FALL2025-ANTH 204/")
    ("CHEM 201" . "~/Documents/Uni/FALL2025-CHEM 201/")
    ("CHEM 207" . "~/Documents/Uni/FALL2025-CHEM 207/")
    ("ENGL 105" . "~/Documents/Uni/FALL2025-ENGL 105/")
    ("ENGR 101" . "~/Documents/Uni/FALL2025-ENGR 101/")
    ("ENGR 110" . "~/Documents/Uni/FALL2025-ENGR 110/"))
  "Mapping of class names to their document directories.")
(defvar bard/uni-notes-file "~/Notes/denote/uni.org"
  "Path to the main university org file.")
(defun bard/jump-to-class (class)
  "Jump to CLASS heading in `bard/uni-notes-file` and open its dir
   in dired."
  (interactive
   (list (completing-read "Class: " (mapcar #'car bard/class-dirs)))
   )
  (let* ((dir (cdr (assoc class bard/class-dirs))))
    ;; split windows
    (delete-other-windows)
    (let ((notes-window (selected-window))
          (dired-window (split-window-right)))
      ;; open notes file and jump to heading
      (with-selected-window notes-window
        (find-file bard/uni-notes-file)
        (widen)
        (goto-char (point-min))
        (message class)
        (search-forward class nil nil))
      ;; open dired in right window
      (with-selected-window dired-window
        (dired dir)))))
```

```
(defun bard/jump-to-class-new-frame (class)
  "Open CLASS notes and dir in a new frame titled after CLASS, even
   with beframe."
  (interactive
   (list (completing-read "Class: " (mapcar #'car bard/class-dirs)))
   )
  (let* ((dir (cdr (assoc class bard/class-dirs)))
         (frame (make-frame `((frame-title-format . ,class)))))
    (select-frame-set-input-focus frame)
    (delete-other-windows)
    (let ((notes-window (selected-window))
          (dired-window (split-window-right)))
      (with-selected-window notes-window
        (find-file bard/uni-notes-file)
        (widen)
        (goto-char (point-min))
        (search-forward class nil nil))
      (with-selected-window dired-window
        (dired dir)
        (beframe-rename-current-frame)))))
;; Optional: bind to a key
(global-set-key (kbd "C-c u") #'bard/jump-to-class)
(global-set-key (kbd "C-c U") #'bard/jump-to-class-new-frame)
```

3.12.5 Denote journal template

```
(defun bard/denote-todo-template ()
  "Return string for daily tasks heading in `denote-journal' entries
   ."
  (with-temp-buffer
    (org-mode)
    (insert (format "* Tasks for %s\n** Время я потратил
   бездельничая\n\n* Notes for today\n\n"
                    (format-time-string "%Y-%m-%d (%a)")))
    (let ((org-clock-clocktable-default-properties
           '(:scope file :maxlevel 3 :link nil :compact t)))
      (org-clock-report))
    (buffer-string)))
```

3.12.6 Math input mode

```
;; Taken from: https://stackoverflow.com/a/75314192
(defun add-multiple-into-list (lst items)
  "Add each item from ITEMS into LST."
  (dolist (item items)
    (add-to-list lst item)))
(defun bard/cdlatex-add-math-symbols ()
  "Add functions into list."
  (add-multiple-into-list
   'cdlatex-math-symbol-alist-comb
   '((?V "\\vec"))))
(define-minor-mode bard/org-math-mode
  "Enable features to write math in `org-mode'."
  :init-value nil
  :lighter " S="
  :global nil
  (org-fragtog-mode t)
  (org-cdlatex-mode t)
  (electric-pair-local-mode t)
  (bard/cdlatex-add-math-symbols))
```

3.12.7 Provide library

```
(provide 'bard-writing)
```

3.13 prot-common

This library is mostly used in the `bard-modeline.el` library. It is mostly unmodified from the time when I copied it from Protesilaos's Emacs configuration over 2 years ago.

```
;;; prot-common.el --- Common functions for my dotemacs -*-
   lexical-binding: t -*-
;; Copyright (C) 2020-2023  Protesilaos Stavrou
;; Author: Protesilaos Stavrou <info@protesilaos.com>
;; URL: https://protesilaos.com/emacs/dotemacs
;; Version: 0.1.0
;; Package-Requires: ((emacs "30.1"))
;; This file is NOT part of GNU Emacs.
;; 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:
;;
;; Common functions for my Emacs: <https://protesilaos.com/emacs/
   dotemacs/>.
;;
;; Remember that every piece of Elisp that I write is for my own
;; educational and recreational purposes.  I am not a programmer and
    I
;; do not recommend that you copy any of this if you are not certain
    of
;; what it does.
;;; Code:
(eval-when-compile
  (require 'subr-x)
  (require 'cl-lib))
(defgroup prot-common ()
  "Auxiliary functions for my dotemacs."
  :group 'editing)
;;;###autoload
(defun prot-common-number-even-p (n)
  "Test if N is an even number."
  (if (numberp n)
      (= (% n 2) 0)
    (error "%s is not a number" n)))
;;;###autoload
(defun prot-common-number-integer-p (n)
  "Test if N is an integer."
  (if (integerp n)
      n
    (error "%s is not an integer" n)))
;;;###autoload
(defun prot-common-number-integer-positive-p (n)
  "Test if N is a positive integer."
  (if (prot-common-number-integer-p n)
      (> n 0)
    (error "%s is not a positive integer" n)))
;; Thanks to Gabriel for providing a cleaner version of
;; `prot-common-number-negative': <https://github.com/gabriel376>.
;;;###autoload
(defun prot-common-number-negative (n)
  "Make N negative."
  (if (and (numberp n) (> n 0))
      (* -1 n)
    (error "%s is not a valid positive number" n)))
;;;###autoload
(defun prot-common-reverse-percentage (number percent change-p)
  "Determine the original value of NUMBER given PERCENT.
CHANGE-P should specify the increase or decrease.  For simplicity,
nil means decrease while non-nil stands for an increase.
NUMBER must satisfy `numberp', while PERCENT must be `natnump'."
  (unless (numberp number)
    (user-error "NUMBER must satisfy numberp"))
  (unless (natnump percent)
    (user-error "PERCENT must satisfy natnump"))
  (let* ((pc (/ (float percent) 100))
         (pc-change (if change-p (+ 1 pc) pc))
         (n (if change-p pc-change (float (- 1 pc-change)))))
    ;; FIXME 2021-12-21: If float, round to 4 decimal points.
    (/ number n)))
;;;###autoload
(defun prot-common-percentage-change (n-original n-final)
  "Find percentage change between N-ORIGINAL and N-FINAL numbers.
When the percentage is not an integer, it is rounded to 4
floating points: 16.666666666666664 => 16.667."
  (unless (numberp n-original)
    (user-error "N-ORIGINAL must satisfy numberp"))
  (unless (numberp n-final)
    (user-error "N-FINAL must satisfy numberp"))
  (let* ((difference (float (abs (- n-original n-final))))
         (n (* (/ difference n-original) 100))
         (round (floor n)))
    ;; FIXME 2021-12-21: Any way to avoid the `string-to-number'?
    (if (> n round) (string-to-number (format "%0.4f" n)) round)))
;; REVIEW 2023-04-07 07:43 +0300: I just wrote the conversions from
;; seconds.  Hopefully they are correct, but I need to double check.
(defun prot-common-seconds-to-minutes (seconds)
  "Convert a number representing SECONDS to MM:SS notation."
  (let ((minutes (/ seconds 60))
        (seconds (% seconds 60)))
    (format "%.2d:%.2d" minutes seconds)))
(defun prot-common-seconds-to-hours (seconds)
  "Convert a number representing SECONDS to HH:MM:SS notation."
  (let* ((hours (/ seconds 3600))
         (minutes (/ (% seconds 3600) 60))
         (seconds (% seconds 60)))
    (format "%.2d:%.2d:%.2d" hours minutes seconds)))
;;;###autoload
(defun prot-common-seconds-to-minutes-or-hours (seconds)
  "Convert SECONDS to either minutes or hours, depending on the
   value."
  (if (> seconds 3599)
      (prot-common-seconds-to-hours seconds)
    (prot-common-seconds-to-minutes seconds)))
;;;###autoload
(defun prot-common-rotate-list-of-symbol (symbol)
  "Rotate list value of SYMBOL by moving its car to the end.
Return the first element before performing the rotation.
This means that if `sample-list' has an initial value of `(one
two three)', this function will first return `one' and update the
value of `sample-list' to `(two three one)'.  Subsequent calls
will continue rotating accordingly."
  (unless (symbolp symbol)
    (user-error "%s is not a symbol" symbol))
  (when-let* ((value (symbol-value symbol))
              (list (and (listp value) value))
              (first (car list)))
    (set symbol (append (cdr list) (list first)))
    first))
;;;###autoload
(defun prot-common-empty-buffer-p ()
  "Test whether the buffer is empty."
  (or (= (point-min) (point-max))
      (save-excursion
        (goto-char (point-min))
        (while (and (looking-at "^\\([a-zA-Z]+: ?\\)?$")
                    (zerop (forward-line 1))))
        (eobp))))
;;;###autoload
(defun prot-common-minor-modes-active ()
  "Return list of active minor modes for the current buffer."
  (let ((active-modes))
    (mapc (lambda (m)
            (when (and (boundp m) (symbol-value m))
              (push m active-modes)))
          minor-mode-list)
    active-modes))
;;;###autoload
(defun prot-common-truncate-lines-silently ()
  "Toggle line truncation without printing messages."
  (let ((inhibit-message t))
    (toggle-truncate-lines t)))
;; NOTE 2023-08-12: I tried the `clear-message-function', but it did
;; not work.  What I need is very simple and this gets the job done.
;;;###autoload
(defun prot-common-clear-minibuffer-message (&rest _)
  "Print an empty message to clear the echo area.
Use this as advice :after a noisy function."
  (message ""))
;;;###autoload
(defun prot-common-disable-hl-line ()
  "Disable Hl-Line-Mode (for hooks)."
  (hl-line-mode -1))
;;;###autoload
(defun prot-common-window-bounds ()
  "Return start and end points in the window as a cons cell."
  (cons (window-start) (window-end)))
;;;###autoload
(defun prot-common-page-p ()
  "Return non-nil if there is a `page-delimiter' in the buffer."
  (or (save-excursion (re-search-forward page-delimiter nil t))
      (save-excursion (re-search-backward page-delimiter nil t))))
;;;###autoload
```

```
(defun prot-common-window-small-p ()
  "Return non-nil if window is small.
Check if the `window-width' or `window-height' is less than
`split-width-threshold' and `split-height-threshold',
respectively."
  (or (and (numberp split-width-threshold)
           (< (window-total-width) split-width-threshold))
      (and (numberp split-height-threshold)
           (> (window-total-height) split-height-threshold))))
;;;###autoload
(defun prot-common-three-or-more-windows-p (&optional frame)
  "Return non-nil if three or more windows occupy FRAME.
If FRAME is non-nil, inspect the current frame."
  (>= (length (window-list frame :no-minibuffer)) 3))
;;;###autoload
(defun prot-common-read-data (file)
  "Read Elisp data from FILE."
  (with-temp-buffer
    (insert-file-contents file)
    (read (current-buffer))))
;;;###autoload
(defun prot-common-completion-category ()
  "Return completion category."
  (when-let ((window (active-minibuffer-window)))
    (with-current-buffer (window-buffer window)
      (completion-metadata-get
       (completion-metadata (buffer-substring-no-properties
                             (minibuffer-prompt-end)
                             (max (minibuffer-prompt-end) (point)))
                            minibuffer-completion-table
                            minibuffer-completion-predicate)
       'category))))
;; Thanks to Omar íAntoln Camarena for providing this snippet!
;;;###autoload
(defun prot-common-completion-table (category candidates)
  "Pass appropriate metadata CATEGORY to completion CANDIDATES.
This is intended for bespoke functions that need to pass
completion metadata that can then be parsed by other
tools (e.g. `embark')."
  (lambda (string pred action)
    (if (eq action 'metadata)
        `(metadata (category . ,category))
      (complete-with-action action candidates string pred))))
;;;###autoload
(defun prot-common-completion-table-no-sort (category candidates)
  "Pass appropriate metadata CATEGORY to completion CANDIDATES.
Like `prot-common-completion-table' but also disable sorting."
  (lambda (string pred action)
    (if (eq action 'metadata)
        `(metadata (category . ,category)
                   (display-sort-function . ,#'identity))
      (complete-with-action action candidates string pred))))
;; Thanks to Igor Lima for the `prot-common-crm-exclude-selected-p':
;; <https://github.com/0x462e41>.
;; This is used as a filter predicate in the relevant prompts.
(defvar crm-separator)
;;;###autoload
(defun prot-common-crm-exclude-selected-p (input)
  "Filter out INPUT from `completing-read-multiple'.
Hide non-destructively the selected entries from the completion
table, thus avoiding the risk of inputting the same match twice.
To be used as the PREDICATE of `completing-read-multiple'."
  (if-let* ((pos (string-match-p crm-separator input))
            (rev-input (reverse input))
            (element (reverse
                      (substring rev-input 0
                                 (string-match-p crm-separator
   rev-input))))
            (flag t))
      (progn
        (while pos
          (if (string= (substring input 0 pos) element)
              (setq pos nil)
            (setq input (substring input (1+ pos))
                  pos (string-match-p crm-separator input)
                  flag (when pos t))))
        (not flag))
    t))
;; The `prot-common-line-regexp-p' and `
   prot-common--line-regexp-alist'
;; are contributed by Gabriel: <https://github.com/gabriel376>.
   They
```

```
;; provide a more elegant approach to using a macro, as shown
   further
;; below.
(defvar prot-common--line-regexp-alist
  '((empty . "[\s\t]*$")
    (indent . "^[\s\t]+")
    (non-empty . "^.+$")
    (list . "^\\([\s\t#*+]+\\|[0-9]+[^\s]?[).]+\\)")
    (heading . "^[=-]+"))
  "Alist of regexp types used by `prot-common-line-regexp-p'.")
(defun prot-common-line-regexp-p (type &optional n)
  "Test for TYPE on line.
TYPE is the car of a cons cell in
`prot-common--line-regexp-alist'.  It matches a regular
expression.
With optional N, search in the Nth line from point."
  (save-excursion
    (goto-char (line-beginning-position))
    (and (not (bobp))
         (or (beginning-of-line n) t)
         (save-match-data
           (looking-at
            (alist-get type prot-common--line-regexp-alist))))))
;; The `prot-common-shell-command-with-exit-code-and-output'
   function is
;; courtesy of Harold Carr, who also sent a patch that improved
;; `prot-eww-download-html' (from the `prot-eww.el' library).
;;
;; More about Harold: <http://haroldcarr.com/about/>.
(defun prot-common-shell-command-with-exit-code-and-output (command
   &rest args)
  "Run COMMAND with ARGS.
Return the exit code and output in a list."
  (with-temp-buffer
    (list (apply 'call-process command nil (current-buffer) nil args
   )
          (buffer-string))))
(defvar prot-common-url-regexp
  (concat
   "~?\\<\\([-a-zA-Z0-9+&@#/%?=~_|!:,.;]*\\)"
   "[.@]"
   "\\([-a-zA-Z0-9+&@#/%?=~_|!:,.;]+\\)\\>/?")
  "Regular expression to match (most?) URLs or email addresses.")
(autoload 'auth-source-search "auth-source")
;;;###autoload
(defun prot-common-auth-get-field (host prop)
  "Find PROP in `auth-sources' for HOST entry."
  (when-let ((source (auth-source-search :host host)))
    (if (eq prop :secret)
        (funcall (plist-get (car source) prop))
      (plist-get (flatten-list source) prop))))
;;;###autoload
(defun prot-common-parse-file-as-list (file)
  "Return the contents of FILE as a list of strings.
Strings are split at newline characters and are then trimmed for
negative space.
Use this function to provide a list of candidates for
completion (per `completing-read')."
  (split-string
   (with-temp-buffer
     (insert-file-contents file)
     (buffer-substring-no-properties (point-min) (point-max)))
   "\n" :omit-nulls "[\s\f\t\n\r\v]+"))
(defun prot-common-ignore (&rest _)
  "Use this as override advice to make a function do nothing."
  nil)
;; NOTE 2023-06-02: The `prot-common-wcag-formula' and
;; `prot-common-contrast' are taken verbatim from my `modus-themes'
;; and renamed to have the prefix `prot-common-' instead of
;; `modus-themes-'.  This is all my code, of course, but I do it
   this
;; way to ensure that this file is self-contained in case someone
;; copies it.
;; This is the WCAG formula: <https://www.w3.org/TR/WCAG20-TECHS/G18
   .html>.
(defun prot-common-wcag-formula (hex)
  "Get WCAG value of color value HEX.
The value is defined in hexadecimal RGB notation, such #123456."
  (cl-loop for k in '(0.2126 0.7152 0.0722)
           for x in (color-name-to-rgb hex)
           sum (* k (if (<= x 0.03928)
                        (/ x 12.92)
```

```
                      (expt (/ (+ x 0.055) 1.055) 2.4)))))
;;;###autoload
(defun prot-common-contrast (c1 c2)
  "Measure WCAG contrast ratio between C1 and C2.
C1 and C2 are color values written in hexadecimal RGB."
  (let ((ct (/ (+ (prot-common-wcag-formula c1) 0.05)
               (+ (prot-common-wcag-formula c2) 0.05))))
    (max ct (/ ct))))
;;;; EXPERIMENTAL macros (not meant to be used anywhere)
;; TODO 2023-09-30: Try the same with `cl-defmacro' and &key
(defmacro prot-common-if (condition &rest consequences)
  "Separate the CONSEQUENCES of CONDITION semantically.
Like `if', `when', `unless' but done by using `:then' and `:else'
keywords.  The forms under each keyword of `:then' and `:else'
belong to the given subset of CONSEQUENCES.
- The absence of `:else' means: (if CONDITION (progn CONSEQUENCES)).
- The absence of `:then' means: (if CONDITION nil CONSEQUENCES).
- Otherwise: (if CONDITION (progn then-CONSEQUENCES)
   else-CONSEQUENCES)."
  (declare (indent 1))
  (let (then-consequences else-consequences last-kw)
    (dolist (elt consequences)
      (let ((is-keyword (keywordp elt)))
        (cond
         ((and (not is-keyword) (eq last-kw :then))
          (push elt then-consequences))
         ((and (not is-keyword) (eq last-kw :else))
          (push elt else-consequences))
         ((and is-keyword (eq elt :then))
          (setq last-kw :then))
         ((and is-keyword (eq elt :else))
          (setq last-kw :else)))))
    `(if ,condition
         ,(if then-consequences
              `(progn ,@(nreverse then-consequences))
            nil)
       ,@(nreverse else-consequences))))
(provide 'prot-common)
;;; prot-common.el ends here
```