README.org

An Academic's Opinionated DOOM Emacs Config

About This Config

This config has evolved over the last few months to reflect my needs as an academic. I initially started with emacs to take notes on papers. I eventually discovered Org-Roam and its supporting packages `org-roam-bibtex` and `org-roam-server`. I hope this config can be useful to other academics who are starting out with DOOM emacs and want to replicate my workflow or are looking to incorporate some of the packages I use into their own configs. The config has since then grown to include `org-super-agenda` and `org-journal` which are two of the packages that I use to keep on track with my commitments and tasks.

Move to GCC emacs

One of the things that irritated me the most about emacs was how slow it was when the text in one of the files got very long. If a line (paragraphs are considered one line in emacs) gets too long, emacs starts to slow down till you start a new paragraph. As someone who does writing for a living, taking notes, inserting citations, this starts to get very annoying. Upon conversation with the creator of DOOM emacs, I installed GCC emacs. There are a bunch of different ways to do this, but I followed the amazing gist written by mjlbach. When I ran into trouble he took time out to help me figure it out. Much thanks to him. As a result, my emacs runs as fast as it ever has.

What is the workflow:

I have made a couple of posts on reddit about my notetaking. You can find them here and here. This particular post has short gifs on my workflow and how everything ties together.

The Preambles

This section of the config takes care of some of the housekeeping, and default behavior I want in my emacs.Huge thanks to Tecosaur for allowing me to copy large chunks of his setup.

Name and Email

```
;;; $DOOMDIR/config.el -*- lexical-binding: t; -*-
(setq user-full-name "Sunny Hasija"
      user-mail-address "hasija.4@osu.edu")
```

Rudimentary Settings

```
(setq-default
 delete-by-moving-to-trash t                      ; Delete files to trash
 tab-width 4                                                          ; Set wid
 uniquify-buffer-name-style 'forward      ; Uniquify buffer names
 window-combination-resize t                   ; take new window space from a
 x-stretch-cursor t)                                                  ; Stretch curso

(setq undo-limit 80000000                          ; Raise undo-limit to 80Mb
      evil-want-fine-undo t                             ; By default while in
      auto-save-default t                                    ; Nobody likes to
      inhibit-compacting-font-caches t      ; When there are lots of glyphs, k
      truncate-string-ellipsis "…")                ; Unicode ellispis are nicer

(delete-selection-mode 1)                              ; Replace selection when
(display-time-mode 1)                                    ; Enable time in the m
```

```
(global-subword-mode 1)                         ; Iterate through CamelCase
(setq line-spacing 0.3)                                 ; seems like a nice
```

Battery Display

I use the same emacs config on my desktop and my laptop. I added a little snippet that shows battery level on my laptop.

```
(unless (equal "Battery status not available"
               (battery))
  (display-battery-mode 1))                          ; On laptops it's nice t
```

Full Screen Emacs

I like to emacs to be fullscreen when I open it for the first time.

```
(if (eq initial-window-system 'x)                 ; if started by emacs comman
    (toggle-frame-maximized)
  (toggle-frame-fullscreen))
```

Windows Behavior

The following code allows me to choose which buffer I want to see when I split a window. It first does a split to the right, and then opens Ivy and and shows me a preview.

```
(setq evil-vsplit-window-right t
      evil-split-window-below t)
(defadvice! prompt-for-buffer (&rest _)
  :after '(evil-window-split evil-window-vsplit)
  (+ivy/switch-buffer))
(setq +ivy-buffer-preview t)
```

Modeline

I expect most of the documents I work on to be UTF - 8, So I don’t want to see that taking up space unless the encoding is something different

```
(defun doom-modeline-conditional-buffer-encoding ()
  (setq-local doom-modeline-buffer-encoding
```

```
                (unless (or (eq buffer-file-coding-system 'utf-8-unix)
                            (eq buffer-file-coding-system 'utf-8)))))
  (add-hook 'after-change-major-mode-hook #'doom-modeline-conditional-buffer-enc
```

Windows Layout

I like being able to rotate the windows, and this functionality already exists within DOOM under `SPC w r` and `SPC w R`. Layout rotation is also preferable, so I added this under `SPC w SPC`

```
(map! :map evil-window-map
      "SPC" #'rotate-layout
      "<left>"     #'evil-window-left
       "<down>"     #'evil-window-down
       "<up>"       #'evil-window-up
       "<right>"    #'evil-window-right
       ;; Swapping windows
       "C-<left>"       #'+evil/window-move-left
       "C-<down>"       #'+evil/window-move-down
       "C-<up>"         #'+evil/window-move-up
       "C-<right>"      #'+evil/window-move-right)
```

Windows Title

The following shows just the buffer name and if applicable, the project folder. Moreover, the snippet also gives a visual indication if the file is modifed or not.

```
(setq frame-title-format
    '(""
      (:eval
       (if (s-contains-p org-roam-directory (or buffer-file-name ""))
           (replace-regexp-in-string ".*/[0-9]*-?" "◁ " buffer-file-name)
         "%b"))
      (:eval
       (let ((project-name (projectile-project-name)))
         (unless (string= "-" project-name)
           (format (if (buffer-modified-p)  " ◉ %s" " ● %s") project-name))))
```

Splash Screen

After using DOOM for a while, I decided spruce up the splash screen. I tried many things, including changing the splash images, but because I am running gccEmacs, I cannot get the transparency in png's to work. So the result is a white background. However, I did discover a cool ascii CLI tool, that I end up using to create random splash screens when I load up DOOM. You need to install the CLI tool first (and it is available here). Then you can run the following code

```
(defvar +fl/splashcii-query ""
  "The query to search on asciiur.com")

(defun +fl/splashcii ()
  (split-string (with-output-to-string
                  (call-process "splashcii" nil standard-output nil +fl/splash
                "\n" t))

(defun +fl/doom-banner ()
  (let ((point (point)))
    (mapc (lambda (line)
            (insert (propertize (+doom-dashboard--center +doom-dashboard--widt
                                'face 'doom-dashboard-banner) " ")
            (insert "\n"))
          (+fl/splashcii))
    (insert (make-string (or (cdr +doom-dashboard-banner-padding) 0) ?\n))))

;; override the first doom dashboard function
(setcar (nthcdr 0 +doom-dashboard-functions) #'+fl/doom-banner)

(setq +fl/splashcii-query "space")
```

Fonts

Doom exposes five (optional) variables for controlling fonts in Doom. Here are the three important ones:

- `doom-font'
- `doom-variable-pitch-font'
- `doom-big-font' – used for `doom-big-font-mode'; use this for presentations or streaming.

They all accept either a font-spec, font string ("Input Mono-12"), or xlfd font string. You generally only need these two:

```
(setq doom-font (font-spec :family "Iosevka Term SS04" :size 16)
           doom-big-font (font-spec :family "Iosevka Term SS04" :size 36)
           ;doom-variable-pitch-font (font-spec :family "ETBembo" :size 24)
```

```
;doom-serif-font (font-spec :family "ETBembo" :size 24)
)
```

Theme

There are two ways to load a theme. Both assume the theme is installed and available. You can either set `doom-theme' or manually load a theme with the `load-theme' function. I like gruvbox light as it is very easy to read and, to me atleast, looks to be helpful for doing longform writing. I am also partial to zaiste's theme and I toggle between the two.

Note: Added doom-flatwhite-theme.elisp

There is a cool new theme that has syntax highlighting, and has a nice light background. The theme file is available in the *theme* folder.

```
;;light themes
;(setq doom-theme 'doom-gruvbox-light)
;(setq doom-theme 'zaiste)
;(setq doom-theme 'doom-flatwhite)
;;dark themes
(setq doom-theme 'doom-palenight)
```

I also like to see the line numbers. `display-line-numbers-type` controls this functionality. If set to `nil', line numbers are disabled. For relative line numbers, set this to `relative'.

```
(setq display-line-numbers-type t)
```

Use Proportional Fonts

```
(defun my-buffer-face-mode-variable ()
  "Set font to a variable width (proportional) fonts in current buffer"
  (interactive)
  (setq buffer-face-mode-face '(:family "Iosevka Term SS04" :height 100 ))
  (buffer-face-mode))
(add-hook 'org-mode-hook 'my-buffer-face-mode-variable)
```

Org Directory

If you use `org' and don't want your org files in the default location below, change `org-directory'. It must be set before org loads! My org directory lives in dropbox, so that it is accessible from any of my computers. Also helps in backing everything up.

```
(setq org-directory "~/Dropbox/Org/")
```

Packages

Here are some additional functions/macros that could help you configure Doom:

- `load!' for loading external *.el files relative to this one
- `use-package' for configuring packages
- `after!' for running code after a package has loaded
- `add-load-path!' for adding directories to the `load-path', relative to `config.el`. Emacs searches the `load-path' when you load packages with `require' or `use-package'.
- `map!' for binding new keys

To get information about any of these functions/macros, move the cursor over the highlighted symbol at press 'K' (non-evil users must press 'C-c g k'). This will open documentation for it, including demos of how they are used. You can also try 'gd' (or 'C-c g d') to jump to their definition and see how they are implemented.

Org-Ref

I initially started using this package in order to manage my citations. However, I soon found that this package required a lot of manual work for my liking. I still keep it in my config as it allows me to use citations in org-documents, if I am working on them directly. I can output these org-documents later to pdf or .docx later.

```
(use-package! org-ref
    :after org
    :init
    ; code to run before loading org-ref
    :config
    ; code to run after loading org-ref
    )
(setq org-ref-notes-directory "~/Dropbox/Org/references/notes"
     ; org-ref-bibliography-notes "~/Dropbox/Org/references/articles.org" ;; n
      org-ref-default-bibliography '("~/Dropbox/Org/references/library.bib")
      org-ref-pdf-directory "~/Dropbox/Zotero/")

(after! org
  (add-to-list 'org-capture-templates
               '(("a"              ; key
                  "Article"        ; name
                  entry            ; type
                  (file+headline "~/Dropbox/Org/phd.org" "Article")  ; target
                  "\* %^{Title} %(org-set-tags)  :article: \n:PROPERTIES:\n:Cr
```

```
                    :prepend t        ; properties
                    :empty-lines 1    ; properties
                    :created t        ; properties
  ))) )
```

Helm-Bibtex

This is the jumping off point in my workflow inside emacs. The config gives the directory of where the PDFs are stored, where I want my notes to be stored, and where the Bibtex file is.

```
(use-package! helm-bibtex
  :after org
  :init
  ; blah blah
  :config
  ;blah blah
  )

(setq bibtex-format-citation-functions
      '((org-mode . (lambda (x) (insert (concat
                                         "\\cite{"
                                         (mapconcat 'identity x ",")
                                         "}")) ""))))
(setq
      bibtex-completion-pdf-field "file"
      bibtex-completion-bibliography
      '("~/Dropbox/Org/references/library.bib")
      bibtex-completion-library-path '("~/Dropbox/Zotero/")
     ; bibtex-completion-notes-path "~/Dropbox/Org/references/articles.org"  ;
      )
```

Zotxt

Allows for syncing of the notes between zotero and emacs.

Note: This package only seems to load initial notes into emacs - Probably not needed anymore.

```
(use-package! zotxt
  :after org)
;(add-to-list 'load-path (expand-file-name "ox-pandoc" starter-kit-dir))
```

Ox-Pandoc

This is a great package that I use to output org files to different formats.

```
(use-package! ox-pandoc
  :after org)
;; default options for all output formats
(setq org-pandoc-options '((standalone . _)))
;; cancel above settings only for 'docx' format
(setq org-pandoc-options-for-docx '((standalone . nil)))
;; special settings for beamer-pdf and latex-pdf exporters
(setq org-pandoc-options-for-beamer-pdf '((pdf-engine . "xelatex")))
(setq org-pandoc-options-for-latex-pdf '((pdf-engine . "pdflatex")))
;; special extensions for markdown_github output
(setq org-pandoc-format-extensions '(markdown_github+pipe_tables+raw_html))
```

ORB : Org-Roam-Bibtex

This fantastic package allows me to use my bibtex file to create and take notes and store them in a zettlekasten.

```
(use-package! org-roam-bibtex
  :load-path "~/Dropbox/Org/references/library.bib" ;Modify with your own path
  :hook (org-roam-mode . org-roam-bibtex-mode)
  :bind (:map org-mode-map
         (("C-c n a" . orb-note-actions))))
(setq orb-templates
      '(("r" "ref" plain (function org-roam-capture--get-point) ""
         :file-name "${citekey}"
         :head "#+TITLE: ${citekey}: ${title}\n#+ROAM_KEY: ${ref}\n" ; <--
         :unnarrowed t)))
(setq orb-preformat-keywords   '(("citekey" . "=key=") "title" "url" "file" "a

(setq orb-templates
      '(("n" "ref+noter" plain (function org-roam-capture--get-point)
         ""
         :file-name "${slug}"
         :head "#+TITLE: ${citekey}: ${title}\n#+ROAM_KEY: ${ref}\n#+ROAM_TAGS

- tags ::
- keywords :: ${keywords}
\* ${title}
:PROPERTIES:
:Custom_ID: ${citekey}
:URL: ${url}
:AUTHOR: ${author-or-editor}
:NOTER_DOCUMENT: %(orb-process-file-field \"${citekey}\")
:NOTER_PAGE:
:END:")))
```

Org Roam

Fantastic package, the heart of my note taking. This is an implementation of the zettlekasten method inspired by Roam Research. Used in conjunction with `org-roam-bibtex` and `org-roam-server` is central to my workflow.

```
; org-roam settings
(setq org-roam-directory "~/Dropbox/Org/references/notes")
(after! org-roam
        (map! :leader
            :prefix "n"
            :desc "org-roam" "l" #'org-roam
            :desc "org-roam-insert" "i" #'org-roam-insert
            :desc "org-roam-switch-to-buffer" "b" #'org-roam-switch-to-buffer
            :desc "org-roam-find-file" "f" #'org-roam-find-file
            :desc "org-roam-show-graph" "g" #'org-roam-show-graph
            :desc "org-roam-insert" "i" #'org-roam-insert
            :desc "org-roam-capture" "c" #'org-roam-capture))
(after! org-roam
      (setq org-roam-ref-capture-templates
            '(("r" "ref" plain (function org-roam-capture--get-point)
               "%?"
               :file-name "websites/${slug}"
               :head "#+TITLE: ${title}
    #+ROAM_KEY: ${ref}
    - source :: ${ref}"
               :unnarrowed t))))  ; capture template to grab websites. Require
```

Org-Journal

I was tired of using one todo file, and given the temporal nature of tasks, I decided to incorporate org-journal into my workflow. I create a daily note for each day and mark tasks and schedule them as they arise. Moreover, I also keep a running track of all thoughts and notes I might have during a meeting.

```
;; org-journal the DOOM way
(use-package org-journal
  :init
  (setq org-journal-dir "~/Dropbox/Org/Daily/"
        org-journal-date-prefix "#+TITLE: "
        org-journal-file-format "%Y-%m-%d.org"
        org-journal-date-format "%A, %d %B %Y")
  :config
  (setq org-journal-find-file #'find-file-other-window )
  (map! :map org-journal-mode-map
        "C-c n s" #'evil-save-modified-and-close )
  )
```

```
(setq org-journal-enable-agenda-integration t)
```

Deft

Allows me to quickly search through recently created org-roam files. Configured to only look into my roam folder.

```
(use-package deft
      :after org
      :bind
      ("C-c n d" . deft)
      :custom
      (deft-recursive t)
      (deft-use-filter-string-for-filename t)
      (deft-default-extension "org")
      (deft-directory "~/Dropbox/Org/references/notes/"))
```

Org-Roam-Server

Allows me to see my org-roam-graph. This is a fantastic package and I have my roam - server almost always open as I take notes.

```
(use-package! org-roam-server
  :after org-roam
  :config
  (setq org-roam-server-host "127.0.0.1"
        org-roam-server-port 8080
        org-roam-server-export-inline-images t
        org-roam-server-authenticate nil
        org-roam-server-label-truncate t
        org-roam-server-label-truncate-length 60
        org-roam-server-label-wrap-length 20)
  (defun org-roam-server-open ()
    "Ensure the server is active, then open the roam graph."
    (interactive)
    (org-roam-server-mode 1)
    (browse-url-xdg-open (format "http://localhost:%d" org-roam-server-port)))
(after! org-roam
  (org-roam-server-mode))
```

Org-Download

Allows me to download screenshots and images. Work in progress still.

```
(use-package! org-download
  :after org
  :bind
  (:map org-mode-map
        (("s-Y" . org-download-screenshot)
         ("s-y" . org-download-yank))))
```

Centaur Tabs

Currently not using, but will return to these.

```
;(after! centaur-tabs
 ; (centaur-tabs-mode -1)
  ;(setq centaur-tabs-height 36
   ;     centaur-tabs-set-icons t
    ;    centaur-tabs-modified-marker "o"
     ;   centaur-tabs-close-button "×"
      ;  centaur-tabs-set-bar 'above)
       ; centaur-tabs-gray-out-icons 'buffer
  ;(centaur-tabs-change-fonts "P22 Underground Book" 160))
;; (setq x-underline-at-descent-line t)
```

Org Fancy Priorities

Makes TODOs look pretty with color coded symbols that can convey information at a glance.

```
 (use-package! org-fancy-priorities
; :ensure t
  :hook
  (org-mode . org-fancy-priorities-mode)
  :config
   (setq org-fancy-priorities-list '("⚡" "⬆" "⬇" "☕")))
```

Org Super Agenda

The agenda gets super cluttered. Org-Super-Agenda is fantastic and allows me to declutter my busy agenda, while not missing anything important.

```
(use-package! org-super-agenda
  :commands (org-super-agenda-mode))
(after! org-agenda
  (org-super-agenda-mode))
```

```
      (setq org-agenda-skip-scheduled-if-done t
            org-agenda-skip-deadline-if-done t
            org-agenda-include-deadlines t
            org-agenda-block-separator nil
            org-agenda-tags-column 100 ;; from testing this seems to be a good value
            org-agenda-compact-blocks t)
      (setq org-agenda-files "~/Dropbox/Org/Daily/")
      (setq org-agenda-custom-commands
            '(("o" "Overview"
               ((agenda "" ((org-agenda-span 'day)
                            (org-super-agenda-groups
                             '((:name "Today"
                                :time-grid t
                                :date today
                                :todo "TODAY"
                                :scheduled today
                                :order 1)))))
                (alltodo "" ((org-agenda-overriding-header "")
                             (org-super-agenda-groups
                              '((:name "Next to do"
                                 :todo "NEXT"
                                 :order 1)
                                (:name "Important"
                                 :tag "Important"
                                 :priority "A"
                                 :order 1)
                                (:name "Due Today"
                                 :deadline today
                                 :order 2)
                                (:name "Due Soon"
                                 :deadline future
                                 :order 8)
                                (:name "Overdue"
                                 :deadline past
                                 :face error
                                 :order 7)
                                (:name "Work"
                                 :tag  "Work"
                                 :order 3)
                                (:name "Dissertation"
                                 :tag "Dissertation"
                                 :order 7)
                                (:name "Emacs"
                                 :tag "Emacs"
                                 :order 13)
                                (:name "Projects"
                                 :tag "Project"
                                 :order 14)
                                (:name "Essay 1"
                                 :tag "Essay1"
                                 :order 2)
                                (:name "Reading List"
                                 :tag "Read"
                                 :order 8)
                                (:name "Work In Progress"
```

```
         :tag "WIP"
         :order 5)
        (:name "Blog"
         :tag "Blog"
         :order 12)
        (:name "Essay 2"
         :tag "Essay2"
         :order 3)
        (:name "Trivial"
         :priority<= "E"
         :tag ("Trivial" "Unimportant")
         :todo ("SOMEDAY" )
         :order 90)
        (:discard (:tag ("Chore" "Routine" "Daily")))))))))))
```

Custom Set Variables

```
(custom-set-variables
 ;; custom-set-variables was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(org-journal-date-format "%A, %d %B %Y" t)
 '(org-journal-date-prefix "#+TITLE: " t)
 '(org-journal-dir "~/Dropbox/Org/Daily/" t)
 '(org-journal-file-format "%Y-%m-%d.org" t)
 '(package-selected-packages (quote (org-fancy-priorities))))
(custom-set-faces
 ;; custom-set-faces was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 )
```

Custom Key Bindings

Some custom key bindings I wrote for the most used functions in my workflow.

```
;; adding custom key-bindings for most used functions
(map! :leader "f a"#'helm-bibtex)  ; "find article" : opens up helm bibtex for
(map! :leader "o n"#'org-noter)    ; "org noter"  : opens up org noter in a he
(map! :leader "r c i"#'org-clock-in); "routine clock in" : clock in to a habit
(map! :leader "c b"#'beacon-blink) ; "cursor blink" : makes the beacon-blink
```

Org Bullets

I like Zaiste's bullets and typeface colors. Let's use those.

```
(use-package! org
  :config
  (setq
  ; org-bullets-bullet-list '("⁖")
   org-todo-keyword-faces
   '(("TODO" :foreground "#7c7c75" :weight normal :underline t)
     ("WAITING" :foreground "#9f7efe" :weight normal :underline t)
     ("INPROGRESS" :foreground "#0098dd" :weight normal :underline t)
     ("DONE" :foreground "#50a14f" :weight normal :underline t)
     ("CANCELLED" :foreground "#ff6480" :weight normal :underline t))
   org-priority-faces '((65 :foreground "#e45649")
                        (66 :foreground "#da8548")
                        (67 :foreground "#0098dd"))
   ))
```

Company

For auto-complete and saving those keystrokes.

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

(setq-default history-length 1000) ; remembering history from precedent
(setq-default prescient-history-length 1000)
```

Info Colors

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

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

(add-hook 'Info-mode-hook #'mixed-pitch-mode)
```

Ox-Hugo

I have recently started using `ox-hugo` to help post on my Hugo based website. The following section sets up an org-capture to enable quick blogging.

```
(defun org-hugo-new-subtree-post-capture-template ()
  "Returns `org-capture' template string for new Hugo post.
See `org-capture-templates' for more information."
  (let* (;; http://www.holgerschurig.de/en/emacs-blog-from-org-to-hugo/
         (date (format-time-string (org-time-stamp-format  :inactive) (org-cur
         (title (read-from-minibuffer "Post Title: ")) ;Prompt to enter the po
         (fname (org-hugo-slug title)))
    (mapconcat #'identity
               `(
                 ,(concat "* TODO " title)
                 ":PROPERTIES:"
                 ,(concat ":EXPORT_FILE_NAME: " fname)
                 ,(concat ":EXPORT_DATE: " date) ;Enter current date and time
                 ,(concat ":EXPORT_HUGO_CUSTOM_FRONT_MATTER: "  ":tags somethi
                 ":END:"
                 "%?\n")          ;Place the cursor here
               "\n")))
(defvar hugo-org-path "/home/cantos/Dropbox/blog/sunny-website/org-content/"
  "define the place where we put our org files for hugo")
;;(defvar org-capture-blog (concat hugo-org-path "blog.org"))

(setq org-capture-templates
      '(
        ("h" "Hugo Post"
         entry
         (file+olp "/home/cantos/Dropbox/blog/sunny-website/org-content/blog.o
         (function  org-hugo-new-subtree-post-capture-template))))
```

Releases

No releases published

Packages

No packages published

Languages

- **Emacs Lisp** 100.0%