ChongChong He's Emacs dotfile

Chong-Chong He

Table of Contents

1. About

This is my Emacs configuration file written in Org mode. It is an attempt to keep my `~/.emacs.d/init.el` file organized and readable, as Emacs configuration could be a life-long practice.

The general structure of this org-file is inspired by this github repository.

1.1. If you want to use this dotfile

- Make a backup of your own `~/.emacs.d/`
- Copy `init.org` and `init.el.tangle` to `~/.emacs.d/` and rename `init.el.tangle` to `init.el`
- Open your Emacs. Your Emacs will automatically export `init.org` into `init.el` which overwrites the originial `init.el`.

1.2. How to update this dotfile

- Edit this file.
- Execute `C-c C-v t` . This will export everything in blocks that start with `#+BEGIN_SRC emacs-lisp` and end with `#+END_SRC` to init.el.

1.3. Editing tips

- Use `C-c '` to edit a code block in an individual buffer in an individual buffer.
- `lisp<tab>` to insert a `#+BEGIN_SRC emacs-lisp #+END_SRC` block.

1.4. Resources

Here is a list of resources where I learned to configure Emacs

1.4.1. Articles

- Elisp Programming
- Emacs on MacOS Catalina 10.15 in ~~2019~~ 2020
- Advanced Techniques for Reducing Emacs Startup Time
- Syncing Custom Set Variables: https://assortedarray.com/posts/my-init-org-setup/

1.4.2. Emacs dotfile examples

- https://github.com/yanghaoxie/emacs-dotfile
- One Dotfile to rule them all!
- dakrone-dotfiles: https://github.com/dakrone/dakrone-dotfiles/blob/master/emacs.org
- Juan José García Ripoll: http://juanjose.garciaripoll.com/blog/my-emacs-windows-configuration/index.html
- 极简 Emacs 开发环境配置
- https://hugocisneros.com/org-config/: very nice

1.5. TODO

- what is ERC?
- Emacs: Next/Previous User Buffer

1.6. Start to use this configuration

When this configuration is loaded for the first time, the init.el is the file that is loaded. It should look like this:

```
;; This file replaces itself with the actual configuration at first run.

;; We can't tangle without org!
(require 'org)
;; Open the configuration
(find-file (concat user-emacs-directory "init.org"))
;; tangle it
(org-babel-tangle)
;; load it
(load-file (concat user-emacs-directory "init.el"))
;; finally byte-compile it
(byte-compile-file (concat user-emacs-directory "init.el"))
```

It tangles the org-file, so that this file is overwritten with the actual configuration. Afterwards, this is not needed, because we can use `C-c C-v t` to run `org-babel-tangle`, which does the same job.

Alternatively, we can add the following configuration to execute `org-babel-tangle` after changes. I prefer to not enable this and do it manually.

```
;; -*- mode: emacs-lisp -*-

;; (defun tangle-init ()
;;       "If the current buffer is 'init.org' the code-blocks are
;; tangled, and the tangled file is compiled."
;;       (when (equal (buffer-file-name)
;;                    ;; (expand-file-name (concat user-emacs-directory
"init.org")))
;;                    "/Users/chongchonghe/dotfiles/emacs/init.org")
;;  ;; Avoid running hooks when tangling.
;;  (let ((prog-mode-hook nil))
;;       (org-babel-tangle)
;;       (byte-compile-file (concat user-emacs-directory "init.el")))))
;; (add-hook 'after-save-hook 'tangle-init)
```

(Not in use) The following config is a common optimization to temporarily disable garbage collection during initialization.

```
;; -*- mode: emacs-lisp -*-

(lexical-let ((old-gc-treshold gc-cons-threshold))
  (setq gc-cons-threshold most-positive-fixnum)
  (add-hook 'after-init-hook
            (lambda () (setq gc-cons-threshold old-gc-treshold))))
```

1.7. Dot emacs debugging

- Remember to delete init.elc as you update init.el
- `Debugger entered--Lisp error: (void-variable personal-keybindings)`
  - `(use-package org)`

2. Basic usage

Bookmarks

Commands: `bookmark-set`, `helm-bookmarks`

org-agenda

Shortcuts: `C-c c`: org-capture, `C-c a t`: org-todo, `C-c a a`: org-agenda

3. Emacs initialization

3.1. Initialization

```
;; Make startup faster by reducing the frequency of garbage
;; collection.  The default is 800 kilobytes.  Measured in bytes.
(setq gc-cons-threshold 100000000)

(package-initialize)
(add-to-list 'load-path "~/.emacs.d/pkgs")

;; (defvar myPackages
;;   '(better-defaults
;;     ;;ein
;;     elpy
;;     flycheck
;;     ;;spolky   ; theme?
;;     ;; py-autopep8

;;     ))
```

3.2. Add package sources

```
(unless (assoc-default "melpa" package-archives)
  (add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t))
(unless (assoc-default "org" package-archives)
  (add-to-list 'package-archives '("org" . "https://orgmode.org/elpa/") t))
```

Use `M-x package-refresh-contents` to reload the list of packages after adding these for the first time.

3.3. Use-package

3.3.1. Usage

To use local packages, use `:ensure nil`, or use `:load-path` keyword. e.g.

```
  (use-package si-general
    :ensure nil
    :load-path "~/.emacs.d/lib")
```

3.3.2. Config

```
  ;; This is only needed once, near the top of the file
  (require 'bind-key)
  (eval-when-compile
```

```
    ;; Following line is not needed if use-package.el is in ~/.emacs.d
    (add-to-list 'load-path "~/.emacs.d/elpa")
    (require 'use-package))

  ;; (add-to-list 'load-path "~/.emacs.d/elpa")
  ;; (require 'use-package)

  ;; (unless (package-installed-p 'use-package)
  ;;   (package-refresh-contents)
  ;;   (package-install 'use-package))

  (setq use-package-always-ensure t)
  (setq use-package-verbose t)

  ;; (use-package ein)
  ;; (use-package ein-notebook)
  ;; (use-package ein-subpackages)
  ;; (use-package better-defaults)
```

4. General configuration

4.1. Fundamentals

```
;; remove backup files (e.g. README.md~)
(setq make-backup-files nil)
;; enable line numbers globally
(global-linum-mode t)
;; Shift-arrow to swith windows
(windmove-default-keybindings)
;; (global-unset-key (kbd "C-x C-c"))
;; (global-unset-key (kbd "M-`")) ; not working
(setq default-fill-column 80)
;; Auto revert mode
(global-auto-revert-mode 1)
;; Keep track of loading time
(defconst emacs-start-time (current-time))
;; start server at startup
(server-start)
;; Disable welcome screen
(setq inhibit-startup-screen t)
;; Search only visible
(setq search-invisible nil)
(setq column-number-mode t)
(setq x-select-enable-clipboard t)
;; (desktop-save-mode 1)
;; end file with new line ("\n")
(setq mode-require-final-newline t)
;; always follow symlinks
(setq find-file-visit-truename t)
```

4.2. Use CMD key for meta

```
;; Use cmd key for meta
;; https://superuser.com/questions/297259/set-emacs-meta-key-to-be-the-mac-key
;; (setq mac-option-key-is-meta nil
;;       mac-command-key-is-meta t
;;       mac-command-modifier 'meta
;;       mac-option-modifier 'super)
```

```
;; (setq mac-option-modifier 'super)
(setq mac-option-modifier 'meta)
(setq mac-command-modifier 'super)
```

4.3. Reload

```
(defun my/reload-emacs-configuration ()
  (interactive)
  (load-file "~/.emacs.d/init.el"))
```

4.4. Column marker at 80

```
  (setq-default
   ;; Column Marker at 80
   whitespace-line-column 80
   whitespace-style       '(face lines-tail))
  (add-hook 'prog-mode-hook #'whitespace-mode)
```

4.5. Set encoding

```
  ;; set encoding
  (set-language-environment "utf-8")
  (set-default-coding-systems 'utf-8)
  ;; (set-buffer-file-coding-system 'utf-8-unix)
  ;; (set-clipboard-coding-system 'utf-8-unix)
  ;; (set-file-name-coding-system 'utf-8-unix)
  ;; (set-keyboard-coding-system 'utf-8-unix)
  ;; (set-next-selection-coding-system 'utf-8-unix)
  ;; (set-selection-coding-system 'utf-8-unix)
  ;; (set-terminal-coding-system 'utf-8-unix)
  ;; (setq locale-coding-system 'utf-8)
  (prefer-coding-system 'utf-8)
```

4.6. Compile

Define `compile-command` as `make`

```
(setq compile-command "make ")
;; (global-set-key (kbd "C-c r") #'recompile)
```

Make the compilation window go away in a few seconds on success (ref: https://emacs.stackexchange.com/a/336)

```
; from enberg on #emacs
(add-hook 'compilation-finish-functions
  (lambda (buf str)
    (if (null (string-match ".*exited abnormally.*" str))
        ;;no errors, make the compilation window go away in a few seconds
        (progn
          (run-at-time
           "2 sec" nil 'delete-windows-on
           (get-buffer-create "*compilation*"))
          (message "No Compilation Errors!")))))
```

Make `*cimpilation*` buffer disappear after done (not working?)

```
(setq special-display-buffer-names
      '("*compilation*"))
```

```
(setq special-display-function
      (lambda (buffer &optional args)
        (split-window)
        (switch-to-buffer buffer)
        (get-buffer-window buffer 0)))
```

4.7. Shared functions

```
  (defun no-auto-fill ()
    "Turn off auto-fill-mode."
    (auto-fill-mode -1)
    (setq word-wrap t)
    )
```

4.8. Clipboard (not working)

simpleclip (https://github.com/rolandwalker/simpleclip)

```
  (use-package simpleclip
      :defer t
      :config
      (simpleclip-mode 1))
```

4.9. default-text-scale

https://github.com/purcell/default-text-scale

```
(use-package default-text-scale
  :defer 2)
```

4.10. Makefile mode indentation

```
(add-hook 'BSDmakefile-mode-hook #'(indent-according-to-mode 1))
```

5. General tools

5.1. Quick open files

```
(global-set-key (kbd "C-c f w") 'isamert/toggle-side-work-org-buffer)
(global-set-key (kbd "C-c f t") 'isamert/toggle-side-work-org-buffer)
(global-set-key (kbd "C-c f j") 'isamert/toggle-side-journal-org-buffer)

(defun isamert/toggle-side-work-org-buffer ()
  "Toggle `bullet.org` in a side buffer for quick note taking.  The buffer is
opened in side window so it can't be accidentaly removed."
  (interactive)
  (isamert/toggle-side-buffer-with-file "~/Dropbox/orgfiles/work.org"))

(defun isamert/toggle-side-journal-org-buffer ()
  "Toggle `bullet.org` in a side buffer for quick note taking.  The buffer is
opened in side window so it can't be accidentaly removed."
  (interactive)
  (isamert/toggle-side-buffer-with-file "~/Dropbox/orgfiles/y-journals.org"))

(defun isamert/buffer-visible-p (buffer)
```

```
 "Check if given BUFFER is visible or not.  BUFFER is a string representing the
buffer name."
  (or (eq buffer (window-buffer (selected-window))) (get-buffer-window
buffer)))

(defun isamert/display-buffer-in-side-window (buffer)
  "Just like `display-buffer-in-side-window' but only takes a BUFFER and rest
of the parameters are for my taste."
  (select-window
   (display-buffer-in-side-window
    buffer
    (list (cons 'side 'right)
          (cons 'slot 0)
          (cons 'window-width 84)
          (cons 'window-parameters (list (cons 'no-delete-other-windows t)
                                         (cons 'no-other-window nil)))))))

(defun isamert/remove-window-with-buffer (the-buffer-name)
  "Remove window containing given THE-BUFFER-NAME."
  (mapc (lambda (window)
          (when (string-equal (buffer-name (window-buffer window)) the-buffer-
name)
            (delete-window window)))
        (window-list (selected-frame))))

(defun isamert/toggle-side-buffer-with-file (file-path)
  "Toggle FILE-PATH in a side buffer. The buffer is opened in side window so it
can't be accidentaly removed."
  (interactive)
  (let ((fname (file-name-nondirectory file-path)))
  (if (isamert/buffer-visible-p fname)
      (isamert/remove-window-with-buffer fname)
    (isamert/display-buffer-in-side-window
     (save-window-excursion
       (find-file file-path)
       (current-buffer))))))
```

6. evil

Ref: http://evgeni.io/posts/quick-start-evil-mode/

```
  ;; (add-to-list 'load-path "~/.emacs.d/evil")
  ;; (require 'evil)
  ;; (evil-mode 1)

  ;; (use-package evil
  ;;   :ensure t
  ;;   :defer .1
  ;;   :init
  ;;   (setq evil-want-integration nil) ;; required by evil-collection
  ;;   (setq evil-want-keybinding nil)
  ;;   (setq evil-search-module 'evil-search)
  ;;   (setq evil-vsplit-window-right t) ;; like vim's 'splitright'
  ;;   (setq evil-split-window-below t) ;; like vim's 'splitbelow'
  ;;   :config
  ;;   (evil-mode 1)
  ;;   (define-key evil-normal-state-map (kbd "<remap> <evil-next-line>")
'evil-next-visual-line)
  ;;   (define-key evil-normal-state-map (kbd "<remap> <evil-previous-line>")
'evil-previous-visual-line)
```

```
  ;;    ;; Make horizontal movement cross lines
  ;;    (setq-default evil-cross-lines t)
  ;;    (setq key-chord-two-keys-delay 0.4)
  ;;    (key-chord-define evil-insert-state-map "jj" 'evil-normal-state)
  ;;    )

  ;; DO NOT PUT EVIL INTO USE-PACKAGE because other part of this dotfile
  ;; relies on it
  (require 'evil)
  ;; (setq evil-want-integration nil) ;; required by evil-collection
  ;; (setq evil-want-keybinding nil)
  (setq evil-search-module 'evil-search)
  (setq evil-vsplit-window-right t) ;; like vim's 'splitright'
  (setq evil-split-window-below t) ;; like vim's 'splitbelow'
  (evil-mode 1)
  (define-key evil-normal-state-map (kbd "<remap> <evil-next-line>") 'evil-
next-visual-line)
  (define-key evil-normal-state-map (kbd "<remap> <evil-previous-line>") 'evil-
previous-visual-line)
  ;; Make horizontal movement cross lines
  (setq-default evil-cross-lines t)
  (setq key-chord-two-keys-delay 0.4)
  ;; (key-chord-define evil-insert-state-map "jj" 'evil-normal-state)
  ;; (define-key evil-insert-state-map (kbd "M-v") 'yank)
  (define-key evil-insert-state-map (kbd "C-e") 'move-end-of-line)
  (define-key evil-insert-state-map (kbd "C-a") 'move-beginning-of-line)

  ;; (use-package evil-collection
  ;;   :after evil
  ;;   :ensure t
  ;;   :config
  ;;   (evil-collection-init))
```

Make movement keys work like they should: instead of go to next logical line, pressing 'j' leads to the next visual line. Ref.

(Disabled) Exit insert mode by pressing j and then j quickly. Reference: https://stackoverflow.com/questions/10569165/how-to-map-jj-to-esc-in-emacs-evil-mode

Treat underscore as part of a word. Ref.

```
;; For python
(add-hook 'python-mode-hook #'(lambda () (modify-syntax-entry ?_ "w")))
;; julia
(add-hook 'julia-mode-hook #'(lambda () (modify-syntax-entry ?_ "w")))
;; For Javascript
(add-hook 'js2-mode-hook #'(lambda () (modify-syntax-entry ?_ "w")))
;; ;; For ruby
;; (add-hook 'ruby-mode-hook #'(lambda () (modify-syntax-entry ?_ "w")))
```

7. org-mode

I use Org Mode to take notes, write my agenda, and do all sorts of stuff. I have the best writing experience with org-mode over all software.

7.1. Usage

7.1.1. Key-bindings

Editing	
`C-c i`	clock-in (and start toggl timer)
`C-c C-p`	org-cliplink
`C-c '`	edit babel code block.

org-agenda	
`C-c t`	org-todo
`C-c c`	org-capture
`C-c a`	org agenda
`f/b/.`	(without evil-org) move forward/backword/today in agenda view
`[[` or `]]`	(with evil-org) agenda-earlier / agenda-later

latex	
C-p	org-latex-preview
C-M-p	org-fragtog-mode

7.1.2. My org files

Here are the Org files I use.

TODO.org	My main TODO file for academics and life
agenda.org	Agenda file
todos.org	Org todo file

7.1.3. Reference

- Org Mode - Organize Your Life In Plain Text!
- org config: https://hugocisneros.com/org-config/
- https://github.com/jzohrab/emacs.d/blob/master/init.el#L31

- My Organizational Workflow

7.2. Basics

```
;; (use-package org
;;   :pin gnu)
(require 'org)
(setq org-image-actual-width (list 500))
(setq org-hide-emphasis-markers t)
;; startup: showeverything
(setq org-startup-folded nil)
(add-hook 'LaTeX-mode-hook 'flyspell-mode)
;; inline image size
;; (setq org-image-actual-width nil)
(setq org-blank-before-new-entry
      '((heading . t) (plain-list-item . nil)))

(use-package htmlize
  :defer 10
  :load-path "/Users/chongchonghe/dotfiles/emacs/packages/emacs-htmlize")

  (defun org-toggle-hide-emphasis-markers ()
    "Toggle org-hide-emphasis-markers"
    (interactive)
    (if org-hide-emphasis-markers
        (setq org-hide-emphasis-markers nil)
      (setq org-hide-emphasis-markers t)))
```

Disable descriptive (hidden) links (Reference). Disabled. Use `C-c C-l` to edit a link.

```
  ;; (setq org-descriptive-links nil)
```

Auto indent mode as default

```
  (setq org-startup-indented t)
```

Auto fill? No!

```
  ;; (add-hook 'org-mode-hook 'turn-on-auto-fill)
  (auto-fill-mode -1)
  (remove-hook 'text-mode-hook #'turn-on-auto-fill)

  (setq outline-blank-line 2)
```

org-show-two-levels

```
(defun org-show-two-levels ()
  (interactive)
  (org-content 2))
(with-eval-after-load 'org
  (define-key org-mode-map (kbd "C-c 2") 'org-show-two-levels))
(defun org-show-three-levels ()
  (interactive)
  (org-content 3))
(with-eval-after-load 'org
  (define-key org-mode-map (kbd "C-c 3") 'org-show-three-levels))
;; Evaluate it after startup
;; (add-hook 'org-mode-hook #'org-show-two-levels)
(add-hook 'org-view-mode-hook '(text-scale-adjust))
```

To execute a function at startup, put the following script at the end of a file

```
  ;; Local Variables:
```

```
  ;; eval: (progn (org-show-two-levels) (text-scale-adjust 1))
  ;; End:
```

org-cliplink

```
(use-package org-cliplink
  :defer 6
  :bind ("C-c C-p" . 'org-cliplink))
;; (require 'org-cliplink)
;; (define-key org-mode-map (kbd "C-c C-p") 'org-cliplink)
```

Unknown

```
  ;; (setq org-modules '(org-tempo))
```

org-autolist

```
;; (use-package org-autolist
;;   :config
;;   (add-hook 'org-mode-hook (lambda () (org-autolist-mode)))
;;   )
```

7.3. Key-bindings

7.3.1. Local keys

```
  (defun my-org-mode-config ()
            (local-set-key "\M-n" 'outline-next-visible-heading)
            (local-set-key "\M-p" 'outline-previous-visible-heading)
            ;; table
            (local-set-key "\C-\M-w" 'org-table-copy-region)
            (local-set-key "\C-\M-y" 'org-table-paste-rectangle)
            (local-set-key "\C-\M-l" 'org-table-sort-lines)
            ;; display images
            (local-set-key "\M-I" 'org-toggle-iimage-in-org)
            ;; TODOlist
            ;; fix tab
            ;; (local-set-key "\C-y" 'yank)
            (local-set-key "\M-h" 'windmove-left)
            (local-set-key "\C-cl" 'grg-store-link)
            (local-set-key "\C-cb" 'org-switchb)
            (local-set-key "\C-cp" 'org-display-inline-images)
            (local-set-key "\M-h" 'org-metaleft)
            (local-set-key "\M-l" 'org-metaright)
            (local-set-key "\C-ce" 'org-html-export-to-html)
            ;; (local-set-key (kbd "s-p") (kbd "C-c C-e h h"))
            (setq-local truncate-lines 'nil)
            ;; (org-indent-mode)  ;; not working?
            )
  (add-hook 'org-mode-hook 'my-org-mode-config)

  (with-eval-after-load 'evil-maps
            (define-key evil-normal-state-map (kbd "C-p") 'org-latex-preview)
            (define-key evil-normal-state-map (kbd "C-M-p") 'org-fragtog-mode))
  ;; (define-key org-mode-map (kbd "C-p") 'org-latex-preview)
```

Not used:

```
  ;; https://orgmode.org/manual/Conflicts.html
  ;; Make windmove work in Org mode:
  (add-hook 'org-shiftup-final-hook 'windmove-up)
```

```
  (add-hook 'org-shiftleft-final-hook 'windmove-left)
  (add-hook 'org-shiftdown-final-hook 'windmove-down)
  (add-hook 'org-shiftright-final-hook 'windmove-right)
  (define-key org-mode-map (kbd "M-h") 'windmove-left) ;; org conflicts

  ;; (evil-define-key 'normal org-mode-map (kbd ", ,") 'org-insert-structure-
template)
```

7.3.2. rename-file-and-buffer

```
  ;; source: http://steve.yegge.googlepages.com/my-dot-emacs-file
  (defun rename-file-and-buffer (new-name)
    "Renames both current buffer and file it's visiting to NEW-NAME."
    (interactive "sNew name: ")
    (let ((name (buffer-name))
          (filename (buffer-file-name)))
      (if (not filename)
          (message "Buffer '%s' is not visiting a file!" name)
        (if (get-buffer new-name)
            (message "A buffer named '%s' already exists!" new-name)
          (progn
            (rename-file filename new-name 1)
            (rename-buffer new-name)
            (set-visited-file-name new-name)
            (set-buffer-modified-p nil))))))
```

7.3.3. evil-org-mode

https://github.com/Somelauw/evil-org-mode

```
(use-package evil-org
  :ensure t
  :after org
  :hook (org-mode . (lambda () evil-org-mode))
  :config
  (evil-org-set-key-theme '(textobjects insert navigation additional shift todo
heading))
  (require 'evil-org-agenda)
  (evil-org-agenda-set-keys)
  (setq org-super-agenda-header-map (make-sparse-keymap))
  )

;; 'evil-org-agenda is replaced by the following
;; (define-key org-agenda-mode-map "j" 'evil-next-line)
;; (define-key org-agenda-mode-map "k" 'evil-previous-line)

;; (use-package org-agenda
;;   :bind (:map org-agenda-mode-map
;;          ("j" . org-agenda-next-item)
;;          ("k" . org-agenda-previous-time)))

;; (use-package org-agenda
;;   :config
;;   (define-key org-agenda-mode-map (kbd "j") #'org-agenda-next-item)
;;   (define-key org-agenda-mode-map (kbd "k") #'org-agenda-previous-item))
```

7.3.4. org-evil

```
;; (use-package org-evil
```

```
;;    :ensure t
;;    :after org
;;    )
```

7.4. Appearance

Set up a font-lock substitution for list markers (I always use “-” for lists, but you can change this if you want) by replacing them with a centered-dot character: (Ref)

```
  ;; (font-lock-add-keywords
  ;;  'org-mode
  ;;  '(("^ *\\([-]\\) "
  ;;     (0 (prog1 () (compose-region (match-beginning 1) (match-end 1)
"•"))))))
```

Proportional font, in different sizes, for the headlines. The fonts listed will be tried in sequence, and the first one found will be used.

Beautify org mode

Org-superstar (reference)

```
  ;; (use-package org-bullets
  ;;       :init
  ;;       (add-hook 'org-mode-hook #'org-bullets-mode)
  ;;       (setq org-bullets-bullet-list '("  " "□" "□" "  " "  " "□"))
  ;;       )
  (setq-default org-list-indent-offset 4)
  (use-package org-superstar              ; supersedes `org-bullets'
                        :ensure
                        :after org
                        :config
                        ;; Every non-TODO headline now have no bullet
                        (setq org-superstar-headline-bullets-list '("\u200b"))
                        (setq org-superstar-leading-bullet "\u200b")
                        (setq org-superstar-item-bullet-alist
        '((?+ . ?+)
                              (?* . ?➤)
                              (?- . ?•)))
                        ;; Enable custom bullets for TODO items
                        (setq org-superstar-special-todo-items t)
                        (setq org-superstar-todo-bullet-alist
        '(("TODO" "□")
                              ("NEXT" "✒")
                              ("HOLD" "☆")
                              ("WAIT" "  ")
                              ("CXLD" "✘")
                              ("DONE" "✔")))
                        (org-superstar-restart))
  ;; (setq org-ellipsis "⤵")
  ;; (setq org-ellipsis "▼")

  ;; simple version, only change font size
  ;; (custom-set-faces
  ;;   '(org-level-1 ((t (:inherit outline-1 :height 1.75))))
  ;;   '(org-level-2 ((t (:inherit outline-2 :height 1.5))))
  ;;   '(org-level-3 ((t (:inherit outline-3 :height 1.25))))
  ;;   '(org-level-4 ((t (:inherit outline-4 :height 1.1))))
  ;;   '(org-level-5 ((t (:inherit outline-5))))
  ;;   '(org-level-6 ((t (:inherit outline-6))))
  ;;   '(org-level-7 ((t (:inherit outline-7))))
```

```
  ;;   '(org-document-title ((t (:height 2.0 :underline nil))))
  ;; )

  ;; complex version: change font as well
  (add-hook 'org-mode-hook 'variable-pitch-mode)
  (add-hook 'org-mode-hook 'visual-line-mode)
  (with-eval-after-load 'org
    (setq word-wrap t)
    )

  ;; check (custom-theme-set-faces) in the appearance section

  ;; (custom-set-faces
  ;;   ;; '(org-level-1 ((t (:inherit outline-1 :height 1.75))))
  ;;   ;; '(org-document-title ((t (:height 2.0 :underline nil))))
  ;;   '(mu4e-view-face ((t (:inherit default :height 1.2))))
  ;; )

  ;; (setq org-ellipsis "⤵")
```

7.5. Embed local video

Adapted from this method: http://endlessparentheses.com/embedding-youtube-videos-with-org-mode-links.html. `[[mv:movie.mp4]]` will export a html5 video.

```
(defvar mv-iframe-format
  ;; You may want to change your width and height.
  (concat "<video"
        " height=\"500\""
        " style=\"display:block; margin: 0 auto;\" controls>"
        " <source"
        " src=\"%s\""
        " type=\"video/mp4\">"
        " <figcaption> \"%s\" </figcaption> "
        "</video>"))

(org-add-link-type
 "mv"
 (lambda (handle)
   (browse-url
    (concat "https://www.youtube.com/embed/"
          handle)))
 (lambda (path desc backend)
   (cl-case backend
     (html (format mv-iframe-format
                path (or desc "")))
     (latex (format "\href{%s}{%s}"
                 path (or desc "video"))))))
```

Embed audio

```
(defvar audio-iframe-format
  ;; You may want to change your width and height.
  (concat "<iframe"
        " width=\"600\""
        " height=\"60\""
        " style=\"display:block; margin: 0\""
        " src=\"%s\">"
        "</iframe>"))

(org-add-link-type
```

```
 "audio"
 (lambda (handle)
   (browse-url
    (concat "https://www.youtube.com/embed/"
          handle)))
 (lambda (path desc backend)
   (cl-case backend
     (html (format audio-iframe-format
                path (or desc "")))
     (latex (format "\href{%s}{%s}"
                 path (or desc "audio"))))))
```

Other unknown config

```
  ;; <tab> for 'indent-for-tab-command'
  ;; (evil-define-key 'insert org-mode-map (kbd "C-t") #'indent-for-tab-
command)

  ;; load shared .el followed by Emacs specific config
  ;;(load-file "~/.my-elips/org.el")

  ;; (require 'org-mu4e)
```

7.6. org babel

To supress "risky local variable…" https://emacs.stackexchange.com/questions/21575/mark-a-local-variable-safe-for-any-value

```
;; allow remembering risky variables
(put 'org-babel-python-command 'safe-local-variable
     (lambda (x) t))
```

Ref: https://www.juliabloggers.com/julia-with-emacs-org-mode/

```
;; (require 'ess-site)
(setq  inferior-julia-program-name
"/Applications/Julia-1.5.app/Contents/Resources/julia/bin/julia")
(org-babel-do-load-languages
 'org-babel-load-languages
 '((C . t)
   (julia . t)
   (shell . t)
   (python . t)
   (ipython . t)
   ))

(setq
 org-export-babel-evaluate nil
 org-confirm-python-evaluate nil
 org-confirm-babel-evaluate nil
 org-confirm-C++-evaluate nil
 )
```

Display errors and warnings in an org-mode code block. Ref.

```
  (defvar org-babel-eval-verbose t
    "A non-nil value makes `org-babel-eval' display")

  (defun org-babel-eval (cmd body)
    "Run CMD on BODY.
  If CMD succeeds then return its results, otherwise display
```

```
  STDERR with `org-babel-eval-error-notify'."
    (let ((err-buff (get-buffer-create " *Org-Babel Error*")) exit-code)
      (with-current-buffer err-buff (erase-buffer))
      (with-temp-buffer
        (insert body)
        (setq exit-code
              (org-babel--shell-command-on-region
               (point-min) (point-max) cmd err-buff))
        (if (or (not (numberp exit-code)) (> exit-code 0)
                (and org-babel-eval-verbose (> (buffer-size err-buff) 0))) ;
new condition
            (progn
              (with-current-buffer err-buff
                (org-babel-eval-error-notify exit-code (buffer-string)))
              nil)
          (buffer-string)))))

  (setq org-babel-eval-verbose t)
```

Indentation

```
(setq org-src-preserve-indentation t)
```

ob-async: https://github.com/astahlman/ob-async

```
(use-package ob-async)
```

ob-ipython: https://github.com/gregsexton/ob-ipython

```
(use-package ob-ipython)
```

7.7. org-todo and org-agenda

7.7.1. Config

```
(evil-define-key 'normal org-mode-map (kbd "t") 'org-todo)
(evil-define-key 'normal org-mode-map (kbd "C-t") 'org-todo-list)
(evil-define-key 'normal org-mode-map (kbd "C-t") 'org-todo-list)
(define-key evil-normal-state-map (kbd "C-a") 'org-agenda)
(setq org-directory "~/Dropbox/orgfiles")

(with-eval-after-load 'org
  ;; (setq org-directory "/Users/chongchonghe/Dropbox/orgfiles")
  (setq org-agenda-files '("~/Dropbox/orgfiles/work.org"
                     "~/Dropbox/orgfiles/z-life.org"
                     "~/Dropbox/orgfiles/y-journals.org"
                     "~/Dropbox/orgfiles/done.org"
                     "~/Dropbox/orgfiles/journal/"
                           ))
  ;; (setq org-agenda-files '("~/Dropbox/orgfiles"))
  ;; (setq org-agenda-files
  ;;     "~/Dropbox/orgfiles/agenda.org")
  (setq org-default-notes-file "~/Dropbox/orgfiles/work.org")
  (setq org-agenda-confirm-kill t)
  ;;open agenda in current window
  (setq org-agenda-window-setup (quote current-window))
  )
(defun my/buffer-face-mode-variable ()
  "Set font to a variable width (proportional) fonts in current buffer"
  (interactive)
```

```
  (setq buffer-face-mode-face '(:family "Roboto Slab"
                                :height 180
                                :width normal))
  (buffer-face-mode))

(defun my/style-org-agenda()
  ;; (my/buffer-face-mode-variable)
  (set-face-attribute 'org-agenda-date nil :height 1.1)
  (set-face-attribute 'org-agenda-date-today nil :height 1.1 :slant 'italic)
  (set-face-attribute 'org-agenda-date-weekend nil :height 1.1))

(add-hook 'org-agenda-mode-hook 'my/style-org-agenda)

(setq org-agenda-breadcrumbs-separator " ❱ "
      ;; org-agenda-current-time-string "  □□□□□□□□□□□□ now"
      org-agenda-time-grid '((weekly today require-timed)
                             (800 1000 1200 1400 1600 1800 2000)
                             "......" "┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈")
      org-agenda-prefix-format '((agenda . " %i %-10:c%?-16t%-4e% s")
                        (todo . " %i %-10:c")
                        (tags . " %i %-10:c")
                        (search . " %i %-10:c"))
)

(setq org-agenda-format-date (lambda (date) (concat "\n" (make-string (window-
width) 9472)
                                                    "\n"
                                                    (org-agenda-format-date-
aligned date))))
(setq org-cycle-separator-lines 2)
```

A week view spanning the current day

```
(setq org-agenda-span 7
    org-agenda-start-on-weekday nil
    ;; org-agenda-start-day "-3d"
    )
```

Not used

```
  ;; ; ;; org-agenda
  ;; (use-package org-projectile
  ;;   :bind (("C-c n p" . org-projectile-project-todo-completing-read)
  ;;     ;; ("C-c c" . org-capture)
  ;;     ;; ("C-c a" . org-agenda)
  ;;     )
  ;;   :config
  ;;   (progn
  ;;     (setq org-projectile-projects-file "~/Dropbox/orgfiles/tasks.org")
  ;;     (setq org-agenda-files (append org-agenda-files (org-projectile-todo-
files)))
  ;;     (push (org-projectile-project-todo-entry) org-capture-templates))
  ;;   :ensure t)
```

7.7.2. org-super-agenda

https://github.com/alphapapa/org-super-agenda

Check examples of configuration here.

```
(use-package org-super-agenda
```

```
  :defer 2
  :config
  (org-super-agenda-mode)
  )

(setq org-super-agenda-groups
      '(
      ;; (:name "   Calendar" :time-grid t)
      (:name "Today" ; Optionally specify section name
             :time-grid t  ; Items that appear on the time grid
             :todo "TODAY") ; Items that have this TODO keyword
      (:name "⚠ Overdue!" :deadline past)
      (:name "⚠ Overdue!" :scheduled past)
      (:name "   Important" :priority "A")
      (:name "Optional" :priority "D" :order 90)
      (:auto-category t)))

;; (let ((org-super-agenda-groups
;;        '((:auto-category t))))
;;   (org-agenda-list))
```

7.7.3. org-journal

```
(use-package org-journal
  :ensure t
  :defer t
  :bind
  ("s-j" . org-journal-new-entry)
  ("s-s" . org-journal-new-scheduled-entry)
  ("s-n" . org-journal-open-next-entry)
  ("s-p" . org-journal-open-previous-entry)
  :init
  ;; Change default prefix key; needs to be set before loading org-journal
  (setq
   org-journal-prefix-key "M-g "
   org-journal-enable-agenda-integration t)
  :config
  (setq org-journal-dir (concat org-directory "/journal")
        org-journal-file-type 'weekly
        org-journal-file-format "%Y%m%d.org"
      org-journal-date-format "%A, %Y-%m-%d"
        ))
```

7.7.4. Todo and Priority

To set the priority of the current headline, do `C-c ,`, or `S-UP` / `S_DOWN`.

```
(setq org-default-priority ?A)
(setq org-highest-priority ?A)
(setq org-lowest-priority ?D)
;;set colours for priorities
(setq org-priority-faces '((?A . (:foreground "#FF0000" :weight bold))
                     (?B . (:foreground "#FF9815" :weight bold))
                     (?C . (:foreground "#68DF40"))
                     (?D . (:foreground "#11D3FF"))))
;;Different bullets
(setq org-todo-keywords
      '((sequence "TODO(t!)" "NEXT(n!)" "DOIN(o!)" "WAIT(w!)" "FLUP(f!)"
"REFI(r!)" "|" "(e!)" "SCHE(s!)" "CXLD(c!)" "DONE(d!)"))
```

```
      org-todo-keyword-faces
      '(("TODO" . (:foreground "magenta" :weight bold))
      ("DOIN" . (:foreground "blue"))
      ("FLUP" . (:foreground "orange"))
      ("REFI" . (:foreground "#A52A2A"))
      ;; ("CANCELLED" . (:foreground "white" :background "#4d4d4d" :weight
bold))
      ("CXLD" . (:foreground "gray"))
      ("NEXT" . "#008080")
      ("DONE" . "#333"))
      org-agenda-skip-scheduled-if-done t
      org-agenda-skip-deadline-if-done t
      )

(defun my-org-set-dark-todo-faces ()
  (setq org-todo-keyword-faces
      '(("TODO" . org-warning)
        ("DOIN" . (:foreground "yellow"))
        ("FLUP" . (:foreground "magenta"))
        ("REFI" . (:foreground "#A52A2A"))
        ;; ("CANCELLED" . (:foreground "white" :background "#4d4d4d" :weight
bold))
        ("CXLD" . (:foreground "gray"))
        ("NEXT" . "#008080")
        ("DONE" . "#333"))))

(defun my-org-set-light-todo-faces ()
  (setq org-todo-keyword-faces
      ;; '(("TODO" . org-warning)
      ;; '(("TODO" . (:foreground "purple"))
      '(("DOIN" . (:foreground "orange"))
      ("FLUP" . (:foreground "magenta"))
      ("REFI" . (:foreground "#A52A2A"))
      ;; ("CANCELLED" . (:foreground "white" :background "#4d4d4d" :weight
bold))
      ("CXLD" . (:foreground "gray"))
      ("NEXT" . "#008080")
      ("DONE" . "#333"))))
```

7.7.5. Org capture

My sketch notes: Notability - 2021 Sketchbook pg 2.

References: https://www.youtube.com/watch?v=KdcXu_RdKI0

Templates:

```
(setq org-capture-templates
      '(("t"              ; hotkey
         "Todo list item" ; name
         entry            ; type
         ;; heading type and title
         (file+heading org-default-notes-file "Tasks")
         "* TODO %?\n  %i\n  %a")   ; template
        ))
```

Generally, use `%u/U` (inactive timestamp, [time]) instead of `%t/T` (active timestamp, <time>), because I don't want the creation of a todo item in the agenda. Use `%a` for annotation.

My configuration:

```
(with-eval-after-load 'org
  (setq org-capture-templates
        '(("t" "Todo" entry (file+olp "" "Todos")
           "* TODO [#%^{Priority?||A|B|C|D}] %?\nSCHEDULED: %^t %u\n\n\n\n"
:empty-lines-after 1 :empty-lines-before 1)
          ("s" "Scheduled" entry (file+olp "" "Todos")
           "* [#A] %^{Title}\nSCHEDULED: %^t\n%u\n%?\n\n\n\n" :empty-lines-before
1 :empty-lines-after 1)
          ("l" "Life" entry (file+olp "~/Dropbox/orgfiles/z-life.org" "Todos")
           "* TODO [#%^{Priority?||A|B|C|D}] %?\nSCHEDULED: %^t %u\n\n\n\n"
:empty-lines-before 1 :empty-lines-after 1)
          ("j" "Journal" entry (file+olp "~/Dropbox/orgfiles/y-journals.org"
"2021")
           "** %?\n\n" :empty-lines-before 1 :empty-lines-after 1)
          ("o" "Doing" entry (file+olp "" "Todos")
           "* DOIN [#%^{Priority?||A|B|C|D}] %?\n %u\n\n\n" :empty-lines-before 1
:empty-lines-after 1)
          ("a" "Todo with link" entry (file+olp "" "Todos")
           "* TODO [#%^{Priority?||A|B|C|D}] %?\n %U\n %a\n\n\n" :empty-lines-
before 1 :empty-lines-after 1)
          ;; ("w" "Work task" entry (file+headline "" "Todos")
          ;;  "* TODO [#A] %?\n  %U\n  %a\n\n\n" :empty-lines-before 1)
          ("n" "Notes" entry (file+olp "" "General Notes")
           "* REFI %? :NOTE:\n%U\n\n\n" :empty-lines-before 1 :empty-lines-after
1)
          ("m" "Meeting" entry (file+olp "" "Meetings")
           "* REFI Meeting with %? :MEETING:\n%U\n\n\n" :empty-lines-before 1
:empty-lines-after 1)
          ("i" "Ideas" entry (file+olp "" "Ideas")
           "* %?\n  %u\n\n\n" :empty-lines-before 1 :empty-lines-after 1)
          ("s" "Scheduled" entry (file+olp "" "Todos")
           "* [#A] %^{Title}\nSCHEDULED: %^t\n%u\n%?\n\n\n\n" :empty-lines-before 1
:empty-lines-after 1)
          ("e" "Event" entry (file+olp "" "Todos")
           "* %^{This is a?||TODO |NEXT |FLUP |DOIN |SCHE |REFI}%^{Title}\
nSCHEDULED: %^t\n%t\n%?")
          ("f" "Followup" entry (file+olp "" "Followups")
           "* FLUP [#B] %?\n  %U\n  %a\n\n\n" :empty-lines-before 1 :empty-lines-
after 1)
          ;; ("w" "Work" entry (file+headline "" "Work")
          ;;  "* DOIN [#A] %? :WORK:\n%U\n\n\n" :empty-lines-before 1)
          ;; ("g" "General todo" entry (file+headline
"/Users/chongchonghe/tasks.org" "Todos")
          ;;  "* TODO [#B] %?\n %a" :empty-lines 1)
          )
        ))
```

7.7.6. Refile

Ref: https://blog.aaronbieber.com/2017/03/19/organizing-notes-with-refile.html

```
(setq org-refile-targets '((("~/Dropbox/orgfiles/work.org"
                     "~/Dropbox/orgfiles/z-life.org"
                     "~/Dropbox/orgfiles/y-journals.org")
                      :maxlevel . 2)))
```

7.7.7. Exporting to LaTeX

No used. TODO

```
;; Exporting to LaTeX and PDF, formatting
;; http://pragmaticemacs.com/emacs/org-mode-basics-v-exporting-your-notes/
(with-eval-after-load 'ox-latex
  (add-to-list 'org-latex-classes
           '("bjmarticle"
             "\\documentclass{article}
\\usepackage[utf8]{inputenc}
\\usepackage[T1]{fontenc}
\\usepackage{graphicx}
\\usepackage{longtable}
\\usepackage{hyperref}
\\usepackage{natbib}
\\usepackage{amssymb}
\\usepackage{amsmath}
\\usepackage{geometry}
\\geometry{a4paper,margin=0.5in,marginparsep=7pt, marginparwidth=.6in}"
             ("\\section{%s}" . "\\section*{%s}")
             ("\\subsection{%s}" . "\\subsection*{%s}")
             ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
             ("\\paragraph{%s}" . "\\paragraph*{%s}")
             ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
```

7.8. org-present

```
(setq org-present-text-scale 2)

(eval-after-load "org-present"
  '(progn
     (add-hook 'org-present-mode-hook
               (lambda ()
                 (org-present-big)
                 (org-display-inline-images)
                 (org-present-hide-cursor)
                 (org-present-read-only)
                 ;;(toggle-frame-fullscreen)
                 ))
     (add-hook 'org-present-mode-quit-hook
               (lambda ()
                 (org-present-small)
                 (org-remove-inline-images)
                 (org-present-show-cursor)
                 (org-present-read-write)
                 ;;(toggle-frame-fullscreen)
                 ))))
```

7.9. org-toggl

How to setup? Check the org-toggl github repository for instructions. Basically, add the following code to your init.el, where the toggl-auth-token is copied from your toggl account.

Usage:

'C-c i' then 'C-c o' (re-mapped) to clock in then clock out, which starts a toggl timer. This is enabled by org-toggl. Note that 'C-c o' wont' work. It won't stop the timer.

```
;; (with-eval-after-load 'org
;;   (use-package org-toggl
;;     :init
;;     (setq toggl-auth-token "ce3e8fc3922edda6986a6e729509338f")
```

```
;;      (setq org-toggl-inherit-toggl-properties t)
;;      :load-path "/Users/chongchonghe/dotfiles/emacs/packages"
;;      :config
;;      (toggl-get-projects)
;;      (org-toggl-integration-mode)
;;      ;; remove clock-out since it failed at stopping toggl timer
;;      (remove-hook 'org-clock-out-hook #'org-toggl-clock-out)
;;      ;; bind C-c i to clock-in then clock-out
;;      (define-key org-mode-map (kbd "C-c i")
;;        (lambda () (interactive) (org-clock-in) (sit-for 2) (org-clock-out)))
;;      )
;;    )

(use-package org-toggl
  :after org
  :defer 15
  :init
  (setq toggl-auth-token "ce3e8fc3922edda6986a6e729509338f")
  (setq org-toggl-inherit-toggl-properties t)
  (setq toggl-default-project "Research")
  :load-path "/Users/chongchonghe/dotfiles/emacs/packages"
  :config
  (toggl-get-projects)
  (org-toggl-integration-mode)
  (define-key org-mode-map (kbd "C-c I") 'toggl-select-default-project)

  ;; remove clock-out since it failed at stopping toggl timer
  (remove-hook 'org-clock-out-hook #'org-toggl-clock-out)
  ;; bind C-c i to clock-in then clock-out
  (define-key org-mode-map (kbd "C-c i")
    (lambda () (interactive) (org-clock-in) (sit-for 5) (org-clock-out)))

  ;; This is the originial setup. Need to invoke clock-in and clock-out to
  ;; start and stop a projection
  ;; (add-hook 'org-clock-out-hook #'org-toggl-clock-out)
  ;; (define-key org-mode-map (kbd "C-c i") 'org-clock-in)
  ;; (define-key org-mode-map (kbd "C-c o") 'org-clock-out)

  )
```

7.10. org file apps

```
  (setq org-file-apps
          '(("\\.docx\\'" . default)
        ("\\.mm\\'" . default)
        ("\\.x?html?\\'" . default)
        ("\\.pdf\\'" . default)
        ("\\.md\\'" . default)
        ("\\.png\\'" . default)
        (auto-mode . emacs)))
```

7.11. exec-path-from-shell

```
  (use-package exec-path-from-shell
      :ensure t
      :config
      (when (memq window-system '(mac ns x))
      (exec-path-from-shell-initialize)))
```

7.12. org latex preview

```
  (setq org-format-latex-options (plist-put org-format-latex-options :scale
1.5))
```

7.13. org export

```
  ;; (use-package ox-md)
  (setq org-export-backends '(ascii html md icalendar latex odt))

(defun my-org-html-postamble (plist)
  (concat "<p>Author: Chong-Chong He</p>"
        (format "<p>Last updated: %s</p>" (format-time-string "%Y-%b-%d"))
        "<a
href='https://www.astro.umd.edu/~chongchong/'>www.astro.umd.edu/~chongchong/</
a>"
        ))
(setq org-html-postamble 'my-org-html-postamble)

;; (setq org-export-html-postamble-format
;;          '(("en" "<p class=\"author\">Author: %a (%e)</p><p
class=\"date\">Last Updated %d.</p>")))

;; (setq org-html-postamble-format
;; ;; (setq org-export-html-postamble-format
;;       '(("en" "<p class=\"author\">Author: %a (%e)</p>
;; Last updated: <p class=\"date\">Date: %d</p>
;; <p class=\"creator\">Generated by %c</p>
;; <p class=\"xhtml-validation\">%v</p>
;; ")))
```

Remove validate

```
(setq org-html-validation-link nil)
```

Set postamble

```
  (defun my-website-html-postamble (options)
      (concat "<hr>"
                  (if (and (plist-get options ':keywords) (not (string= (plist-
get options ':keywords) "")))
                        (format "<p>Keywords: %s</p>" (plist-get options
':keywords))
                        "")
                  (format "<p class=\"date\">Modified: %s</p>" (format-time-
string "%Y-%m-%d %H:%M:%S"))
                  (format "<p>Copyright (c) %s %s</p>"
                              (car (split-string (car (plist-get options
':date)) "-")) ;; TODO: get from custom document option
                              (car (plist-get options ':author)))
                  (format "<p>%s</p>" (plist-get options ':creator))))
```

7.14. Publishing

Ref:

- Basic: Publishing Org-mode files to HTML
- 极简 Emacs 开发环境配置: org-publish

7.14.1. From publish.el

Config

```
(require 'ox-publish)

(global-hl-line-mode 1)
(setq org-src-fontify-natively t)
(add-to-list 'load-path "/Users/chongchonghe/dotfiles/emacs/packages/emacs-
htmlize")
(require 'htmlize)

;; (setq org-html-postamble nil)
(setq org-html-postamble
      (concat "<p>Author: %a</p>"
              "<p>%d</p>"
              "<p><a href='https://www.astro.umd.edu/~chongchong/'>"
              "www.astro.umd.edu/~chongchong/</a></p>"))

;; sitemap function
(defun @-org-publish-org-sitemap (title list)
  "Sitemap generation function."
  (concat (format "#+TITLE: %s\n" title)
          ;; "\n#+SETUPFILE: ../style/default.setup\n"
              "\n#+SETUPFILE: ~/dotfiles/org-styles/simple_inline.theme\n"
          "#+OPTIONS: toc:nil\n"
          (org-list-to-subtree list)
          "\n"
          ))

(setq org-publish-project-alist
      '(("body"
         ;; generic
         :base-directory "."
         :base-extension "org"
         :publishing-directory "../public"
         :recursive t
         :language en
         ;; html
         :publishing-function org-html-publish-to-html
         ;; sitemap
         :auto-sitemap t
         :sitemap-filename "index.org"
         :sitemap-title "Index"
         ;; :sitemap-sort-files anti-chronologically
         ;; :sitemap-file-entry-format "%d - %t"
         ;; :sitemap-function my-website-sitemap-function
         ;; :sitemap-function org-publish-org-sitemap
         :sitemap-function @-org-publish-org-sitemap
         ;; :html-home/up-format "<div> <a accesskey='h' href='index.html'> UP
</a> | <a accesskey='H' href='index.html'> HOME </a> </div>"
         )
        ("css"
         :base-directory "../style/"
         :base-extension "css\\|js"
         :publishing-directory "../public/css"
         :publishing-function org-publish-attachment
         :recursive t)
        ("attach"
         :base-directory "../attach/"
         ;; :base-extension "png\\|jpg\\|ico"
```

```
         :base-extension "png\\|jpg\\|ico\\|svg"
         :publishing-directory "../public/attach"
         :publishing-function org-publish-attachment
         :recursive t)
        ("all" :components ("body" "css" "attach"))))

  (setq org-publish-project-alist
        '(("org"
           :base-directory "~/Documents/org/"
           :publishing-directory "~/Documents/publish/"
           :section-numbers nil
           :table-of-contents nil
           :publishing-function org-html-publish-to-html
           ;; :publishing-function 'org-publish-org-to-html
           :style "<link rel=\"stylesheet\"
                  href=\"../style/style.css\"
                  type=\"text/css\"/>")))
```

sitemap:

```
(defun my-website-sitemap-function (project &optional sitemap-filename)
  "Custom sitemap generator that inserts additional options."
  (let ((buffer (org-publish-org-sitemap project sitemap-filename)))
    (with-current-buffer buffer
      (insert "\n#+SETUPFILE: ../style/default.setup")
      (save-buffer))))
```

7.14.2. Other

org-plus-contrib:

```
(require 'ox-extra)
(ox-extras-activate '(ignore-headlines))
;; (use-package ox-extra
;;   :config
;;   (ox-extras-activate '(ignore-headlines))
;;   )
```

7.15. org-ref

I haven't make it work yet. For now, use cite:citekey to cite a paper, altough it won't reder properly in HTML output.

Ref:

- https://github.com/jkitchin/org-ref
- Manual: https://github.com/jkitchin/org-ref/blob/master/org-ref.org
- Quick guide: https://www.reddit.com/r/orgmode/comments/5shz5o/orgref_introduction/
- Author guide: https://kitchingroup.cheme.cmu.edu/blog/2014/05/13/Using-org-ref-for-citations-and-references/

```
(setq
 org-ref-bibliography-notes "~/Academics/org-ref-notes.org"
 org-ref-default-bibliography '("~/folders-and-files/BIB_HE.bib")
 org-ref-pdf-directory "~/Academics/Papers/"
 )
(setq bibtex-completion-bibliography '("~/folders-and-files/BIB_HE.bib"
                                       "~/folders-and-files/Books.bib")
```

```
      bibtex-completion-library-path "~/Academics/Papers/"
      bibtex-completion-notes-path "~/Dropbox/orgfiles/org-ref-notes.org"
      )

;; (use-package org-ref
;;   :init
;;   (setq bibtex-completion-bibliography '("~/folders-and-files/BIB_HE.bib"
;;                                          "~/folders-and-files/Books.bib")
;;   bibtex-completion-library-path "~/Academics/Papers/"
;;   bibtex-completion-notes-path "~/Dropbox/orgfiles/org-ref-notes.org"
;;   )
;;   )

;; (use-package citeproc)

;; Install org-contrib
(use-package org-contrib)

;; Install org
(use-package org
  :config
  (setq org-latex-pdf-process '("latexmk -pdf -outdir=%o %f"))
  (setq org-export-with-smart-quotes t)

  ;; export citations
  (require 'ox-bibtex)

  ;; manage citations
  (require 'org-bibtex-extras)

  ;; ignore headline but include content when exporting
  (require 'ox-extra)
  (ox-extras-activate '(ignore-headlines))

  :custom (org-startup-indented t)
  :bind (:map org-mode-map
              ("<f12>" . org-bibtex-yank)))
```

7.16. plain-org-wiki

Usage:

- `plain-org-wiki name` to open a wiki page from `~/org/wiki/org` or create a new one.
- `plain-org-academic name` to open a academic page from `~/org/astronomy/org`.

Installation:

- Downloaded from github repo: https://github.com/alraban/org-recoll. Modified. DO NOT OVERWRITE.
- Stackoverflow page: https://stackoverflow.com/a/26675657/4635427

Modifications to the source code:

- Defined `plain-org-academic` similar to `plain-org-wiki`. It calls the following definitions:
  - `plain-org-academic-directory`

- `plain-org-academic-extra-dirs`
- `plain-org-academic-extra-files`
- `plain-org-academic-files`
- `plain-org-academic-find-file`

```
(use-package plain-org-wiki
  :ensure t
  :load-path "/Users/chongchonghe/dotfiles/emacs/packages/plain-org-wiki/"
  :config
  (setq plain-org-wiki-directory "~/org/wiki/org")
  (setq plain-org-academic-directory "~/org/astronomy/org")
  (setq plain-org-wiki-extra-dirs '("~/org/astronomy/org"))
  (global-set-key (kbd "C-M-w") 'plain-org-wiki))
```

7.17. Link to a Mail message

A link that opens a Mail message on macOS. The link should be in the following format:

```
[[message:<message_id@domain.com>][Apple Mail Link]]
```

```
(org-add-link-type "message"
 (lambda (id)
  (shell-command
   ;; (concat "open -a mail.app message:" id))))
   (concat "open message:" id))))
```

Source: https://emacs.stackexchange.com/a/46028/26582. Original answer applied `url-hexify-string` to `id`: `(url-hexify-string id)`.

7.18. org-fancy-priorities

```
(use-package org-fancy-priorities
  :diminish
  :ensure t
  :hook (org-mode . org-fancy-priorities-mode)
  :config
  (setq org-fancy-priorities-list '("🅰" "🅱" "🅲" "🅳" "🅴")))
```

7.19. org-reveal

```
(use-package ox-reveal)
(setq org-reveal-root "/Users/chongchonghe/local/reveal.js-master")
```

7.20. org preview

org-preview-html

```
(use-package org-preview-html)
```

skewer-mode

```
(use-package skewer-mode)
```

impatient-mode

```
(use-package impatient-mode)
```

7.21. org-hugo

```
(use-package ox-hugo
  :ensure t   ;Auto-install the package from Melpa
  :pin melpa  ;`package-archives' should already have ("melpa" .
"https://melpa.org/packages/")
  :after ox)
```

7.22. ox-gfm

```
(use-package ox-gfm)
(eval-after-load "org"
  '(require 'ox-gfm nil t))
```

7.23. liveorg

```
;; (add-to-list 'load-path (expand-file-name "~/.emacs.d/emacs-liveorg"))
;; (require 'liveorg)
(use-package liveorg
  :defer 5
  :load-path "~/.emacs.d/emacs-liveorg"
  :config
  (setq liveorg-browser "Safari.app")
  (define-key org-mode-map "\C-cE" 'liveorg-preview)
  )
```

7.24. org-inline-pdf

To enable inline PDF preview in Org buffers. Source: https://github.com/shg/org-inline-pdf.el

```
(use-package org-inline-pdf
  :ensure org
  :defer 5
  :mode ("\\.org\\'" . org-mode)
  :config
  (add-hook 'org-mode-hook #'org-inline-pdf-mode))
```

7.25. LibreOffice

```
;; This setup is tested on Emacs 24.3 & Emacs 24.4 on Linux/OSX
;; org v7 bundled with Emacs 24.3
(setq org-export-odt-preferred-output-format "pdf")
;; org v8 bundled with Emacs 24.4
(setq org-odt-preferred-output-format "pdf")
;; BTW, you can assign "pdf" in above variables if you prefer PDF format
;; "doc" for word document

;; Only OSX need below setup
(defun my-setup-odt-org-convert-process ()
  (interactive)
  (let ((cmd "/Applications/LibreOffice.app/Contents/MacOS/soffice"))
    (when (and (eq system-type 'darwin) (file-exists-p cmd))
      ;; org v7
      (setq org-export-odt-convert-processes '(("LibreOffice"
"/Applications/LibreOffice.app/Contents/MacOS/soffice --headless --convert-to
%f%x --outdir %d %i")))
      ;; org v8
      (setq org-odt-convert-processes '(("LibreOffice"
```

```
"/Applications/LibreOffice.app/Contents/MacOS/soffice --headless --convert-to
%f%x --outdir %d %i"))))
    ))
(my-setup-odt-org-convert-process)
```

8. yasnippet

The default snippets are located in `~/.emacs.d/elpa/yasnippet-0.12.2/snippets`. My own snippet files are located in `~/dotfiles/emacs/snippets`. To add more snippets for auto-completion, add snippets files into that folder.

8.1. Usage

`TAB`	hippie-expand: expand template at point
`M-x yas-visit-snippet-file`	yas-expand: expand template at point
`M-/` (spacemacs only)	Edit snippet files for current mode

8.2. Available snippets

org-mode	
`lisp`	`#+begin_src emacs-lisp` stuff
`img_`	`<img src=...` stuff

python-mode	
`np`	`import numpy as np`
`plt`	`import matplotlib.pyplot as plt`
`ifm`	`if __name__ == '__main__':`

8.3. Config

```
(setq-default mode-require-final-newline nil)

(use-package yasnippet
  :diminish yas-minor-mode
  ;; :init (yas-global-mode)
  :ensure t
  :config
  (setq yas-snippet-dirs '("~/dotfiles/emacs/snippets/yasnippet-snippets-
20210105.1346/snippets" "~/dotfiles/emacs/snippets/personal"))
  (yas-global-mode 1)
  ;; (add-to-list #'yas-snippet-dirs "~/dotfiles/emacs/snippets/yasnippet-
```

```
snippets-20210105.1346/snippets")
  ;; (add-to-list #'yas-snippet-dirs "~/dotfiles/emacs/snippets/personal")
  (yas-reload-all)
  ;; (progn
  ;;   (add-hook 'hippie-expand-try-functions-list 'yas-hippie-try-expand)
  ;;   ;; (setq yas-key-syntaxes '("w_" "w_." "^ "))
  ;;   (setq yas-installed-snippets-dir "~/dotfiles/emacs/snippets")
  ;;   (setq yas-expand-only-for-last-commands nil))
  )
```

Not used

```
  (defun my-yasnippet-config ()
    (require 'yasnippet)
    (setq yas-triggers-in-field t)
  ;;; https://superuser.com/questions/1006188/can-emacs-be-set-up-to-display-
python-code-in-python-mode-and-display-docstrings
                                          ;(add-to-list 'load-path
"~/.emacs.d/python-docstring-mode")
                                          ;(require 'python-docstring)
                                          ;(add-hook 'python-mode-hook (lambda
() (python-docstring-mode t)))
    (yas-minor-mode 1)
    ;; (defun my/autoinsert-yas-expand()
    ;;   "Replace text in yasnippet template."
    ;;   (yas/expand-snippet (buffer-string) (point-min) (point-max)))
    )
```

9. helm

Before we load any helm things, need to load helm-flx so it uses flx instead of helm's fuzzy matching. (Ref)

```
  (use-package helm-flx
    :init (helm-flx-mode +1))
```

A good reference: https://writequit.org/org/settings.html#sec-1-34

```
(use-package helm
  ;; :bind
  ;; (("C-M-z" . helm-resume)
  ;;  ("C-x C-f" . helm-find-files)
  ;;  ("C-h b" . helm-descbinds)
  ;;  ("C-x C-r" . helm-mini)
  ;;  ("C-x M-o" . helm-occur)
  ;;  ("M-y" . helm-show-kill-ring)
  ;;  ("C-h a" . helm-apropos)
  ;;  ("C-h m" . helm-man-woman)
  ;;  ("M-g >" . helm-ag-this-file)
  ;;  ("M-g ," . helm-ag-pop-stack)
  ;;  ("M-g ." . helm-do-grep)
  ;;  ("C-x C-i" . helm-semantic-or-imenu)
  ;;  ("M-x" . helm-M-x)
  ;;  ("C-x C-b" . helm-buffers-list)
  ;;  ("C-x C-r" . helm-mini)
  ;;  ("C-x b" . helm-mini)
  ;;  ("C-h t" . helm-world-time))
  :init
  (helm-mode 1)
  :config
  (global-set-key (kbd "M-x") #'helm-M-x)
```

```
  ;; (global-set-key (kbd "C-x r b") #'helm-filtered-bookmarks)
  ;; (global-set-key (kbd "M-C-b") #'helm-filtered-bookmarks)
  (global-set-key (kbd "M-C-o") #'helm-filtered-bookmarks)
  (global-set-key (kbd "C-x C-f") #'helm-find-files)
  ;; rebind tab to run persistent action
  (define-key helm-map (kbd "<tab>") 'helm-execute-persistent-action)
  ;; make TAB work in terminal
  (define-key helm-map (kbd "C-i") 'helm-execute-persistent-action)
  ;; list actions using C-z
  (define-key helm-map (kbd "C-z")  'helm-select-action)
  )
```

10. python

10.1. Basics

```
(defun my/turn-on-elpy-mode ()
  (interactive)
  (elpy-mode))

(use-package python
  :defer t
  :mode ("\\.py\\'" . python-mode)
  :interpreter ("python" . python-mode)
  ;; :hook hs-minor-mode
  :bind (:map python-mode-map
              ("C-c C-c" . compile)
              ("s-e" . my/turn-on-elpy-mode)
              )
  :config
  (setq python-shell-interpreter "/Users/chongchonghe/anaconda3/bin/python3")
  (define-key python-mode-map (kbd "C-c C-z") 'run-python)
  (define-key python-mode-map (kbd "<backtab>") 'python-back-indent)
  (setq python-python-command "/Users/chongchonghe/anaconda3/bin/python")
  (defun my-insert-comments (string)
    "Insert \label{ARG} \index{\nameref{ARG}} at point"
    (interactive "sString for \\label and \\nameref: ")
    (insert "##### "  string  " #####"))
  (define-key python-mode-map (kbd "<f5>") 'my-insert-comments)
  (defun my-insert-comments-block (string)
    "Insert \label{ARG} \index{\nameref{ARG}} at point"
    (interactive "sString for \\label and \\nameref: ")
    (insert "# {{{ "  string  "
  # }}}"))
  (define-key python-mode-map (kbd "<f6>") 'my-insert-comments-block)
  )
(add-hook 'python-mode-hook 'hs-minor-mode)
```

```
(use-package python
  :defer t
  :mode ("\\.py\\'" . python-mode)
  :interpreter ("python" . python-mode)
  :init
  (setq-default indent-tabs-mode nil)
  :hook hs-minor-mode
  ;; :bind (:map python-mode-map
  ;;         ("C-c C-c" . compile)
  ;;         ((kbd "<backtab>") . python-back-indent)
  ;;         )
```

```
  :config
  (setq python-indent-offset 4)
  (elpy-enable)
  ;; (elpy-use-ipython)
  (setq elpy-rpc-backend "jedi")
  ;; (add-hook 'elpy-mode-hook) ;;'py-autopep8-enable-on-save)
  (setq python-shell-interpreter "ipython"
      python-shell-interpreter-args "--simple-prompt -i")
  (defun my-insert-comments (string)
    "Insert \label{ARG} \index{\nameref{ARG}} at point"
    (interactive "sString for \\label and \\nameref: ")
    (insert "##### "  string  " #####"))
  (define-key python-mode-map (kbd "<f5>") 'my-insert-comments)
  )
```

10.2. Jedi

elpy include jedi???

```
  ;; (use-package jedi
  ;;        :ensure t)
```

10.3. elpy

Ref: https://medium.com/analytics-vidhya/managing-a-python-development-environment-in-emacs-43897fd48c6a

```
(use-package flycheck)

(use-package elpy
  :bind
  (:map elpy-mode-map
      ("C-M-n" . elpy-nav-forward-block)
      ("C-M-p" . elpy-nav-backward-block))
  :hook ((elpy-mode . flycheck-mode))
  ;; :init
  ;; (elpy-enable)
  :config
  (setq elpy-modules (delq 'elpy-module-flymake elpy-modules))
                              ; fix for MacOS, see
https://github.com/jorgenschaefer/elpy/issues/1550
  (setq elpy-shell-echo-output nil)
  (setq elpy-rpc-python-command "python3")
  (setq elpy-rpc-timeout 30)        ; elpy autopep8 call timeout
  )
;; (use-package elpy
;;   :ensure t
;;   :commands elpy-enable
;;   :init (with-eval-after-load 'python (elpy-enable))
;;   )
```

10.4. old

```
  (with-eval-after-load 'python
     ;; Disable elpy Vertical Guide Lines for Indentation
     (add-hook 'elpy-mode-hook (lambda () (highlight-indentation-mode -1)))
     (when (require 'flycheck nil t)
       (setq elpy-modules (delq 'elpy-module-flymake elpy-modules))
       (add-hook 'elpy-mode-hook 'flycheck-mode))
```

```
    ;;(require 'py-autopep8)
    ;; (define-key ein:notebook-mode-map (kbd "C-c C-x d")
    ;;   'ein:worksheet-delete-cell)
    ;; Autoinsert Python comments
    ;;(global-set-key (kbd "<f6>") 'my-insert-docstring)
    ;;(defun my-insert-docstring (string)
    ;;  "Insert \label{ARG} \index{\nameref{ARG}} at point"
    ;;  (interactive "sString for \\label and \\nameref: ")
    ;;  (insert '""" '  string  ' """'))
    ;; jedi, replaced by (setq elpy-rpc-backend "jedi")
    ;; (add-hook 'python-mode-hook 'jedi:setup)
    ;; (setq jedi:complete-on-dot t)
    (setq elpy-rpc-ignored-buffer-size 204800)
    )
```

Auto-insert (disabled). Ref. Replaced by yasnippet

```
  ;; ref: https://www.webscalability.com/blog/2018/07/auto-insert-snippet-for-
python-emacs/
  ;; insert python skeleton with auto-insert
  ;; (setq python-skeleton-autoinsert nil)
  ;; (eval-after-load 'autoinsert
  ;;   '(define-auto-insert
  ;;      '("\\.\\py\\'" . "python skeleton")
  ;;      '(""
  ;;        "#!/usr/bin/env python" \n
  ;;        "\"\"\" "
  ;;        (file-name-nondirectory (buffer-file-name)) \n \n
  ;;        "Author: Chong-Chong He (che1234@umd.edu)" \n
  ;;        "Written on " (format-time-string "%a, %e %b %Y.") \n
  ;;        "\"\"\"" \n
  ;;        \n
  ;;        "import numpy as np" \n
  ;;        "import matplotlib.pyplot as plt" \n
  ;;        \n
  ;;        > _ \n
  ;;        \n
  ;;        "if __name__ == '__main__':" \n
  ;;        "pass" \n \n)))
```

10.5. Snippets

Usage: `defg<tab>`. Ref: stackexchange, Library for automatically inserting python docstring in Google style

```
  (defun python-args-to-google-docstring (text &optional make-fields)
    "Return a reST docstring format for the python arguments in yas-text."
    (let* ((indent (concat "\n" (make-string (current-column) 32)))
           (args (python-split-args text))
       (nr 0)
           (formatted-args
          (mapconcat
           (lambda (x)
             (concat "    " (nth 0 x)
                     (if make-fields (format " ${%d:arg%d}" (cl-incf nr) nr))
                     (if (nth 1 x) (concat " \(default " (nth 1 x) "\)"))))
           args
           indent)))
      (unless (string= formatted-args "")
          (concat
```

```
            (mapconcat 'identity
                 (list "" "Args:" formatted-args)
                 indent)
            "\n"))))
```

10.6. Anaconda-mode

```
  (use-package anaconda-mode)
```

11. Julia mode

```
(use-package julia-mode)
```

12. Markdown

12.1. Links

- https://jblevins.org/projects/markdown-mode/

12.2. Key-bindings

C-c LEFT/RIGHT	Downmote/Promote heading, list, etc
C-c C-x C-m	markdown-toggle-markup-hiding

12.3. Basic configures

```
(use-package markdown-mode
  :ensure t
  :mode ("README\\.md\\'" . gfm-mode)
  :init
  (setq markdown-command "multimarkdown")
)
(eval-after-load 'markdown-mode
'(progn
  (define-key evil-normal-state-map (kbd "TAB") 'markdown-cycle)
(define-key markdown-mode-map (kbd "M-n") 'markdown-outline-next)
(define-key markdown-mode-map (kbd "M-p") 'markdown-outline-previous))
)
```

Keybindings

```

```

Use tab as indent:

```
  (add-hook 'text-mode-hook
          '(lambda ()
             (setq indent-tabs-mode t)
             (setq tab-width 4)))
```

12.4. Livedown

```
(add-to-list 'load-path (expand-file-name "~/.emacs.d/emacs-livedown"))
(require 'livedown)
```

12.5. Marked 2.app

Source: https://www.nistara.net/post/2016-07-21-emacs-marked/

```
;; Getting emacs to use the 'Marked' app
(defun markdown-preview-file ()
  "run Marked on the current file and revert the buffer"
  (interactive)
  (shell-command
   (format "open -a /Applications/Marked\\ 2.app %s"
       (shell-quote-argument (buffer-file-name))))
)
(eval-after-load 'markdown-mode
  '(define-key markdown-mode-map (kbd "C-c m") 'markdown-preview-file))
(defalias 'marked 'markdown-preview-file)
```

12.6. Beautify

https://emacs.stackexchange.com/questions/14740/how-to-configure-markdown-mode-to-render-headings-like-org-mode

`C-c C-x C-m` to toggle markup hiding.

13. Latex

13.1. Basics

```
(use-package latex
  :defer t
  :ensure auctex
  :mode ("\\.tex\\'" . LaTeX-mode)
  :bind
  (:map LaTeX-mode-map
      ("M-n" . outline-next-heading)
      ("M-p" . outline-previous-heading)
      ("C-c C-c" . TeX-command-run-all)
      ("C-c l" . TeX-error-overview)
      ;; ((kbd "C-tab") . TeX-complete-symbol)
      ("C-c w" . juanjo:textcount))
  :config
  (setq TeX-auto-save t)
  (setq TeX-auto-save t)
  (setq TeX-PDF-mode t)       ;; Compile documents to PDF by default
  (setq TeX-parse-self t)
  (setq TeX-save-query nil)
  ;; (setq-default TeX-master nil) ;; Make emacs aware of multi-file projects
  (add-hook 'LaTeX-mode-hook #'visual-line-mode)
  (add-hook 'LaTeX-mode-hook #'no-auto-fill)
  (add-hook 'LaTeX-mode-hook 'hs-minor-mode)
  (add-hook 'LaTeX-mode-hook 'outline-minor-mode)
  (add-hook 'LaTeX-mode-hook 'flyspell-mode)
  (add-hook 'LaTeX-mode-hook 'LaTeX-math-mode)
```

```
  (evil-define-key 'normal outline-minor-mode-map (kbd "SPC") 'evil-toggle-
fold)
  ;; CDLaTeX
  (add-hook 'LaTeX-mode-hook 'turn-on-cdlatex)   ; with AUCTeX LaTeX mode
  ;; (setq reftex-plug-into-auctex t)
  (add-hook 'LaTeX-mode-hook 'turn-on-reftex)
  (setq reftex-plug-into-AUCTeX t)
  (autoload 'helm-bibtex "helm-bibtex" "" t)
  (electric-pair-mode)
  ;; compile
  ;; (evil-define-key 'normal LaTeX-mode-map (kbd ", l") 'TeX-command-master)
  ;; do not query the user before saving each file with TeX-save-document
  (setq TeX-save-query nil)
  (evil-define-key 'normal LaTeX-mode-map (kbd ", l") 'TeX-command-run-all)
  (evil-define-key 'normal LaTeX-mode-map (kbd ", v") 'TeX-view)
  (evil-define-key 'normal LaTeX-mode-map (kbd "M-w") 'LaTeX-fill-region)
  ;; sync
  ;; Enable the clicking feature of the sync
  (add-hook 'LaTeX-mode-hook
          (lambda () (local-set-key (kbd "<S-s-mouse-1>") #'TeX-view))
          )
  (setq TeX-PDF-mode t)       ;; Compile documents to PDF by default
  ;; Use Skim as viewer, enable source <-> PDF sync
  ;; make latexmk available via C-c C-c
  ;; Note: SyncTeX is setup via ~/.latexmkrc (see below)
  (add-hook 'LaTeX-mode-hook (lambda ()
                        (push
                       '("latexmk" "latexmk -pdf %s" TeX-run-TeX nil t :help
"Run latexmk on file")
                       TeX-command-list)))
  (add-hook 'TeX-mode-hook '(lambda () (setq TeX-command-default "latexmk")))
  ;; use Skim as default pdf viewer
  ;; Skim's displayline is used for forward search (from .tex to .pdf)
  ;; option -b highlights the current line; option -g opens Skim in the
background
  (setq TeX-view-program-selection '((output-pdf "PDF Viewer")))
  (setq TeX-view-program-list
      '(("PDF Viewer" "/Applications/Skim.app/Contents/SharedSupport/displayline
-b -g %n %o %b")))
  ;; keybindings
  ;; (define-key outline-mode-map [M-left] 'outline-hide-body)
  ;; (define-key outline-mode-map [M-right] 'outline-show-all)
  ;; (define-key outline-mode-map [M-up] 'outline-previous-heading)
  ;; (define-key outline-mode-map [M-down] 'outline-next-heading)
  ;; (define-key outline-mode-map [C-M-left] 'outline-hide-sublevels)
  ;; (define-key outline-mode-map [C-M-right] 'outline-show-children)
  ;; (define-key outline-mode-map [C-M-up] 'outline-previous-visible-heading)
  ;; (define-key outline-mode-map [C-M-down] 'outline-next-visible-heading)
  (defun turn-on-outline-minor-mode () (outline-minor-mode 1))
  (add-hook 'LaTeX-mode-hook 'turn-on-outline-minor-mode)
  (add-hook 'LaTeX-mode-hook 'turn-on-outline-minor-mode)
  (defun turn-on-flycheck-mode () (flycheck-mode 1))
  (add-hook 'LaTeX-mode-hook 'turn-on-flycheck-mode)
  )
```

Set preview image format as svg. Note: this rely on the use of [Yamamoto's Emacs a.k.a. Emacs Mac Port](), which has `svg` support. Regular emacs downloaded from emacsformacosx.com will not work.

Source: [https://emacs.stackexchange.com/a/34085]()

```
(setq org-latex-create-formula-image-program 'dvisvgm)
```

```
(setq org-preview-latex-default-process 'dvisvgm)
;; (setq org-preview-latex-default-process 'divpng)
;; (setq my:dvi-to-svg
;;      (my:dvi-to-svg :programs
;;          ("latex" "dvisvgm")
;;             :description "dvi > svg"
;;             :message "you need to install the programs: latex and dvisvgm."
;;             :use-xcolor t
;;             :image-input-type "dvi"
;;             :image-output-type "svg"
;;             :image-size-adjust (1.7 . 1.5)
;;             :latex-compiler ("latex -interaction nonstopmode -output-
directory %o %f")
;;             :image-converter ("dvisvgm %f -e -n -b min -c %S -o %O")))
;; (with-eval-after-load 'ox-latex
;;      (add-to-list 'org-preview-latex-process-alist my:dvi-to-svg)
;;      (setq org-preview-latex-default-process 'my:dvi-to-svg))
(use-package preview-dvisvgm)
```

```
  ;; not used

  (global-set-key [M-left] 'outline-hide-body)
  (global-set-key [M-right] 'outline-show-all)
  (global-set-key [M-up] 'outline-previous-heading)
  (global-set-key [M-down] 'outline-next-heading)
  (global-set-key [C-M-left] 'outline-hide-sublevels)
  (global-set-key [C-M-right] 'outline-show-children)
  (global-set-key [C-M-up] 'outline-previous-visible-heading)
  (global-set-key [C-M-down] 'outline-next-visible-heading)

  ;; AucTeX
  (setq reftex-default-bibliography
'("/Users/chongchonghe/Academics/Bib/BIB_HE.bib") )
  (setq helm-bibtex-bibliography
'("/Users/chongchonghe/Academics/Bib/BIB_HE.bib") )
  (setq reftex-default-bibliography
'("/Users/chongchonghe/Academics/Bib/BIB_HE.bib"))
  (setq helm-bibtex-bibliography
'("/Users/chongchonghe/Academics/Bib/BIB_HE.bib"))


  (add-hook 'LaTeX-mode-hook
                       '(lambda ()
             (define-key LaTeX-mode-map (kbd "$") 'self-insert-command)
             ))


  (require 'smartparens-config)
  (add-hook 'LaTex-mode-hook #'smartparens-mode)
```

13.2. citation

```
(use-package reftex)
(setq reftex-default-bibliography
      '("/Users/chongchonghe/Academics/Bib/BIB_HE.bib"))
;; (setq reftex-default-bibliography
;;          '("/Users/chongchonghe/Academics/Bib/BIB_HE.bib",
;;    "/Users/chongchonghe/Academics/Bib/Books.bib",
;;    "/Users/chongchonghe/Academics/Bib/Bib_HE_PhD.bib"))
```

```
(setq reftex-external-file-finders
      '(("tex" . "/path/to/kpsewhich -format=.tex %f")
      ("bib" . "/path/to/kpsewhich -format=.bib %f")))
```

14. Other packages

14.1. windresize

```
  (use-package windresize
            :defer t
            :bind
            ("C-c w" . windresize)
            )
```

14.2. clipboard2org (not working)

https://github.com/itf/clipboard2org

```
(use-package clipboard2org
  :load-path "/Users/chongchonghe/.emacs.d/pkgs")
```

14.3. html2org-clipboard

Source: https://stackoverflow.com/a/64408897 Source2: https://emacs.stackexchange.com/questions/12121/org-mode-parsing-rich-html-directly-when-pasting

```
(defun my-html2org-clipboard ()
  "Convert clipboard contents from HTML to Org and then paste (yank)."
  (interactive)
  (setq cmd "osascript -e 'the clipboard as \"HTML\"' | perl -ne 'print chr
foreach unpack(\"C*\",pack(\"H*\",substr($_,11,-3)))' | pandoc -f html -t json
| pandoc -f json -t org --wrap=none")
  (kill-new (shell-command-to-string cmd))
  (yank))

(define-key org-mode-map (kbd "s-V") #'my-html2org-clipboard)
```

15. Keybindings

15.1. Make swithing windows easier

```
  ;; ;; make swithing windows easier
  (global-set-key (kbd "M-p") (kbd "C-- C-x o"))
  (global-set-key (kbd "M-n") (kbd "C-x o"))
  (global-set-key (kbd "M-j") 'windmove-down)
  (global-set-key (kbd "M-k") 'windmove-up)
  (global-set-key (kbd "M-h") 'windmove-left)
  (global-set-key (kbd "M-l") 'windmove-right)
  ;; (global-set-key (kbd "M-j") 'evil-window-down)
  ;; (global-set-key (kbd "M-k") 'evil-window-up)
  ;; (global-set-key (kbd "M-h") 'evil-window-left)
  ;; (global-set-key (kbd "M-l") 'evil-window-right)
  (define-key evil-normal-state-map (kbd "M-h") #'evil-window-left)
  (define-key evil-normal-state-map (kbd "M-j") #'evil-window-down)
```

```
  (define-key evil-normal-state-map (kbd "M-k") #'evil-window-up)
  (define-key evil-normal-state-map (kbd "M-l") #'evil-window-right)

  (use-package transpose-frame)

  (global-set-key (kbd "<f12>") 'next-buffer)
  (global-set-key (kbd "<f11>") 'previous-buffer)
```

15.2. super keys

```
  (global-set-key (kbd "s-v") 'clipboard-yank)
  (global-set-key (kbd "s-k") 'kill-current-buffer)
  (global-set-key (kbd "s-e") 'eval-region)
  (global-set-key (kbd "s-b") 'eval-buffer)
  (global-set-key (kbd "s-c") 'compile)
  (global-set-key (kbd "s-r") 'recompile)
  (global-set-key (kbd "s-,") 'previous-buffer)
  (global-set-key (kbd "s-.") 'next-buffer)
  ;; (global-unset-key (kbd "s-j"))
  ;; (global-set-key (kbd "s-j") 'jump-to-register)

  ;; (global-set-key (kbd "M-v") 'evil-paste-after)
```

15.3. Clipboard

```
  (global-set-key (kbd "M-v") 'clipboard-yank)
```

15.4. Adjust window size

```
(fset 'my/shrink (kbd "C-u 39 C-x {"))
```

16. Smooth scroll

```
  ;; (use-package smooth-scroll
  ;;       :config
  ;;       (smooth-scroll-mode 1)
  ;;       (setq smooth-scroll/vscroll-step-size 5)
  ;;       )
  ;; (use-package smooth-scrolling
  ;;        :config
  ;;        (smooth-scrolling-mode 1))
```

17. tramp

Make tramp faster:

```
  (setq remote-file-name-inhibit-cache nil)
  (setq vc-ignore-dir-regexp
                (format "%s\\|%s"
                                vc-ignore-dir-regexp
                                tramp-file-name-regexp))
  (setq tramp-verbose 1)

(setq server-use-tcp t
      server-port    9999)
(defun server-start-and-copy ()
  (server-start)
  (copy-file "~/.emacs.d/server/server" "/des:.emacs.d/server/server" t))
```

```
(add-hook 'emacs-startup-hook 'server-start-and-copy)
```

18. hide-show

```
  (add-hook 'hs-minor-mode-hook
            (lambda ()
              ;;(local-set-key (kbd "C-c p") 'hs-toggle-hiding)
              ;; (local-set-key (kbd "SPC") 'hs-toggle-hiding)
              (local-set-key (kbd "C-c h") 'hs-hide-all)
              (local-set-key (kbd "C-c s") 'hs-show-all)
              (local-set-key (kbd "C-c l") 'hs-hide-level)))
  (add-hook 'emacs-lisp-mode-hook 'hs-minor-mode)
  (evil-define-key 'normal hs-minor-mode-map (kbd "SPC") 'hs-toggle-hiding)

  ;; (add-hook 'hs-minor-mode-hook 'my-hideshow-config)
  ;; (defun my-hideshow-config ()
  ;;       "For use in 'hs-minor-mode-hook'."
  ;;       ;;(local-set-key (kbd "C-c p") 'hs-toggle-hiding)
  ;;       ;; (local-set-key (kbd "SPC") 'hs-toggle-hiding)
  ;;       (local-set-key (kbd "C-c h") 'hs-hide-all)
  ;;       (local-set-key (kbd "C-c s") 'hs-show-all)
  ;;       (local-set-key (kbd "C-c l") 'hs-hide-level)
  ;;       )
  ;; (add-hook 'emacs-lisp-mode-hook 'hs-minor-mode)
  ;; (evil-define-key 'normal hs-minor-mode-map (kbd "SPC") 'hs-toggle-hiding)
```

19. Some Automatics

19.1. Initial my tasks.org view

Not working…

```
  (defun taskinit ()
            (interactive)
            (split-window-right)
            ((kbd "C-u 10 C-x {"))
            (set-frame-height (selected-frame) 60)
            )
```

20. dashboard

- https://github.com/rakanalh/emacs-dashboard
- https://emacs.stackexchange.com/questions/14282/replace-splash-screen-with-list-of-recentf

```
(use-package dashboard
  :ensure t
  :diminish dashboard-mode
  :config
  (setq dashboard-banner-logo-title "Welcome! Work day.")
  ;; (setq dashboard-startup-banner "/path/to/image")
  (setq dashboard-items '((recents  . 10)
                          (bookmarks . 10)))
  (dashboard-setup-startup-hook)
  )
```

21. Folding mode

21.1. Usage

Folding block are defined as # {{{ and # }}} pairs (in python mode). Use the snippet '#{<tab>' to quick insert. Use <F9> to toggle folding.

Source: https://www.emacswiki.org/emacs/FoldingMode#toc6

21.2. Config

```
  (use-package folding
      :ensure t
      :load-path "/Users/chongchonghe/dotfiles/emacs/packages/project-emacs--
folding-mode"
      :config
      (folding-mode-add-find-file-hook)
      (add-hook 'folding-mode
              (lambda () (local-set-key [f9] 'folding-toggle-show-hide)))
      )
```

22. f.el

```
  (use-package helm-org)
  (use-package f)
```

23. layout-restore.el

Old and causing problem. Not using.

Ref: https://www.emacswiki.org/emacs/layout-restore.el

```
;; (use-package layout-restore
;;   :defer nil
;;   :load-path "~/dotfiles/emacs/packages/layout-restore.el"
;;   :config
;;   (global-set-key (kbd "C-c l") 'layout-save-current)
;;   (global-set-key (kbd "C-c r") 'layout-restore)
;;   (global-set-key (kbd "C-c d") 'layout-delete-current)
;;   )

;; (require 'layout-restore)
;; (global-set-key (kbd "C-c l") 'layout-save-current)
;; (global-set-key (kbd "C-c r") 'layout-restore)
;; (global-set-key (kbd "C-c d") 'layout-delete-current)
```

24. Neotree

```
(use-package neotree
  :config
  (defun my-neotree-mode-config ()
    "For use in 'neotree-mode-hook'."
    ;; (local-set-key (kbd "j") 'neotree-next-line)
    ;; (local-set-key (kbd "k") 'neotree-previous-line)
    (local-set-key (kbd "C-j") 'neotree-change-root)
    (local-set-key (kbd "C-k") 'neotree-select-up-node)
```

```
    (local-set-key (kbd "<return>") 'neotree-enter)
    ;; (local-set-key (kbd "SPC") 'neotree-quick-look)
    (evil-define-key 'normal neotree-mode-map (kbd "SPC") 'neotree-quick-look)
    ;; (with-eval-after-load 'neotree
    ;;   (define-key neotree-mode-map (kbd "<return>") 'neotree-enter))
    (define-key evil-normal-state-local-map (kbd "q") 'neotree-hide)
    (define-key evil-normal-state-local-map (kbd "g") 'neotree-refresh)
    (define-key evil-normal-state-local-map (kbd "A") 'neotree-stretch-toggle)
    (define-key evil-normal-state-local-map (kbd "H") 'neotree-hidden-file-
toggle)
    )
  (add-hook 'neotree-mode-hook 'my-neotree-mode-config))

(add-hook 'neotree-mode-hook
        (lambda () (define-key evil-motion-state-local-map (kbd "g") 'neotree-
refresh)))
```

25. Other major and minor modes

25.1. matlab

```
;; (use-package matlab-mode)
```

26. Themes and fonts

26.1. Not used

The following is an example of installing a theme

```
;;customize theme
(add-to-list 'custom-theme-load-path "~/.emacs.d/moe-theme.el/")
(add-to-list 'load-path "~/.emacs.d/moe-theme.el/")
(require 'moe-theme)
```

The tsdh theme (disabled, replaced with doom theme)

```
;;customize theme
(add-to-list 'custom-theme-load-path "~/.emacs.d/themes/")
(add-to-list 'load-path "~/.emacs.d/themes/")
(require 'tsdh-light-theme)
;; (set-background-color "#d4cec3")
;; (set-background-color "#e6e3df")
(set-background-color "#f5efe6")
```

26.2. Doom theme

```
(defun my-light-theme ()
  (interactive)
  (load-theme 'doom-one-light t)
  ;; (set-background-color "#e6e3df")
  (set-background-color "#fffcf7")
  (my-org-set-light-todo-faces)
  )
(defun my-dark-theme ()
  (interactive)
  (load-theme 'doom-one t)
  (set-foreground-color "#eee")
```

```
  (my-org-set-dark-todo-faces)
  )
(use-package doom-themes
  :ensure t
  :config
  ;; Global settings (defaults)
  (setq doom-themes-enable-bold t    ; if nil, bold is universally disabled
        doom-themes-enable-italic t) ; if nil, italics is universally disabled
  ;; (load-theme 'doom-one t)
  ;; (load-theme 'doom-one-light t)
  (my-light-theme)
  )
;; Default font
(set-face-attribute 'default nil :font "Monaco-16")
```

26.3. Mixed-pitch

Ref: https://gitlab.com/jabranham/mixed-pitch

```
;; mixed-pitch
(use-package mixed-pitch
  :hook
  ;; If you want it in all text modes:
  (text-mode . mixed-pitch-mode)
  ;; not sure. ref: https://emacs-china.org/t/doom-emacs-mixed-pitch-set-
height-org/16558
  ;; (mixed-pitch-mode . #'solaire-mode-reset)
  :config
  (set-face-attribute 'variable-pitch nil :family "Helvetica Neue" :height 1.2)
  (set-face-attribute 'fixed-pitch nil :family "Monaco" :height 1.0)
  ;; bigger text for org-mode headings
  (custom-theme-set-faces
   'user
   `(org-level-4 ((t (:inherit outline-4 :height 1.0))))
   `(org-level-3 ((t (:inherit outline-3 :height 1.1))))
   `(org-level-2 ((t (:inherit outline-2 :height 1.2))))
   `(org-level-1 ((t (:inherit outline-1 :height 1.3))))
   `(org-document-title ((t (:family "Helvetica Neue" :height 1.6 :underline
nil))))
   )
  (setq mixed-pitch-set-height t)
  )
```

26.4. Chinese font

```
;;;中文与英文字体设置
;; Setting English Font
;; (set-face-attribute
;;  'default nil :font "Monaco 14")
;; Chinese Font
(dolist (charset '(kana han symbol cjk-misc bopomofo))
  (set-fontset-font (frame-parameter nil 'font)
                    charset (font-spec :family "STHeiti" :height 1.0)))
;; more options:
;; "PingFang SC": ugly
;; "STXihei": thicker
```

26.5. Window size and more

Window size (wide high, in chars)

```
(when window-system (set-frame-size (selected-frame) 130 50))
```

Here are some more sofisticated config (not being used)

```
  ;; Define variable font
  (let* ((variable-tuple
          (cond ((x-list-fonts "ETBembo")          '(:font "ETBembo"))
                ((x-list-fonts "Source Sans Pro") '(:font "Source Sans Pro"))
                ((x-list-fonts "Lucida Grande")   '(:font "Lucida Grande"))
                ((x-list-fonts "Verdana")         '(:font "Verdana"))
                ((x-family-fonts "Sans Serif")    '(:family "Sans Serif"))
                (nil (warn "Cannot find a Sans Serif Font.  Install Source Sans
Pro."))))))

    (custom-theme-set-faces
     'user
     `(org-level-7 ((t (:inherit outline-7 ,@variable-tuple))))
     `(org-level-6 ((t (:inherit outline-6 ,@variable-tuple))))
     `(org-level-5 ((t (:inherit outline-5 ,@variable-tuple))))
     `(org-level-4 ((t (:inherit outline-4 ,@variable-tuple :height 1.1))))
     `(org-level-3 ((t (:inherit outline-3 ,@variable-tuple :height 1.25))))
     `(org-level-2 ((t (:inherit outline-2 ,@variable-tuple :height 1.5))))
     `(org-level-1 ((t (:inherit outline-1 ,@variable-tuple :height 1.75))))
     `(org-document-title ((t (,@variable-tuple :height 2.0 :underline nil))))
     ))


  ;; (custom-theme-set-faces
  ;;  'user
  ;;  ;; '(variable-pitch ((t (:inherit default :family "ETBembo" :height
180))))
  ;;  '(variable-pitch ((t (:inherit default :family "Source Serif Pro" :height
180))))
  ;;  ;; '(fixed-pitch ((t (:inherit default :family "Source Font Pro" :height
140))))
  ;;  ;; '(org-table   ((t (:inherit fixed-pitch))))
  ;;  ;; '(org-block   ((t (:inherit fixed-pitch))))
  ;;  ;; '(org-block-begin-line :foreground fg :slant 'italic :inherit 'fixed-
pitch)
  ;;  )
```

27. Keybinding in the end

27.1. Global keys

```
(global-set-key (kbd "C-c c") 'org-capture)
(global-set-key (kbd "C-c a") 'org-agenda)
;; (global-set-key (kbd "C-c a") 'org-agenda-list)
;; (define-key yas-minor-mode-map (kbd "C-c t") nil)
;; (global-set-key (kbd "C-c t") 'org-todo-list)
;; ;; (bind-key* "C-c t" 'org-todo-list)
;; (global-set-key (kbd "<f9>") 'org-todo-list)
(defun my/kill-other-buffers ()
      "Kill all other buffers."
      (interactive)
      (mapc 'kill-buffer (delq (current-buffer) (buffer-list))))
```

```
(defun my/kill-all-buffers ()
      "Kill all other buffers."
      (interactive)
      (mapc 'kill-buffer (buffer-list)))
```

27.2. org-mode keymap

```
;; (with-eval-after-load 'org
;;   (define-key org-agenda-mode-map (kbd "j") #'org-agenda-next-item)
;;   (define-key org-agenda-mode-map (kbd "k") #'org-agenda-previous-item))
```

27.3. my-keys-minor-mode (not using)

https://stackoverflow.com/questions/683425/globally-override-key-binding-in-emacs/5340797#5340797

```
(defvar my-mode-map
  (let ((map (make-sparse-keymap)))
    (
define-key map (kbd "C-i") 'some-function)
    map)
  "my-keys-minor-mode keymap.")

(define-minor-mode my-mode
  "A minor mode so that my key settings override annoying major modes."
  :init-value t
  :lighter " my-keys")

(my-mode 1)

(defun my-minibuffer-setup-hook ()
  (my-mode 0))

(add-hook 'minibuffer-setup-hook 'my-minibuffer-setup-hook)

(define-key my-mode-map (kbd "C-c t") 'org-todo-list)
```

```
(require 'bind-key)
(bind-key* "C-c t" 'org-todo-list)
```

28. Try (TODO)

```
(defvar org-babel-eval-verbose t
  "A non-nil value makes `org-babel-eval' display")

(defun org-babel-eval (cmd body)
  "Run CMD on BODY.
If CMD succeeds then return its results, otherwise display
STDERR with `org-babel-eval-error-notify'."
  (let ((err-buff (get-buffer-create " *Org-Babel Error*")) exit-code)
    (with-current-buffer err-buff (erase-buffer))
    (with-temp-buffer
      (insert body)
      (setq exit-code
            (org-babel--shell-command-on-region
             (point-min) (point-max) cmd err-buff))
      (if (or (not (numberp exit-code)) (> exit-code 0)
              (and org-babel-eval-verbose (> (buffer-size err-buff) 0))) ; new
condition
```

```
          (progn
            (with-current-buffer err-buff
              (org-babel-eval-error-notify exit-code (buffer-string)))
            nil)
        (buffer-string)))))
```

- insert file content into .el file
  - https://stackoverflow.com/questions/34432246/how-to-read-contents-of-the-file-programmatically-in-emacs

29. Startup

```
;; (defun my-shrink ()
;;   (interactive)
;;   ;; (funcall (key-binding (kbd "C-u 39 C-x {")))
;;   ;; (call-interactively (key-binding (kbd "C-u 39 C-x {")))
;;   ;; (/ (loop repeat 39 collect (key-binding (kbd "C-u 39 C-x {"))))
;;   ;; (/ (loop repeat n sum (funcall f arg)) n)
;;   ;; (cl-loop repeat 39 (shrink-window-horizontally))
;;   ;; (r 39 'shrink-window-horizontally 'nil')
;;   'shrink-window-horizontally
;;   'shrink-window-horizontally
;;   'shrink-window-horizontally
;;   'shrink-window-horizontally
;;   )

;; Saving Emacs Sessions
;; (desktop-save-mode 1)

;; (fset 'my-shrink (kbd "C-u 43 C-x {"))
(defun my-shrink ()
  (interactive)
  ;; (shrink-window-horizontally 60)
  (evil-window-set-width 80)
  )
(defun my-todo ()
  (interactive)
  (split-window-right)
  (evil-window-right 1)
  ;; (find-file "~/Dropbox/orgfiles/work.org")
  (org-agenda-list)
  (my-shrink)
  )

;; open agenda list
(org-agenda-list)
```