Emacs Configuration

Saihaj Law

January 5, 2023

Contents

1 Introduction

After reading some of the other amazing configurations written by people like Tecosaur, Shaun Singh, and Sunny Hasja; I decided to upload my own (horrible) **Emacs Configuration**. As a High School student, with an avid interest in **Computer Science** and my class work. This is a **workflow** that I have designed to work well for myself . . . but may not be the best for your workflow! There are lots of bits an pieces cobbled together from many different sources, but I have to admit . . . I forgot half of them! I will do my best to give thanks and credit to those who's code I ~~stole~~ . . . I mean used, but apologies in advance! Well . . . Let's get started.

2 Initial Configuration

As a **Vim** user for most of my life, the first main requirement I had for emacs was that there were **Vim Keybindings**, after some searching I stumbled upon *Doom Emacs* and am currently using that as my daily Emacs driver. Even though I fiddled around with Vanilla Emacs (*to boast my superiority over those greenhorn Spacemacs users*), I in the end decided to return back to Doom because of it's quick runtimes, wonderfully configured EVIL keybindings and also many other things.

There is some initial configuration we have to do in Doom . . . So let's look into it.

2.1 Basic Config

```
(add-to-list 'default-frame-alist '(undecorated . t))

(setq user-full-name "Saihaj Law"
      user-mail-address "laws0817@gmail.com")
```

```
;;(use-package :autothemer)
;;(require 'nano-theme)
;;(require 'ef-themes)
(setq doom-theme 'doom-gruvbox)
(setq +zen-text-scale 0.3)
;;(setq display-line-numbers-type 'relative)

;; (set-frame-parameter (selected-frame) 'alpha '(95 95))
;; (add-to-list 'default-frame-alist '(alpha 95 95))
 (require 'smooth-scroll)
```

Whoa . . . Already we have some messy *lisp*. But just for a short summary . . . I am initializing my personal information, my theme configuration (*catppuccin lovers stand up*) as well as zooming out a bit from **Zen's** overzealous centering. Additionally I am just specifiying my **line numbering** type and (*in the comment region*) is some code for when I feel like making my Emacs Transparent!The autothemer package is there for my catppuccin colour scheme

2.2 Dashboard

I can't be posting my emacs setup on **r/unixporn** when I have a basic doom homepage . . . . That's why I added some stuff to remove most of the startup dashboard's info and instead put an image and the bare minimum amount of textI've edited the Vagabond picture to shrink down and fit my dashboard ... but any picture can likely work

```
(remove-hook '+doom-dashboard-functions #'doom-dashboard-widget-shortmenu)
(add-hook! '+doom-dashboard-functions :append
(setq-hook! '+doom-dashboard-mode-hook evil-normal-state-cursor (list nil))
(setq fancy-splash-image (concat doom-user-dir "vagabond.png")))
```

2.3 Initial Visual Configuration

Now we are getting into the interesting stuff. Some plugins to set up how our frames, modeline, and just general buffers will look/workjust editing the default line spacing, the modeline size, the frame parameters, and the fringe colours

```
(setq-default line-spacing 0.24)
```

```
(modify-all-frames-parameters
'((right-divider-width . 10)
 (internal-border-width . 10)))
(dolist (face '(window-divider
               window-divider-first-pixel
                window-divider-last-pixel))
(face-spec-reset-face face)
(set-face-foreground face (face-attribute 'default :background)))
(set-face-background 'fringe (face-attribute 'default :background))
;;(good-scroll-mode 1)
(unless (equal "Battery status not available"
               (battery))
  (display-battery-mode 1))
(setq centaur-tabs-style "bar")
(setq centaur-tabs-height 32)
(setq centaur-tabs-set-icons t)
(setq centaur-tabs-set-bar 'left)
(setq centaur-tabs-set-modified-marker t)
(after! centaur-tabs
  (setq centaur-tabs-set-bar 'right))
```

Additionally I am using the **Theme-Magic** plugin to make my **Vterm** match the rest of my Emacs Configuration

```
(use-package! theme-magic
  :commands theme-magic-from-emacs
  :config
  (defadvice! theme-magic--auto-extract-16-doom-colors ()
    :override #'theme-magic--auto-extract-16-colors
    (list
     (face-attribute 'default :background)
     (doom-color 'error)
     (doom-color 'success)
     (doom-color 'type)
     (doom-color 'keywords)
     (doom-color 'constants)
     (doom-color 'functions)
     (face-attribute 'default :foreground)
     (face-attribute 'shadow :foreground)
     (doom-blend 'base8 'error 0.1)
     (doom-blend 'base8 'success 0.1)
     (doom-blend 'base8 'type 0.1)
     (doom-blend 'base8 'keywords 0.1)
     (doom-blend 'base8 'constants 0.1)
     (doom-blend 'base8 'functions 0.1)
     (face-attribute 'default :foreground))))
```

Some stuff for ivy posframe

```
(require 'ivy-posframe)
(setq ivy-posframe-display-functions-alist '((t . ivy-posframe-display-at-frame-center)
(ivy-posframe-mode 1)
(setq
  redisplay-dont-pause t
  scroll-margin 1
  scroll-step 1
  scroll-conservatively 10000
  scroll-preserve-screen-position 1)
```

2.4 Nano Stuff

```
;; (defun shaunsingh/apply-nano-theme (appearance)
;;   "Load theme, taking current system APPEARANCE into consideration."
;;   (mapc #'disable-theme custom-enabled-themes)
;;   (pcase appearance
;;     ('light (nano-light))
;;     ('dark (nano-dark))))
;; (use-package nano-theme
;;   :hook (after-init . nano-light)
;;   :config
;;   ;; If emacs has been built with system appearance detection
  ;; add a hook to change the theme to match the system
  ;; (if (boundp 'ns-system-appearance-change-functions)
  ;;     (add-hook 'ns-system-appearance-change-functions #'shaunsingh/apply-nano-theme
  ;; Now to add some missing faces
;;   (custom-set-faces
;;    `(flyspell-incorrect ((t (:underline (:color ,nano-light-salient :style line)))))
;;    `(flyspell-duplicate ((t (:underline (:color ,nano-light-salient :style line)))))

;;    `(git-gutter:modified ((t (:foreground ,nano-light-salient))))
;;    `(git-gutter-fr:added ((t (:foreground ,nano-light-popout))))
;;    ‘(git-gutter-fr:modified ((t (:foreground ,nano-light-salient))))

;;    ‘(lsp-ui-doc-url:added ((t (:background ,nano-light-highlight))))
;;    ‘(lsp-ui-doc-background:modified ((t (:background ,nano-light-highlight))))

;;    ‘(vterm-color-red ((t (:foreground ,nano-light-critical))))
;;    ‘(vterm-color-blue ((t (:foreground ,nano-light-salient))))
;;    ‘(vterm-color-green ((t (:foreground ,nano-light-popout))))
;;    ‘(vterm-color-yellow ((t (:foreground ,nano-light-popout))))
;;    ‘(vterm-color-magenta ((t (:foreground ,nano-light-salient))))

;;    ‘(scroll-bar ((t (:background ,nano-light-background))))
;;    ‘(child-frame-border ((t (:foreground ,nano-light-faded))))

;;    ‘(avy-lead-face-1 ((t (:foreground ,nano-light-subtle))))
;;    ‘(avy-lead-face ((t (:foreground ,nano-light-popout :weight bold))))
;;    ‘(avy-lead-face-0 ((t (:foreground ,nano-light-salient :weight bold))))))
;;    (use-package! nano-modeline
;;      :hook (after-init . nano-modeline-mode)
;;      :config
;;      (setq nano-modeline-prefix 'status
;;            nano-modeline-prefix-padding 1
;;            nano-modeline-position 'bottom))

;; ;;(use-package! minions
 ;; :hook (after-init . minions-mode))

  ;;  (setq-default mode-line-format
   ;;                (cons (propertize "\u200b" 'display '((raise -0.35) (height 1.4))
```

2.5 Initial Functional Configuration

Here we have some stuff that pertains to how my Emacs **works** when I'm on it.I've just begun using xwidget ... but seems like a pretty cool tool!

```
(setq scroll-margin 2
      auto-save-default t
      display-line-numbers-type nil
      delete-by-moving-to-trash t
      truncate-string-ellipsis "..."
      browse-url-browser-function 'xwidget-webkit-browse-url)
(global-subword-mode 1)
;; Time Tracking
(use-package wakatime-mode
  :diminish 'wakatime-mode
  :init
  (add-hook 'prog-mode-hook 'wakatime-mode)
  :config (progn (setq wakatime-cli-path "/usr/local/bin/wakatime")
                 (setq wakatime-api-key "waka_b0c3c9b1-a895-4f1a-8706-f6ce7f52869e")
                 (setq wakatime-python-bin "/usr/local/bin/python")
                 (global-wakatime-mode)))
```

2.6 Minor Modes

I haven't really delved deep into the world of **Minor Modes** in Emacs yet, but I am currently using some of *Prot's* work in my configurationvariable pitch mode to keep my fonts consistent, still trying to debug why my line numbers don't show though...

```
(define-minor-mode prot/variable-pitch-mode
  "Toggle 'mixed-pitch-modei, except for programming modes"
  :init-value nil
  :global nil
  (if prot/variable-pitch-mode
      (unless (derived-mode-p 'prog-mode)
        (variable-pitch-mode 1))
    (variable-pitch-mode -1)))
```

```
(define-minor-mode prot/scroll-center-cursor-mode
  "Toggle centred cursor scrolling behavior"
  :init-value nil
  :lighter " S="
  :global nil
  (if prot/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)))
  )
```

2.7 Mu4e

Not something I use very often … but **Email in Emacs!**I'm using smtp to send my mail and using mu4e to view my emails

```
;; mu4e email
(after! mu4e
  (setq mu4e-index-cleanup nil
        mu4e-index-lazy-check t
        mu4e-update-interval 300)
  (set-email-account! "laws0817"
                      '((mu4e-sent-folder       . "/Sent Mail")
                        (mu4e-drafts-folder     . "/Drafts")
                        (mu4e-trash-folder      . "/Trash")
                        (mu4e-refile-folder     . "/All Mail")
                        (smtpmail-smtp-user     . "laws0817@gmail.com"))))
;; sending with smtpmail
(after! mu4e
  (setq sendmail-program "msmtp"
        send-mail-function #'smtpmail-send-it
        message-sendmail-f-is-evil t
        message-sendmail-extra-arguments '("--read-envelope-from")
        message-send-mail-function #'message-send-mail-with-sendmail))
```

2.8 Modeline

```
(setq doom-modeline-enable-word-count t)
(setq doom-modeline-modal t)
```

2.9 EmacsConf

Just some stuff I'm using for volunteering with **EmacsConf**

```
(use-package subed
  :ensure t
  :config
  ;; Disable automatic movement of point by default
  (add-hook 'subed-mode-hook 'subed-disable-sync-point-to-player)
  ;; Remember cursor position between sessions
  (add-hook 'subed-mode-hook 'save-place-local-mode)
  ;; Break lines automatically while typing
  (add-hook 'subed-mode-hook 'turn-on-auto-fill)
   ;; Break lines at 40 characters
  (add-hook 'subed-mode-hook (lambda () (setq-local fill-column 40))))
```

2.10 EAF

```
(add-to-list 'load-path "~/.emacs.d/site-lisp/emacs-application-framework/")
(require 'eaf)
(require 'eaf-browser)
```

3 Org

Here comes the heavy part of this Emacs configuration ... **ORG**. As a student, a person who programs, and just someone who in general likes writing ... **ORG** mode is indispensable. I have a lot of stuff coming up in here (but hopefully) I explain it enough that it is understandable to everyone including me.

3.1 Org Initial Configuration

DefaultsJust establishing my directories and some other basic configuration

```
(after! org
  (setq org-directory "~/Org"                      ; let's put files here
        org-list-allow-alphabetical t              ; have a. A. a) A) list bullets
        org-use-property-inheritance t             ; it's convenient to have properties
        org-fold-catch-invisible-edits 'smart          ; try not to accidently do weir
        org-log-done 'time                         ; having the time a item is done sou
        org-roam-directory "~/Org/roam/"))         ; same thing, for roam
```

3.2 Org Agenda

Who doesn't want to organize their life, files, and everything else in one place!

3.2.1 Org Agenda Initial Configuration

```
;; org-agenda-config
(after! org-agenda
  (setq org-agenda-files (list "~/org/agenda.org"
                               "~/org/todo.org"))
  (setq org-agenda-window-setup 'current-window
        org-agenda-restore-windows-after-quit t
        org-agenda-show-all-dates nil
        org-agenda-time-in-grid t
        org-agenda-show-current-time-in-grid t
        org-agenda-start-on-weekday 1
        org-agenda-span 7
        org-agenda-tags-column  0
        org-agenda-block-separator nil
        org-agenda-category-icon-alist nil
        org-agenda-sticky t)
  (setq org-agenda-prefix-format
        '((agenda . "%i %?-12t%s")
          (todo .   "%i")
          (tags .   "%i")
          (search . "%i")))
  (setq org-agenda-sorting-strategy
        '((agenda deadline-down scheduled-down todo-state-up time-up
                  habit-down priority-down category-keep)
          (todo   priority-down category-keep)
          (tags   timestamp-up priority-down category-keep)
          (search category-keep))))
```

```
(after! org
  (remove-hook 'org-agenda-finalize-hook '+org-exclude-agenda-buffers-from-workspace-h
  (remove-hook 'org-agenda-finalize-hook
               '+org-defer-mode-in-agenda-buffers-h))
```

3.2.2 Org Agenda Visual Configuration

```
(after! org
  (setq org-agenda-deadline-faces
        '((1.0 . error)
          (1.0 . org-warning)
          (0.5 . org-upcoming-deadline)
          (0.0 . org-upcoming-distant-deadline))))
```

3.3 Org Roam

Org Roam is a wonderful plugin that I use all the time as it really helps me in my academic work, with organization and other wonderful features which I can't find in other Emacs Plugins.

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

(setq org-roam-v2-ack t)
```

3.3.1 Org Roam Visual

```
(use-package! org-roam
  :after org
  :config
  (setq org-roam-v2-ack t)
  (setq org-roam-mode-sections
        (list #'org-roam-backlinks-insert-section
              #'org-roam-reflinks-insert-section
              #'org-roam-unlinked-references-insert-section))
  (org-roam-db-autosync-enable))

(use-package! org-roam-ui
  :after org-roam
  :config
  (setq org-roam-ui-open-on-start nil)
  (setq org-roam-ui-browser-function #'xwidget-webkit-browse-url))

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

 (use-package! org-roam-ui
   :after org-roam
   :commands org-roam-ui-open
   :config
   (setq org-roam-ui-sync-theme t
         org-roam-ui-follow t
         org-roam-ui-update-on-save t
         org-roam-ui-open-on-start t))
 (after! org-roam
 (setq +org-roam-open-buffer-on-find-file nil))
```

3.3.2 Org Roam Capture

I love Org Capture. It is probably my favourite part of Emacs ... I also have it configured with **Emacs Everywhere** as you will see later in the documentation. I have it configured so that each of my classes have their own specific **startup** information which I will later use for other purposesI've reecently gotten into tags in my files so i've added the filetags info as well

```
(after! org-roam
    (setq org-roam-capture-templates
        `(("F" "French" plain "%?"
     :if-new
     (file+head "%<%Y%m%d%H%M%S>-${slug}.org"
      "${title}\n#+filetags:French\n#+LATEX_CLASS:tufte-book\n\n ")
     :unnarrowed t)
        ("D" "Data Management" plain "%?"
     :if-new
     (file+head "%<%Y%m%d%H%M%S>-${slug}.org"
      "${title}\n#+filetags:Data_Management \n#+LATEX_CLASS:tufte-book\n\n ")
     :unnarrowed t)
        ("C" "Computer Engineering" plain "%?"
     :if-new
     (file+head "%<%Y%m%d%H%M%S>-${slug}.org"
      "${title}\n#+filetags:Computer_Engineering\n#+LATEX_CLASS:tufte-book \n\n ")
     :unnarrowed t)
        ("B" "Biology " plain "%?"
     :if-new
     (file+head "%<%Y%m%d%H%M%S>-${slug}.org"
      "${title}\n#+filetags:Biology\n#+LATEX_CLASS:tufte-book\n\n ")
     :unnarrowed t))))
```

3.3.3 Org Capture Pt. 2

I am running an org capture configuration that was created by the wonderful **Tecosaur** which uses the packages *doct* to render the visuals instead. I have just recently begun tinkering with this so this will definitely be updated!I am still working on configuring this so it is somewhat bloated ... but it will be improved!

```
(when (display-graphic-p)
  (require 'all-the-icons))

(use-package! doct
  :defer t
  :commands (doct))

(defun org-capture-select-template-prettier (&optional keys)
  "Select a capture template, in a prettier way than default Lisp programs can force tl
         (or (org-contextualize-keys
              (org-capture-upgrade-templates org-capture-templates)
              org-capture-templates-contexts)
             '(("t" "Task" entry (file+headline "" "Tasks")
                "* TODO %?\n  %u\n  %a")))))
    (if keys
        (or (assoc keys org-capture-templates)
            (error "No capture template referred to by \"%s\" keys" keys))
      (org-mks org-capture-templates
               "Select a capture template\n"
               "Template key: "
               `(("q" ,(concat (all-the-icons-octicon "stop" :face 'all-the-icons-red
(advice-add 'org-capture-select-template :override #'org-capture-select-template-prett

(defun org-mks-pretty (table title &optional prompt specials)

  (save-window-excursion
    (let ((inhibit-quit t)
          (buffer (org-switch-to-buffer-other-window "*Org Select*"))
          (prompt (or prompt "Select: "))
          case-fold-search
          current)
      (unwind-protect
          (catch 'exit
            (while t
              (setq-local evil-normal-state-cursor (list nil))
              (erase-buffer)
              (insert title "\n\n")
              (let ((des-keys nil)
                    (allowed-keys '("\C-g"))
                    (tab-alternatives '("\s" "\t" "\r"))
                    (cursor-type nil))
                ;; Populate allowed keys and descriptions keys
                ;; available with CURRENT selector.
                (let ((re (format "\\`%s\\(.\\)\\'"
                                  (if current (regexp-quote current) "")))
                      (prefix (if current (concat current " ") "")))
                  (dolist (entry table)
                    (pcase entry
                      ;; Description.
                      (`(,(and key (pred (string-match re))) ,desc)
                       (let ((k (match-string 1 key)))
                         (push k des-keys)
                         ;; Keys ending in tab, space or RET are equivalent.
                         (if (member k tab-alternatives)
                             (push "\t" allowed-keys)
                           (push k allowed-keys))
                         (insert (propertize prefix 'face 'font-lock-comment-face) (pro
                      ;; Usable entry.
                      (`(,(and key (pred (string-match re))) ,desc . ,_)
                       (let ((k (match-string 1 key)))
                         (insert (propertize prefix 'face 'font-lock-comment-face) (pro
                         (push k allowed-keys)))
                      (_ nil))))
                ;; Insert special entries, if any.
                (when specials
                  (insert "\n")
                  (pcase-dolist (`(,key ,description) specials)
                    (insert (format "%s   %s\n" (propertize key 'face '(bold all-the-ic
                    (push key allowed-keys)))
                ;; Display UI and let user select an entry or
                ;; a sub-level prefix.
                (goto-char (point-min))
```

```
                (unless (pos-visible-in-window-p (point-max))
                  (org-fit-window-to-buffer))
                (let ((pressed (org--mks-read-key allowed-keys prompt nil)))
                  (setq current (concat current pressed))
                  (cond
                   ((equal pressed "\C-g") (user-error "Abort"))
                   ((equal pressed "ESC") (user-error "Abort"))
                   ;; Selection is a prefix: open a new menu.
                   ((member pressed des-keys))
                   ;; Selection matches an association: return it.
                   ((let ((entry (assoc current table)))
                      (and entry (throw 'exit entry))))
                   ;; Selection matches a special entry: return the
                   ;; selection prefix.
                   ((assoc current specials) (throw 'exit current))
                   (t (error "No entry available")))))))
        (when buffer (kill-buffer buffer))))))
(advice-add 'org-mks :override #'org-mks-pretty)

(setf (alist-get 'height +org-capture-frame-parameters) 15)
;; (alist-get 'name +org-capture-frame-parameters) " Capture") ;; ATM hardcoded in othe
(setq +org-capture-fn
      (lambda ()
        (interactive)
        (set-window-parameter nil 'mode-line-format 'none)
        (org-capture)))

(defun +doct-icon-declaration-to-icon (declaration)
  "Convert :icon declaration to icon"
  (let ((name (pop declaration))
        (set  (intern (concat "all-the-icons-" (plist-get declaration :set))))
        (face (intern (concat "all-the-icons-" (plist-get declaration :color))))
        (v-adjust (or (plist-get declaration :v-adjust) 0.01)))
    (apply set `(,name :face ,face :v-adjust ,v-adjust))))

(defun +doct-iconify-capture-templates (groups)
  "Add declaration's :icon to each template group in GROUPS."
  (let ((templates (doct-flatten-lists-in groups)))
    (setq doct-templates (mapcar (lambda (template)
                                   (when-let* ((props (nthcdr (if (= (length template)
                                  (spec (plist-get (plist-get props :doct
                             (setf (nth 1 template) (concat (+doct-icon-declar
                                                              "\t"
                                                              (nth 1 template)))
                           template)
                         templates))))

(setq doct-after-conversion-functions '(+doct-iconify-capture-templates))


;;(after! org-capture
 ;; (require 'noflet)
  ;;(setq org-capture-templates
   ;;     (doct `(("Todo" :keys "t"
   ;;              :icon ("home" :set "octicon" :color "cyan")
  ;;               :file "~/org/todo.org"
 ;;                :prepend t
     ;;            :headline "Inbox"
    ;;            :template ("* TODO %?"
   ;;                         "%i %a"))
   ;;           ("Agenda" :keys "a"
   ;;            :icon ("business" :set "material" :color "yellow")
   ;;            :file "~/org/agenda.org"
   ;;            :prepend t
   ;;            :headline "Inbox"
   ;;            :template ("* TODO %?"
   ;;                       "SCHEDULED: %^{Schedule:}t"
   ;;                       "DEADLINE: %^{Deadline:}t"
  ;;                        "%i %a"))
  ;;            ("Note" :keys "n"
  ;;             :icon ("sticky-note" :set "faicon" :color "yellow")
  ;;             :file "~/org/notes.org"
  ;;             :template ("* *?"
  ;;                        "%i %a"))
  ;;            ("Journal" :keys "j"
  ;;             :icon ("calendar" :set "faicon" :color "pink")
  ;;             :type plain
  ;;             :function (lambda ()
  ;;                         (org-journal-new-entry t)
  ;;                         (unless (eq org-journal-file-type 'daily)
 ;;                             (org-narrow-to-subtree))
 ;;                           (goto-char (point-max)))
 ;;                 :template "** %(format-time-string org-journal-time-format)%^{Title
 ;;                 :jump-to-captured t
 ;;                 :immediate-finish t)
 ;;                ("Project" :keys "p"
 ;;                 :icon ("repo" :set "octicon" :color "silver")
 ;;                 :prepend t
 ;;                 :type entry
 ;;                 :headline "Inbox"
 ;;                 :template ("* %{keyword} %?"
 ;;                            "%i"
 ;;                            "%a")
 ;;                 :file ""
 ;;                 :custom (:keyword "")
  ;;                :children (("Task" :keys "t"
 ;;                              :icon ("checklist" :set "octicon" :color "green")
;;                            :keyword "TODO"
;;                             :file +org-capture-project-todo-file)
;;                            ("Note" :keys "n"
;;                             :icon ("sticky-note" :set "faicon" :color "yellow")
;;                             :keyword "%U"
;;                             :file +org-capture-project-notes-file)))))))
```

3.3.4 Org Capture Everywhere

Since I am a MacOs user I have a configuration made (using the **Automation** tool) that allows me to call **Org-Capture** from anywhere! Yippie!

```
;;(require 'noflet)
;;(defun timu-func-make-capture-frame ()
;;  "Create a new frame and run ‘org-capture’."
;;  (interactive)
;;  (make-frame '((name . "capture")
;;                (top . 300)
;;                (left . 700)
;;                (width . 80)
;;                (height . 25)))
;;  (select-frame-by-name "capture")
;;  (delete-other-windows)
;;  (noflet ((switch-to-buffer-other-window (buf) (switch-to-buffer buf)))
;;           (org-capture)))
;;(defadvice org-capture-finalize
;;    (after delete-capture-frame activate)
;;  "Advise capture-finalize to close the frame."
;;  (if (equal "capture" (frame-parameter nil 'name))
;;      (delete-frame)))
;;(defadvice org-capture-destroy
;;    (after delete-capture-frame activate)
;;  "Advise capture-destroy to close the frame."
;;  (if (equal "capture" (frame-parameter nil 'name))
;;      (delete-frame)))
```

3.4 ORG Visual Configuration

3.4.1 Org Modern

One of the first **Org plugins** I have is **ORG MODERN** which is a wonderful plugin that helps hide and clean up Org buffers, the configuration is copied straight from the documentation and works wonderfully for me.This gives me those beautiful ToDo's and Done's

```
;; org modern
(setq ;; Edit settings
 org-auto-align-tags nil
 org-tags-column 0
 org-fold-catch-invisible-edits 'show-and-error
 org-special-ctrl-a/e t
 org-insert-heading-respect-content t

 ;; Org styling, hide markup etc.
 org-hide-emphasis-markers t
 org-pretty-entities t
 org-ellipsis "..."

 ;; Agenda styling
 org-agenda-tags-column 0
 org-agenda-block-separator ?
 org-agenda-time-grid
 '((daily today require-timed)
   (800 1000 1200 1400 1600 1800 2000)
   "  " "")
 org-agenda-current-time-string
 " now ")
(global-org-modern-mode)
```

3.4.2 SVG-Tag Mode

This is a wonderful package created by the amazing **Rougier** (*check out his stuff if you like aesthetic Emacs*) which gives us beautifully rendered **Tags** inside of our org filesit's made by Rougier ... it's going to look good

```
(use-package svg-tag-mode
  :commands svg-tag-mode
  :config
  (defconst date-re "[0-9]\\{4\\}-[0-9]\\{2\\}-[0-9]\\{2\\}")
  (defconst time-re "[0-9]\\{2\\}:[0-9]\\{2\\}")
  (defconst day-re "[A-Za-z]\\{3\\}")
  (defconst day-time-re (format "\\(%s\\)? ?\\(%s\\)?" day-re time-re))
  (defun svg-progress-percent (value)
    (svg-image (svg-lib-concat
                (svg-lib-progress-bar (/ (string-to-number value) 100.0)
                                  nil :margin 0 :stroke 2 :radius 3 :padding 2 :width
                (svg-lib-tag (concat value "%")
                             nil :stroke 0 :margin 0)) :ascent 'center))

  (defun svg-progress-count (value)
    (let* ((seq (mapcar #'string-to-number (split-string value "/")))
           (count (float (car seq)))
           (total (float (cadr seq))))
    (svg-image (svg-lib-concat
                (svg-lib-progress-bar (/ count total) nil
                                      :margin 0 :stroke 2 :radius 3 :padding 2 :width
                (svg-lib-tag value nil
                             :stroke 0 :margin 0)) :ascent 'center)))

  (setq svg-tag-tags
        `(
          ;; Org tags
          (":\\([A-Za-z0-9]+\\)" . ((lambda (tag) (svg-tag-make tag))))
          (":\\([A-Za-z0-9]+[ \-]\\)" . ((lambda (tag) tag)))
```

```
;; Task priority
("\\[#[A-Z]\\]" . ( (lambda (tag)
                      (svg-tag-make tag :face 'org-priority
                                    :beg 2 :end -1 :margin 0))))

;; Progress
("\\(\\[[0-9]\\{1,3\\}%\\]\\)" . ((lambda (tag)
                                    (svg-progress-percent (substring tag 1 -2
("\\(\\[[0-9]+/[0-9]+\\]\\)" . ((lambda (tag)
                                  (svg-progress-count (substring tag 1 -1)))

;; TODO / DONE
("TODO" . ((lambda (tag) (svg-tag-make "TODO" :face 'org-todo :inverse t :mar
("DONE" . ((lambda (tag) (svg-tag-make "DONE" :face 'org-done :margin 0))))


;; Citation of the form [cite:@Knuth:1984]
("\\(\\[cite:@[A-Za-z]+:\\)" . ((lambda (tag)
                                  (svg-tag-make tag
                                                :inverse t
                                                :beg 7 :end -1
                                                :crop-right t))))
("\\[cite:@[A-Za-z]+:\\([0-9]+\\]\\)" . ((lambda (tag)
                                           (svg-tag-make tag
                                                         :end -1
                                                         :crop-left t))))


;; Active date (with or without day name, with or without time)
(,(format "\\(<%s>\\)" date-re) .
 ((lambda (tag)
    (svg-tag-make tag :beg 1 :end -1 :margin 0))))
(,(format "\\(<%s \\)%s>" date-re day-time-re) .
 ((lambda (tag)
    (svg-tag-make tag :beg 1 :inverse nil :crop-right t :margin 0))))
(,(format "<%s \\(%s>\\)" date-re day-time-re) .
 ((lambda (tag)
    (svg-tag-make tag :end -1 :inverse t :crop-left t :margin 0))))

;; Inactive date  (with or without day name, with or without time)
```

```
           (,(format "\\(\\[%s\\]\\)" date-re) .
            ((lambda (tag)
               (svg-tag-make tag :beg 1 :end -1 :margin 0 :face 'org-date))))
           (,(format "\\(\\[%s \\)%s\\]" date-re day-time-re) .
            ((lambda (tag)
               (svg-tag-make tag :beg 1 :inverse nil :crop-right t :margin 0 :face 'org
           (,(format "\\[%s \\(%s\\]\\)" date-re day-time-re) .
            ((lambda (tag)
               (svg-tag-make tag :end -1 :inverse t :crop-left t :margin 0 :face 'org-
```

3.4.3 Ligatures

I have some custom ligatures to simplify how all my *"Org Code"* looks in my buffers

```
(after! org
(setq org-ellipsis "  ")
  (appendq! +ligatures-extra-symbols
          `(:checkbox      ""
            :pending       ""
            :checkedbox    ""
            :list_property ""
            :em_dash       "-"
            :ellipses      "..."
            :arrow_right   "→"
            :arrow_left    "←"
            :title        " "
            :subtitle      ""
            :author        ""
            :date          ""
            :property      ""
            :options       ""
            :startup       ""
            :macro         ""
            :html_head     ""
            :html          ""
            :latex_class   ""
            :latex_header  ""
            :beamer_header ""
            :latex         ""
            :attr_latex    ""
            :attr_html     ""
            :attr_org      ""
            :begin_quote   ""
            :end_quote     ""
            :caption       ""
            :header        ">"
            :results       ""
            :begin_export  ""
            :end_export    ""
            :properties    ""
            :end           ""
            :priority_a   ,(propertize "" ’face ’all-the-icons-red)
            :priority_b   ,(propertize "" ’face ’all-the-icons-orange)
            :priority_c   ,(propertize "" ’face ’all-the-icons-yellow)
            :priority_d   ,(propertize "" ’face ’all-the-icons-green)
            :priority_e   ,(propertize "" ’face ’all-the-icons-blue)
            :roam_tags nil
            :filetags nil))
(set-ligatures! ’org-mode
  :merge t
  :checkbox      "[ ]"
  :pending       "[-]"
  :checkedbox    "[X]"
  :list_property "::"
  :em_dash       "---"
  :ellipsis      "..."
  :arrow_right   "->"
  :arrow_left    "<-"
  :title         "#+title:"
  :subtitle      "#+subtitle:"
  :author        "#+author:"
  :date          "#+date:"
  :property      "#+property:"
  :options       "#+options:"
  :startup       "#+startup:"
  :macro         "#+macro:"
  :html_head     "#+html_head:"
  :html          "#+html:"
  :latex_class   "#+latex_class:"
  :latex_header  "#+latex_header:"
  :beamer_header "#+beamer_header:"
  :latex         "#+latex:"
  :attr_latex    "#+attr_latex:"
  :attr_html     "#+attr_html:"
  :attr_org      "#+attr_org:"
  :begin_quote   "#+begin_quote"
  :end_quote     "#+end_quote"
  :caption       "#+caption:"
  :header        "#+header:"
  :begin_export  "#+begin_export"
  :end_export    "#+end_export"
  :results       "#+RESULTS:"
  :property      ":PROPERTIES:"
  :end           ":END:"
  :priority_a    "[#A]"
  :priority_b    "[#B]"
  :priority_c    "[#C]"
  :priority_d    "[#D]"
  :priority_e    "[#E]"
  :roam_tags     "#+roam_tags:"
  :filetags      "#+filetags:")
(plist-put +ligatures-extra-symbols :name "")
)
```

3.4.4 Fonts

I have my fonts fairly simply configured ... I am using **ETBembo** (wonderful font btw) throughout my configuration and have it change in size based on the heading levels.

```
    (custom-theme-set-faces
     'user
     `(org-level-4 ((t (:height 0.9))))
     `(org-level-3 ((t (:height 1.15 :inherit nano-popout))))
     `(org-level-2 ((t (:height 1.3 :inherit nano-popout))))
     `(org-level-1 ((t (:height 1.45 :inherit nano-salient))))
     `(org-document-title ((t (:height 1.7 :underline t :inherit nano-salient)))))

;;(set-face-attribute 'default nil :font "IBM 3270" :height 160 :weight normal)
```

```
(setq doom-font (font-spec :family "FiraCode Nerd Font" :size 12))
(set-face-attribute 'fixed-pitch nil :family "IBM 3270" :height 160)
(set-face-attribute 'variable-pitch nil :family "Ogg" :height 160)
(add-hook 'org-mode-hook 'variable-pitch-mode)
```

3.4.5 Org Functional Visuals

Here I am adding the visuals which are pertinent to how my Org will look when opening it up ... and just the general settings for my ORG files

```
(after! org
    (setq org-src-fontify-natively t
    org-fontify-whole-heading-line t
    org-pretty-entities t
    org-ellipsis "  " ;; folding symbol
    org-hide-emphasis-markers t
    org-agenda-block-separator ""
    org-fontify-done-headline t
    prot/scroll-center-cursor-mode t
    org-fontify-quote-and-verse-blocks t
    org-startup-with-inline-images t
    org-startup-indented t))

    (lambda () (progn
      (setq left-margin-width 2)
      (setq right-margin-width 2)
      (set-window-buffer nil (current-buffer))))
(setq header-line-format " ")
(add-hook 'org-mode-hook
          (lambda ()
            (font-lock-add-keywords
             nil
             '(("^-\\{5,\\}"  0 '(:foreground "purple" :weight bold))))))

(require 'ink)
```

3.4.6 Olivetti Mode

```
(use-package! olivetti
  :after org
  ;:hook (olivetti-mode . double-header-line-mode)
  :config
    (setq olivetti-min-body-width 50
          olivetti-body-width 130
          olivetti-style 'fancy ; fantastic new layout
          olivetti-margin-width 12)
    (add-hook! 'olivetti-mode-hook (window-divider-mode -1))
    (add-hook! 'olivetti-mode-hook (set-face-attribute 'window-divider nil :foreground
    (add-hook! 'olivetti-mode-hook (set-face-attribute 'vertical-border nil :foregroun
    )
```

3.5 LaTeX Configuration

I always export the work I am handing in to teachers and printing off into

$$\LaTeX$$

(*I know Lamport is happy with that typesetting*) and so I have some basic configuration going on in hereI personally use the tufte-latex classes as they look beautiful, but any classes can be added below

```
(with-eval-after-load 'ox
  (require 'ox-hugo))

;; Tufte Latex Classes

(with-eval-after-load 'ox-latex
(add-to-list 'org-latex-classes
'("tuftebook"
"\\documentclass{tufte-book}\n
\\usepackage{color}
\\usepackage{amssymb}
\\usepackage{gensymb}
\\usepackage{nicefrac}
\\usepackage{units}"
("\\section{%s}" . "\\section*{%s}")
("\\subsection{%s}" . "\\subsection*{%s}")
("\\paragraph{%s}" . "\\paragraph*{%s}")
("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
 ;; tufte-handout class for writing classy handouts and papers
(with-eval-after-load 'ox-latex
(add-to-list 'org-latex-classes
'("tuftehandout" "\\documentclass{tufte-handout}
\\usepackage{color}
\\usepackage{amssymb}
\\usepackage{amsmath}
\\usepackage{gensymb}
\\usepackage{nicefrac}
\\usepackage{units}"
("\\section{%s}" . "\\section*{%s}")
("\\subsection{%s}" . "\\subsection*{%s}")
("\\paragraph{%s}" . "\\paragraph*{%s}")
("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
(with-eval-after-load 'ox-latex
(add-to-list 'org-latex-classes
                   '("rbt-mathnotes-formula-sheet"
                   " \\documentclass[]{rbt-mathnotes-formula-sheet}")))

(with-eval-after-load 'ox-latex
(add-to-list 'org-latex-classes
                   '("rbt-mathnotes"
                   " \\documentclass[]{rbt-mathnotes}")))

(with-eval-after-load 'ox-latex
(add-to-list 'org-latex-classes
                   '("rbt-mathnotes-hw"
                   " \\documentclass[]{rbt-mathnotes-hw}")))

(with-eval-after-load 'ox-latex
(add-to-list 'org-latex-classes
'("lectures"
"\\documentclass[english]{lectures}\n"
("\\section{%s}" . "\\section*{%s}")
("\\subsection{%s}" . "\\subsection*{%s}")
("\\paragraph{%s}" . "\\paragraph*{%s}")
("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))

(with-eval-after-load 'ox-latex
(add-to-list 'org-latex-classes
'("math_lectures"
"\\documentclass[]{report}\n"
("\\section{%s}" . "\\section*{%s}")
("\\subsection{%s}" . "\\subsection*{%s}")
("\\paragraph{%s}" . "\\paragraph*{%s}")
("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
(with-eval-after-load 'org
  (plist-put org-format-latex-options :background 'default))
```

3.5.1 Org-Noter

I also often use Org-Noter as well to annotate my PDF's

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

3.6 Emacs-Calc

Not really part of the whole **ORG** section, but I often use it when I am working with Org-Files . . . . So here it is.Calc

```
;; CALC mode
(map! :map calc-mode-map
      :after calc
      :localleader
      :desc "Embedded calc (toggle)" "e" #'calc-embedded)
(map! :map org-mode-map
      :after org
      :localleader
      :desc "Embedded calc (toggle)" "E" #'calc-embedded)
(map! :map latex-mode-map
      :after latex
      :localleader
      :desc "Embedded calc (toggle)" "e" #'calc-embedded)
(defvar calc-embedded-trail-window nil)
(defvar calc-embedded-calculator-window nil)

(defadvice! calc-embedded-with-side-pannel (&rest _)
  :after #'calc-do-embedded
  (when calc-embedded-trail-window
    (ignore-errors
      (delete-window calc-embedded-trail-window))
    (setq calc-embedded-trail-window nil))
  (when calc-embedded-calculator-window
    (ignore-errors
      (delete-window calc-embedded-calculator-window))
    (setq calc-embedded-calculator-window nil))
  (when (and calc-embedded-info
             (> (* (window-width) (window-height)) 1200))
    (let ((main-window (selected-window))
          (vertical-p (> (window-width) 80)))
      (select-window
       (setq calc-embedded-trail-window
             (if vertical-p
                 (split-window-horizontally (- (max 30 (/ (window-width) 3))))
               (split-window-vertically (- (max 8 (/ (window-height) 4)))))))
      (switch-to-buffer "*Calc Trail*")
      (select-window
       (setq calc-embedded-calculator-window
             (if vertical-p
                 (split-window-vertically -6)
               (split-window-horizontally (- (/ (window-width) 2))))))
      (switch-to-buffer "*Calculator*")
      (select-window main-window))))
```

3.7 NLP/GPT

```
(require 'gpt)
(setq gpt-openai-key "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")
```