Emacs Documentation

Release latest

Jun 04, 2021

Documentation

The road to Emacs is not easy: I have tried to use Emacs for many years, and started using on daily basis from Jun 2014. The transition is difficulty, and full of tears and bloody, and every day I feel like being doomed in the dark. About Jan 2015, I start to the light. And conquered Emacs, and it now becomes a symbol of me, and I use it does most of productive work.

As the configuration grows bigger and bigger, a single *init.el* is not suitable for organising, testing, and expanding any more. Previously, I have about 7 `.el` files, for example, `setup-org.el`, `setup-email.el`, and I document on each file. Inspired by Sachua's new posts, I think it would be a brilliant idea to org Emacs configuration code into one single org file, letting along the convince of organising and share my settings, the precious thing I would appreciate is it provide a way I could start with a long comments, thoughts or workflow. In this way, the code becomes less important as it should be.

My configuration file is initially separated by different purpose or mode, but as it growths, it became probelmatic in tracking, and, As for other Emacs user, my configuration is keep growing, and This documents is first combined by 5 configuration files, and it keeps expanding, I use literate programming to include all the notes, and keep a log of how I use them

This Emacs configuration is free of bug and fully tested on Ubuntu and OS X.

Normally you could tangle an org file to extract all the source code into one file, that you could use. But I would like to push literate programming further in two aspects: 1) the source code takes input from this org file, I.e. table. 2) it facility Babel Library to integrate not only Emacs Lisp, but also sh and R functions that could be run in Emacs, and I found it particular useful.

CHAPTER 1

Licence

```
Copyright (C)  2015  Yi Tang
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

Code in this document is free software: you can redistribute it
and/or modify it under the terms of the GNU General Public
License as published by the Free Software Foundation, either
version 3 of the License, or (at your option) any later version.

This code is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.
```

CHAPTER 2

How to Use This Document

Add the following scripts to your *.emacs*

```
(load-file "~/git/.emacs.d/load_config.el")
```

CHAPTER 3

Install Packages

All the packages used in this configuration are available the major Emacs repositories. It is recommended to install them via Emacs's package management tool, and by the following snippets. Note the initial installation may takes a while to finish, depends on how many are already in the *~/emacs.d/elpa/* folder.

CHAPTER 4

General

4.1 Utilities

Firstly, define a function for reloading Emacs configuration, need this function in debugging this configuration file.

```
(defun yt/reload-dot-emacs ()
  "Save the .emacs buffer if needed, then reload .emacs."
  (interactive)
  (let ((dot-emacs "~/.emacs"))
    (and (get-file-buffer dot-emacs)
     (save-buffer (get-file-buffer dot-emacs)))
    (load-file dot-emacs))
  (message "Re-initialized!"))
(setq confirm-kill-emacs 'y-or-n-p)
```

Life is too short to type “yes” or “no”. ‘y’ or ‘n’ is enough.

```
(fset 'yes-or-no-p 'y-or-n-p)
```

4.2 Remove Keybind

```
(global-unset-key (kbd "C-x b"))
(global-unset-key (kbd "C-x C-b"))
(global-unset-key (kbd "C-x C-c"))  ;; save-buffers-kill-terminal
(global-unset-key (kbd "C-x o"))  ;; other window. replace by f2 - ace-window.
```

4.3 Assorted Pieces

Automatically backup buffers/files into the working directory and the ~.emacs.d/backup// directory.

```
;; ref: http://stackoverflow.com/questions/151945/how-do-i-control-how-emacs-makes-
↪backup-files
;; save all backup files (foo~) to this directory.
(setq backup-directory-alist '(("." . "~/.emacs.d/backup"))
      backup-by-copying t    ; Don't delink hardlinks
      version-control t      ; Use version numbers on backups
      delete-old-versions t  ; Automatically delete excess backups
      kept-new-versions 20   ; how many of the newest versions to keep
      kept-old-versions 5    ; and how many of the old
      auto-save-timeout 20   ; number of seconds idle time before auto-save (default:␣
↪30)
      auto-save-interval 200 ; number of keystrokes between auto-saves (default: 300)
      )

;; guide-key package
;; (require 'guide-key)
;; (setq guide-key/guide-key-sequence t) ;; on for all key-bindings
;; (guide-key-mode 1)

;; use company for all except few modes
(require 'company)
(add-hook 'after-init-hook 'global-company-mode)
;; Don't enable company-mode in below major modes, OPTIONAL
(setq company-global-modes '(not eshell-mode comint-mode erc-mode rcirc-mode))

;; config company mode
(setq company-selection-wrap-around t
      company-tooltip-align-annotations t
      company-idle-delay 0.36
      company-minimum-prefix-length 2
      company-tooltip-limit 10)

(setq company-ddabbrev-code-everywhere t)
(setq company-dabbrev-code-modes t)
(setq company-dabbrev-code-other-buffers 'all)
(setq company-dabbrev-ignore-buffers "\\`\\'")
(setq company-dabbrev-char-regexp "\\(\\sw\\|\\s_\\|_\\|-\\)")

;; config company for ESS mode
(defun yt/ess_company_mode_setup ()
  ;; this is really important. to source vairbales defined in the scripts.
     (make-local-variable 'company-backends)
     (add-to-list 'company-backends 'company-dabbrev-code)
     )
(add-hook 'ess-mode-hook 'yt/ess_company_mode_setup)


(defun text-mode-hook-setup ()
  (make-local-variable 'company-backends)
  (add-to-list 'company-backends 'company-ispell)
  ;; (setq company-ispell-dictionary (file-truename "~/git/.emacs.d/english_words.txt
↪"))
  )

(add-hook 'text-mode-hook 'text-mode-hook-setup)
(company-quickhelp-mode 1)
(define-key company-active-map (kbd "M-h") #'company-quickhelp-manual-begin)
```

(continues on next page)

(continued from previous page)

```
(define-key company-active-map (kbd "M-h") 'company-show-doc-buffer)

(setq company-dabbrev-downcase nil)
(setq company-show-numbers t)
```

Configure recent opened files. I use `helm-mini` to navigate between files, which is a lot convenient and faster than actually locate the file path.

```
(recentf-mode 1)
(setq recentf-max-saved-items 200
      recentf-max-menu-items 15)
```

Shows an notication for invalid operations.

```
(setq visible-bell nil)
(setq ring-bell-function 'ignore)
```

Disable startup message

```
(setq inhibit-startup-message t)
```

yasnippet is a powerful package that I'd like to explore in the future, and this stage, I turned if off since it will slow down the start-up.

```
(require 'yasnippet)
(yas/global-mode 1)
(add-to-list 'yas/snippet-dirs "~/git/.emacs.d/snippets" t)
(yas/reload-all)
```

4.4 Window Layout/Navigation

Quickly jump between windows using `ace-window`, I used it frequently and bind it `F2`.

```
(require 'ace-window)
(global-set-key (kbd "<f2>") 'ace-window)
(global-set-key (kbd "M-o") 'ace-window)
(setq aw-scope 'frame)
```

Instead of equally split the window size, it make a lot sense to have the current window, the one I am working one, has bigger size.

```
(require 'golden-ratio)
(golden-ratio-mode 1)
(add-to-list 'golden-ratio-extra-commands 'ace-window) ;; active golden ratio when␣
↪using ace-window
```

Some actions will add/remove windows, and sometimes I'd like to cycle tough the window layout/changes. In the following settings, `C-c <left>` to undo window layout changes, and `C-c <right>` to redo.

```
(winner-mode 1)
;; winner-undo -> C-c <left>
;; winner-redo -> C-c <right>
```

I'd like to use two frames, one for doing and logging, and other for reference/searching.

```
(defun yt/ref-frame ()
  (interactive)
  ;;   (frame-parameter (car (frame-list)) 'name)
  (if (eq 1 (length (frame-list)))
      (new-frame '((name . "**********REFERENCE**********")))
    nil))
(global-set-key (kbd "M-`") 'other-frame)
```

4.5 System Path/Keyboard

Solve the PATH issues for the software installed via Homebrew in OS

X. Uncomment the `setenv` for CYGWIN since I am not using Windows any

more.

```
(defun set-exec-path-from-shell-PATH ()
  (let ((path-from-shell
     (replace-regexp-in-string "[[:space:]\n]*$" ""
                               (shell-command-to-string "$SHELL -l -c 'echo $PATH'
↪"))))
    (setenv "PATH" path-from-shell)
    (setq exec-path (split-string path-from-shell path-separator))))
(when (equal system-type 'darwin) (set-exec-path-from-shell-PATH))
;; windows path convention
;; (setenv "CYGWIN" "nodosfilewarning")
```

Modify the Mac keyboard: unset the C-z just in case I run Emacs in terminal and C-z won't stop the program without asking.

```
;; modify mac keyboard
(cond ((eq system-type 'darwin)
       (setq mac-command-modifier 'meta)
       (fset 'insertPound "#")
       (global-set-key (kbd "M-3") 'insertPound)
       (global-unset-key (kbd "M-`"))
       (global-set-key (kbd "M-`") 'other-frame)
       (global-set-key (kbd "C-Z") nil)
       ))

(prefer-coding-system 'utf-8)
(when (display-graphic-p)
  (setq x-select-request-type '(UTF8_STRING COMPOUND_TEXT TEXT STRING)))
```

Open PDF files using external program.

[2016-06-20 Mon 21:43] `helm-find-files` has open with default tool functionality. This block is no longer needed.

```
;; (require 'openwith)
;; (openwith-mode t)
;; (if (string= system-type "darwin")
;;     (setq openwith-associations '(("\\.pdf\\'" "Skim" (file))))
;;   (setq openwith-associations '(("\\.pdf\\'" "evince" (file)))))
```

4.6 General Editing

There are a set of characters that are more likely to occur as a pair, for example, quote and brackets. *smartparens mode* allows me to define such set of pairing characters.

```
(smartparens-global-mode 1)
(sp-pair "(" ")" :wrap "C-(")
;; |foobar
;; hit C-(
;; becomes (|foobar)
(sp-pair "'" nil :actions :rem)
```

Modern display is widen. Like many of the Emacs users, I prefer to have the text wrapper inside a small region rather than have a stretch across the whole screen. It's easier to read in this way.

A well accepted rule is to set the width of lines to 80 characters, and force a logical line breaks. This funcitonality is called `auto-fill` in Emacs, and I can do the filling by call `fill-paragraph`.

```
(add-hook 'text-mode-hook 'turn-on-auto-fill) ;;
```

Just in case I need to reverse the auto-fill process.

[2016-06-20 Mon 21:47] Can't remember when was the last time I use unfill. This snippet is not long used.

```
(defun yt/unfill-paragraph ()
  (interactive)
  (let ((fill-column (point-max)))
    (fill-paragraph nil)))
(defun yt/unfill-region ()
  (interactive)
  (let ((fill-column (point-max)))
    (fill-region (region-beginning) (region-end) nil)))
```

4.7 Minibuffer history

Let Emacs remember what I've typed, so I don't need to tediously type the whole thing. Most of the time, I could just select using `helm`.

```
(setq savehist-file "~/git/.emacs.d/local/emacs-history")
(savehist-mode 1)
```

CHAPTER 5

GUI - Emacs Looks Cool

5.1 Fonts

I use the Adobe's *Source Code Pro* font, it is Monospaced font and claimed to be suitable for coding environments but I use it for all modes.

```
;; (set-frame-font "Source Code Pro-11" nil t)
;; (set-face-attribute 'default nil :height 100)
```

5.2 Minimalists GUI

I never click any buttons in the tool-bar, nor need the scroll-bar to tell me the cursor position the in the buffer, so I removed all of them to have minimalist GUI of Emacs.

Recently I found menu-bar is really useful, it shows commonly used functions for a particular mode. Occasionally I found something useful.

```
(tool-bar-mode -1)
; (menu-bar-mode -1)
(scroll-bar-mode -1)
```

5.3 Theme

I have been using *zenburn* theme for a while. It is a popular low contrast colour theme and easy on the eye. Occasionally I apply *tsdh-dark* theme on the top when I really need to focus on.

leuven theme is highly customised for org-mode and I like to use it when my eyes are tired of the *zenburn* theme.

```
;; (load-theme 'zenburn t)
;; (load-theme 'leuven t)
```

5.4 Mode Line

The mode line is at the bottom of every Emacs Window aside from MiniBuffer windows. It has most of the relevant information about the buffer, including Git status, Major mode, clock info, etc.

The smart-mode-line packages can make mode-line "smart and sexy". There are many options to tweak.

```
(setq sml/no-confirm-load-theme t)
(require 'smart-mode-line)
(setq powerline-arrow-shape 'curve)
(setq powerline-default-separator-dir '(right . left))
(setq sml/theme 'powerline)
(sml/setup)
```

There are too much information cluttered at the bottom. I disable the display of minor modes, there are just too many and almost all are irrelevant.

```
(rich-minority-mode 1)
(setf rm-blacklist "")
```

This will leave empty spaces which can be removed by

```
(setq sml/mode-width 0)
(setq sml/name-width 20)
```

Finally, show the current time in the mode-line.

```
(display-time-mode)
```

CHAPTER 6

Completion and Selection

6.1 Helm - Fuzzy Match

Helm and fuzzy match makes selection a lot easier. in

```
(require 'helm)
(require 'helm-config)

;; The default "C-x c" is quite close to "C-x C-c", which quits Emacs.
;; Changed to "C-c h". Note: We must set "C-c h" globally, because we
;; cannot change `helm-command-prefix-key' once `helm-config' is loaded.
(global-set-key (kbd "C-c h") 'helm-command-prefix)
(global-unset-key (kbd "C-x c"))

;; (define-key helm-map (kbd "<tab>") 'helm-execute-persistent-action) ; rebind tab␣
↪to run persistent action
;; (define-key helm-map (kbd "C-i") 'helm-execute-persistent-action) ; make TAB works␣
↪in terminal
;; (define-key helm-map (kbd "C-z")  'helm-select-action) ; list actions using C-z

(global-set-key (kbd "M-x") 'helm-M-x)
(global-set-key (kbd "C-x C-f") 'helm-find-files)

(setq helm-M-x-fuzzy-match t) ;; optional fuzzy matching for helm-M-x

(global-set-key (kbd "M-y") 'helm-show-kill-ring)
;;(global-set-key (kbd "C-x b") 'helm-mini)
(global-set-key (kbd "M-l") 'helm-mini)
(setq helm-buffers-fuzzy-matching t
      helm-recentf-fuzzy-match    t)
(global-set-key (kbd "C-c h o") 'helm-occur)
(global-set-key (kbd "C-h a") 'helm-apropos)
(setq helm-apropos-fuzzy-match t)
(setq helm-semantic-fuzzy-match t
```

(continues on next page)

(continued from previous page)

```
      helm-imenu-fuzzy-match    t)

(helm-autoresize-mode t)
(defun pl/helm-alive-p ()
  (if (boundp 'helm-alive-p)
      (symbol-value 'helm-alive-p)))
(add-to-list 'golden-ratio-inhibit-functions 'pl/helm-alive-p)
(helm-mode 1)

(defun yt/helm-copy-unmarked-to-buffer ()
  (interactive)
  (with-helm-buffer
    (helm-mark-all)
    (cl-loop for cand in (helm-marked-candidates)
         do (with-helm-current-buffer
              (insert cand "\n")))))
;; by default, Cc Ci copy marked to buffer.
(define-key helm-map (kbd "C-c C-i") 'helm-copy-unmmarked-to-buffer)

(setq helm-ff-guess-ffap-urls nil)
```

6.2 Multi-Cursor & Helm-swoop - Multiple Selection

When refactoring code, I need to rename a variable or function names, the normal way to do that is via searching and replacing. `multiple-cursors` provides function to select all the words/symbols that is highlighted and then modify all of them at the same time.

```
(require 'multiple-cursors)
(global-set-key (kbd "C->") 'mc/mark-next-like-this)
(global-set-key (kbd "C-<") 'mc/mark-previous-like-this)

;; (global-set-key (kbd "C-S-<right>") 'mc/mark-next-like-this)
;; (global-set-key (kbd "C-S-<left>") 'mc/mark-previous-like-this)
;; (global-set-key (kbd "C-S-c C-S-c") 'mc/edit-lines)
;; (global-set-key (kbd "C->") 'mc/mark-next-like-this)
;; (global-set-key (kbd "C-<") 'mc/mark-previous-like-this)
;; (global-set-key (kbd "C-c C-<") 'mc/mark-all-like-this)
;; (global-set-key (kbd "C-c C-<") 'mc/mark-all-like-this)
```

I usually use `multi-cursor` with `helm-swoop`, which allows me to search, and then narrow down all the occurrences in a temporary buffer, and then start to edit.

```
(require 'helm-swoop)
;; Change the keybinds to whatever you like :)
;; (global-set-key (kbd "M-i") 'helm-swoop)
;; (global-set-key (kbd "M-I") 'helm-swoop-back-to-last-point)
;; (global-set-key (kbd "C-c M-i") 'helm-multi-swoop)
;; (global-set-key (kbd "C-x M-i") 'helm-multi-swoop-all)
(global-set-key (kbd "<C-f1>") 'helm-swoop)
;; When doing isearch, hand the word over to helm-swoop
;; (define-key isearchp-mode-map (kbd "M-i") 'helm-swoop-from-isearch)
;; From helm-swoop to helm-multi-swoop-all
;; (define-key helm-swoop-map (kbd "M-i") 'helm-multi-swoop-all-from-helm-swoop)
;; When doing evil-search, hand the word over to helm-swoop
```

(continues on next page)

(continued from previous page)

```
;; (define-key evil-motion-state-map (kbd "M-i") 'helm-swoop-from-evil-search)
;; Save buffer when helm-multi-swoop-edit complete
(setq helm-multi-swoop-edit-save t)
;; If this value is t, split window inside the current window
(setq helm-swoop-split-with-multiple-windows nil)
;; Split direcion. 'split-window-vertically or 'split-window-horizontally
(setq helm-swoop-split-direction 'split-window-vertically)
;; If nil, you can slightly boost invoke speed in exchange for text color
(setq helm-swoop-speed-or-color nil)
;; ----------------------------------------------------------------------
```

6.3 ace-jump

Instead of moving into the place I want, ace-jump provides a way to jump directly to there places, just by pressing 4-5 keys. The places can be a character, line, or word. Personally I found it is really efficient to jump to a word when editing.

```
(global-set-key (kbd "C-c w") 'ace-jump-word-mode)
```

6.4 Expand-Region - Incremental Selection

expand-region provides smart way of sectioning, by expanding the scope one at a time. for example,

```
S = "A B C"
```

If the cursor in inside of the quote, I press `C-=`, everything inside of the quote is selected, press it again, the quotes are also selected, press it again, the whole line/region is selected. It saves a lot of keystrokes in highlighting the area.

It works well with *smartparens* mode, if I want to apply markup syntax around a word, I press `C-=` to select it, then insert quote or forward slash, the whole word will be warped inside of quote or forward flash.

```
(require 'expand-region)
(global-set-key (kbd "C-=") 'er/expand-region)
```

CHAPTER 7

File Management

7.1 Alternative to shell

For the file management tasks like rename and delete, I'd like to wrapper it as a Lisp function and call it directly in Emacs.

Rename the buffer-visiting file, and also rename the buffer. Similar to the *save as* idea but will remove the older file.

```
;; rename current buffer-visiting file
(defun yt/rename-current-buffer-file ()
  "Renames current buffer and file it is visiting."
  (interactive)
  (let ((name (buffer-name))
    (filename (buffer-file-name)))
    (if (not (and filename (file-exists-p filename)))
    (error "Buffer '%s' is not visiting a file!" name)
      (let ((new-name (read-file-name "New name: " filename)))
    (if (get-buffer new-name)
        (error "A buffer named '%s' already exists!" new-name)
      (rename-file filename new-name 1)
      (rename-buffer new-name)
      (set-visited-file-name new-name)
      (set-buffer-modified-p nil)
      (message "File '%s' successfully renamed to '%s'"
               name (file-name-nondirectory new-name)))))))
```

Another useful Lisp function is to copy the file path to clipboard for cross reference.

```
;; full path of current buffer
(defun yt/copy-full-path-to-kill-ring ()
  "copy buffer's full path to kill ring"
  (interactive)
  (when buffer-file-name
    (let* ((file-truename buffer-file-name))
```

(continues on next page)

(continued from previous page)

```
      ;;(rel-name (file-relative-name file-truename "~/")))  ; BUG: if filename is␣
↪not relative to home directory.
      ;; (kill-new (concat "~/" rel-name)))))
      (kill-new file-truename))))
```

Open a file as a root user in Emacs, very handy.

```
(defun yt/sudo-find-file (file-name)
  "Like find file, but opens the file as root."
  (interactive "FSudo Find File: ")
  (let ((tramp-file-name (concat "/sudo::" (expand-file-name file-name))))
    (find-file tramp-file-name)))
```

Find out the last modified date for current buffer, I need this often when updating a blog post or documents.

```
(defun yt/last-updated-date ()
  "return modification time of current file-visitng buffer"
  (interactive)
  (let* ((mtime (visited-file-modtime)))
    (unless (integerp mtime)
      (concat "/Last UPdated/: "
          (format-time-string "%d %b %Y" mtime)))))
```

Remove current buffer-visiting file, and kill the buffer. I use this function often in testing and trying out.

```
(defun yt/delete-this-buffer-and-file ()
  "Removes file connected to current buffer and kills buffer."
  (interactive)
  (let ((filename (buffer-file-name))
    (buffer (current-buffer))
    (name (buffer-name)))
    (if (not (and filename (file-exists-p filename)))
    (error "Buffer '%s' is not visiting a file!" name)
      (when (yes-or-no-p "Are you sure you want to remove this file? ")
    (delete-file filename)
    (kill-buffer buffer)
    (message "File '%s' successfully removed" filename)))))
```

It is a good practise to group all the file management related commands together using hydra.

```
(defhydra hydra-file-management (:color red
                                        :hint nil)
  "
_o_pen file
_O_pen file as Sudo user
copy file _P_ath to kill ring
_r_ename buffer-visiting file
_d_elete buffer-visiting file
open with _e_xternal application
_g_it sync"
  ("o" find-file)
  ("O" yt/sudo-find-file)
  ("P" yt/copy-full-path-to-kill-ring)
  ("r" yt/rename-current-buffer-file)
  ("c" yt/copy-file-to)
  ("d" yt/delete-this-buffer-and-file)
```

(continues on next page)

(continued from previous page)

```
  ("e" prelude-open-with)
  ("g" yt/git-up))
(global-set-key [f3] 'hydra-file-management/body)
```

Open the file manager at the default directory.

```
;; http://ergoemacs.org/emacs/emacs_dired_open_file_in_ext_apps.html
(defun yt/open-file-manager ()
  "Show current file in desktop (OS's file manager)."
  (interactive)
  (cond
   ((string-equal system-type "windows-nt")
    (w32-shell-execute "explore" (replace-regexp-in-string "/" "\\" default-directory␣
↪t t)))
   ((string-equal system-type "darwin") (shell-command "open ."))
   ((string-equal system-type "gnu/linux")
    (let ((process-connection-type nil)) (start-process "" nil "xdg-open" "."))
    ;; (shell-command "xdg-open .") ;; 2013-02-10 this sometimes froze emacs till the␣
↪folder is closed.  with nautilus
    )))
```

7.2 Projectile - Directory Access

Projectile is an powerful Emacs package but I only use *projectile* to jump between different git folders, so there isn't much configuration except using `helm` for selection.

```
(require 'projectile)
(projectile-mode +1)
(define-key projectile-mode-map (kbd "C-c p") 'projectile-command-map)
(helm-projectile-on)
(require 'helm-projectile)
(projectile-global-mode)
(setq projectile-enable-caching t)
(setq projectile-switch-project-action 'projectile-dired)
(setq projectile-remember-window-configs t )
(setq projectile-completion-system 'helm)
(setq projectile-switch-project-action 'helm-projectile)
(setq projectile-project-root-files-bottom-up '(".git" ".projectile")) ;; .projectile␣
↪comes first
```

There are many things work out of box. For example, use `C-p p` to choose which project to jump to, but I can type `M-g` to invoke Magit or `M-e` to invoke Eshell for that project.

7.3 Remote (SSH)

I can work on the remote files in Emacs via ssh or tramp, both are build-in packages.

```
(require 'tramp)
(require 'ssh)
```

I'd like catch the password so that I don't need to type it every time to open a file.

```
(setq password-cache-expiry nil)
```

I mainly run R on a remote machine. Sometimes I want to copy the charts I created to local to include them in my report. This workfow is suspended because it fails when the file size is large.

```
;; (defun yt/sync-local-remote ()
;;   (interactive)
;;   "copy all files in remote:~/LR_share to local:~/LR_share,
;; does not support the ther way"
;;   (find-file "/ssh:remote_host:/remote_directory")
;;   ;; (mark-whole-buffer)
;;   (dired-mark-subdir-files)
;;   ;; (find-file "~/LR_share")
;;   ;; (setq-local dirqed-dwim-target t)
;;   (dired-do-copy))
```

7.4 Git Sync

I use git and Github a lot, and usually in `shell-mode`, but I just can't remember all the commands. Magit provides an interface to Git, and it is really pleasant to use. So I don't need to remmeber all the commands, also it comes with excellent manual and cheatsheet.

```
(require 'magit)
(setq magit-last-seen-setup-instructions "1.4.0")
(setq magit-auto-revert-mode nil)
(global-set-key (kbd "<f9> g") 'magit-status)
(global-auto-revert-mode t)
```

Occasionally my office machine goes down because I run R with big data, and it consumes all the memory. If that happens, I potentially lose the newsiest version of scripts, which is bit annoy. The following snippets will save all buffers in every hours.

```
(defun yt/save-all-buffers ()
  "save all files-visiting buffers without user confirmation"
  (interactive)
  (save-some-buffers t nil)
  (message "save all buffers... done"))
(run-at-time "05:59" 3600 'yt/save-all-buffers)
```

Sometimes I have to leave at the last minutes, then what I do is call a functions that commits and upload to the repo so that I can continue work at home.

The `yt/git-up` function will do

1. pull from the remote repo, and make sure the local repo is always up-to-date.
2. add everything and commit with a timesamp.
3. push local changes to the remote repo.

Here is the snippets.

```
(defun yt/git-backup ()
  (let ((git-sh-scripts "
echo Start to Sync: $(date)

```

(continues on next page)

(continued from previous page)

```
REPOS=\"org\"
for REPO in $REPOS
do
    echo
    echo \"Repository: $REPO\"
    cd ~/git/$REPO
    # update
    git pull
    # Remove deleted files
    git ls-files --deleted -z | xargs -0 git rm >/dev/null 2>&1
    # Add new files
    git add -A . >/dev/null 2>&1
    git commit -m \"$(date)\"
    git push origin master
done

echo Finished Sync: $(date)
"))
    (async-shell-command git-sh-scripts))
  (message "all git sync... done"))

(defun yt/git-up ()
  (interactive)
  (yt/save-all-buffers)
  (yt/git-backup))
```

Few times I did some important work over the weenend, but once I arrived office I realised I forgot uploading, These situations are quick frustrating. The following snippets will start to uploads once every three hours on my MacbookPro, but I don't use it anymore, since I can get most of my work done in the office.

Note this workflow is suspended for it's unsafe.

```
;; (cond ((eq system-type 'darwin)
;;        (run-at-time "05:59" 10800 'yt/git-up)))
```

7.5 Testing Buffers

scratch buffer is usually used for testing Emacs lisp functions. I also need temporary buffers for testing R code and org-mode. In the following settings, I can use `F9-f` to select temporal buffers.

```
(defvar yt/temp-dir "~/.tmp"
  "temporay folders")

(defun yt/open-tmp-R ()
  (interactive)
  (find-file (expand-file-name "tmp.R" yt/temp-dir)))
(defun yt/open-tmp-el ()
  (interactive)
  (find-file (expand-file-name "tmp.el" yt/temp-dir)))
(defun yt/open-tmp-org ()
  (interactive)
  (find-file (expand-file-name "tmp.org" yt/temp-dir)))
(global-set-key (kbd "<f9> f r") 'yt/open-tmp-R)
(global-set-key (kbd "<f9> f e") 'yt/open-tmp-el)
(global-set-key (kbd "<f9> f o") 'yt/open-tmp-org)
```

7.6 Frequently visiting buffers

```
(defun yt/org-find-file (filepath)
  (interactive)
  (find-file (expand-file-name filepath "~/git/org") nil))

(defhydra hydra/open-common-files (:color blue)
  "Open file:
      "
  ("R" (find-file "~/git/career/Profession/R.org") "R.org")
  ("p" (find-file "~/git/career/Profession/Python.org") "Python.org")
  ("E" (find-file "~/git/career/Profession/Emacs.org") "Emacs.org")
  ("l" (find-file "~/git/org/life/life.org") "life.org")
  ("i" (find-file "~/git/.emacs.d/init.org" t) "init.org")
  ("e" (find-file "~/.emacs" t) ".emacs")
  ("d" (yt/org-find-file "dournal/diary.org") "diary.org")
  ("r" (yt/org-find-file "life/review.org") "review.org")
  ("f" (yt/org-find-file "finance/ledger_transaction_2019.org") "ledger.org")
  )
(global-set-key (kbd "<f6>") 'hydra/open-common-files/body)
```

CHAPTER 8

ESS - Emacs Speaks Statistics

As Statistician, coding in R and writing report is what I do most of the day. I have been though a long way of searching the perfect editor for me, tried Rstudio, SublimeText, TextMate and settled down happily with ESS/Emacs, for both coding and writing.

There three features that have me made the decision:

1) Auto Formatting

   Scientists has reputation of being bad programmers, who wrote the code that is unreadable and therefore incomprehensible to others. I have intention to become top level programmer and followed a style guide strictly. It means I have to spent sometime in adding and removing space in the code.

   To my surprise, Emacs will do it for me automatically, just by hitting the TAB and it also indent smartly, which make me conformable to write long function call and split it into multiple lines. Here's an example. Also if I miss placed a ')' or ']' the formatting will become strange and it reminders me to check.

```
rainfall.subset <- data.table(rainfall.london,
                      rainfall.pairs,
                      rainfall.dublin)
```

2) Search Command History

   I frequently search the command history. Imaging I was produce a plot and I realised there was something miss in the data, so I go back and fix the data first, then run the ggplot command again, I press Up/Down bottom many times, or just search once/two times. `M-x ggplot(` will gives me the most recent command I typed containing the keyword *ggplot(*, then I press `RET` to select the command, which might be `ggplot(gg.df, aes(lon, lat, col = city)) + geom_line() + .....`. If it is not I want, I press `C-r` again to choose the second most recent one and repeat until I find right one.

3) Literate Programming

   I am an supporter of literate statistical analysis and believe we should put code, results and discoveries together in developing models. Rstudio provides an easy to use tool for this purpose, but it does not support different R sessions, so if I need to generate a report, I have to re-run all the code from beginning, which isn't particle for me with volumes data because it will take quit long.

ESS and org-mode works really well via Babel, which is more friendly to use. I can choose to run only part of the code and have the output being inserted automatically, no need to copy/paste. Also, I can choose where to execute the code, on my local machine or the remote server, or both at the same time.

These are only the surface of ESS and there are lot more useful features like spell checking for comments and documentation templates, that makes me productive and I would recommend anyone use R to learn ESS/Emacs. The following is my current setting.

```
;; Adapted with one minor change from Felipe Salazar at
;; http://www.emacswiki.org/emacs/EmacsSpeaksStatistics
(require 'ess-site)
(setq ess-ask-for-ess-directory nil) ;; start R on default folder
(setq ess-local-process-name "R")
(setq ansi-color-for-comint-mode 'filter) ;;
;; (setq comint-scroll-to-bottom-on-input t)
;; (setq comint-scroll-to-bottom-on-output nil)
;; (setq comint-move-point-for-output nil)
(setq ess-eval-visibly-p 'nowait) ;; no waiting while ess evalating
(defun my-ess-start-R ()
  (interactive)
  (if (not (member "*R*" (mapcar (function buffer-name) (buffer-list))))
      (progn
    (delete-other-windows)
    (setq w1 (selected-window))
    (setq w1name (buffer-name))
    (setq w2 (split-window w1))
    (R)
    (set-window-buffer w2 "*R*")
    (set-window-buffer w1 w1name))))
(defun my-ess-eval ()
  (interactive)
  (my-ess-start-R)
  (if (and transient-mark-mode mark-active)
      (call-interactively 'ess-eval-region)
    (call-interactively 'ess-eval-line-and-step)))
(add-hook 'ess-mode-hook
      '(lambda()
         (local-set-key [(shift return)] 'my-ess-eval)))
(add-hook 'ess-mode-hook
      (lambda ()
        (flyspell-prog-mode)
        (run-hooks 'prog-mode-hook)
        ))
(add-hook 'ess-R-post-run-hook (lambda () (smartparens-mode 1)))

;; REF: http://stackoverflow.com/questions/2901198/useful-keyboard-shortcuts-and-tips-
↪for-ess-r
;; Control and up/down arrow keys to search history with matching what you've already␣
↪typed:
(define-key comint-mode-map [C-up] 'comint-previous-matching-input-from-input)
(define-key comint-mode-map [C-down] 'comint-next-matching-input-from-input)
(setq ess-history-file "~/.Rhisotry")
(setq ess-indent-with-fancy-comments nil)


(define-key ess-r-mode-map "_" #'ess-insert-assign)
(define-key inferior-ess-r-mode-map "_" #'ess-insert-assign)
```

8.1 Syntax highlight

In Emacs, syntax highlighting is known as font-locking. You can customize the amount of syntax highlighting that you want to see. At the top of the Emacs window, click on the ESS menu and select "Font Lock". This will display a menu of buttons corresponding to language elements that you can syntax highlight.

```
(setq ess-R-font-lock-keywords
    '((ess-R-fl-keyword:modifiers . t)
     (ess-R-fl-keyword:fun-defs . t)
     (ess-R-fl-keyword:keywords . t)
     (ess-R-fl-keyword:assign-ops)
     (ess-R-fl-keyword:constants . t)
     (ess-fl-keyword:fun-calls . t)
     (ess-fl-keyword:numbers)
     (ess-fl-keyword:operators)
     (ess-fl-keyword:delimiters)
     (ess-fl-keyword:=)
     (ess-R-fl-keyword:F&T)
     (ess-R-fl-keyword:%op%)))
```

use pretty mode

```
;; (add-hook 'ess-mode-hook 'turn-on-pretty-mode)
```

8.2 Programming Mode

After 2014, Emacs comes a prog-mode, for programming langauge. it is generic mode, just like text-mode, that sits underneth all the programming language, either R, phython, C++ etc. The good thinkg to have this concept is that we can define few things that will apply to all these mode, when we write scripts.

One thing I find particulaar usefull and necessary is to highliht characters in comments that has particullar meaning, like TODO, FIXME or other. which can be particular handy in code reivew, I can navite and jump between the code quickly.

```
;; highlights FIXME: TODO: and BUG: in prog-mode
;; (add-hook 'prog-mode-hook
;;           (lambda ()
;;             (font-lock-add-keywords nil
;;                                     '(("\\<\\(YT\\|FIXME\\|TODO\\|BUG\\):" 1 font-
↪lock-warning-face t)))))
```

we usually have long scripts, and in Subimetext, one cold folder and unfolder a function. in Emacs, this feature could be extended to furture, by define folder-characters. at this statge, I tented to used the deafault, I.e. folder functions only. in the folliwng setting, I could press `F3` to folder/unfolder a function, `C-F3` or `S-F3` to folder/unfolder all functions.

One potentially solution is to use `outshine` package, to show/hide the whole section.

```
;; (add-hook 'prog-mode-hook 'hs-minor-mode)
;; (defalias 'fold-toggle 'hs-toggle-hiding)
;; (global-set-key (kbd "<f4>") 'hs-toggle-hiding)
;; (global-set-key (kbd "S-<f4>") 'hs-show-all) ;; S->show
;; (global-set-key (kbd "C-<f4>") 'hs-hide-all)
;; ;;   hs-hide-block                      C-c @ C-h
```

(continues on next page)

(continued from previous page)

```
;; ;;   hs-show-block                       C-c @ C-s
;; ;;   hs-hide-all                         C-c @ C-M-h
;; ;;   hs-show-all                         C-c @ C-M-s
;; ;;   hs-hide-level                       C-c @ C-l
;; ;;   hs-toggle-hiding
;; ;;   hs-mouse-toggle-hiding              [(shift mouse-2)]
;; ;;   hs-hide-initial-comment-block
(global-set-key (kbd "C-d") 'comment-region) ;; overwite delete-char
(global-set-key (kbd "C-S-d") 'uncomment-region)

(defhydra hydra-fold (:pre (hs-minor-mode 1))
  "fold"
  ("t" hs-toggle-hiding "toggle")
  ("s" hs-show-all "hide-all")
  ("h" hs-hide-all "show-all")
  ("q" nil "quit"))
(global-set-key (kbd "<f4>") 'hydra-fold/body)
```

use subword-mode then ThisPhase has two word, and I can use `C-DEL` it will remove the Phase and left This. Very useful in CamerCase.

```
(subword-mode 1)
```

highlights the text that are longer than 80 columns rule.

```
(require 'whitespace)
(setq whitespace-line-column 120) ;; limit line length
(setq whitespace-style '(face lines-tail))
(add-hook 'prog-mode-hook 'whitespace-mode)
```

Rainbow-delimiters. constantly have problem with package, and tired of fixing it, so I turned it off at this stage.

```
(require 'rainbow-delimiters)
(add-hook 'prog-mode-hook 'rainbow-delimiters-mode)
(show-paren-mode t) ;for Emacs
```

use f8 to remove the R process buffer.

```
(defun yt/prog-previous-output-region ()
  "return start/end points of previous output region"
  (save-excursion
    (beginning-of-line)
    (setq sp (point))
    (comint-previous-prompt 1)
    (next-line)
    (beginning-of-line)
    (setq ep (point))
    (cons sp ep)))
(defun yt/prog-kill-output-backwards ()
  (interactive)
  (save-excursion
    (let ((reg (yt/prog-previous-output-region)))
      (delete-region (car reg) (cdr reg))
      (goto-char (cdr reg))
      (insert "*** output flushed ***\n"))))
;; (global-set-key (kbd "<f8>") 'yt/prog-kill-output-backwards)
```

8.3 Documentation

```
;; edit roxy template
;; ess-roxy-update-entry
(setq ess-roxy-template-alist '(("description" . " content for description")
                                ("details" . "content for details")
                                ("title" . "")
                                ("param" . "")
                                ("return" . "")
                                ("export" . "")
                                ("author" . "Yi Tang")))
```

8.4 R Style Check - Flycheck

https://github.com/jimhester/lintr the default R-style is not meet my with current R project style, has to turn it off.

```
(require 'flycheck)
;; '(flycheck-lintr-caching nil) ;; need to customised it inside of Emacs
;; (add-hook 'ess-mode-hook
;;           (lambda () (flycheck-mode t)))
```

8.5 Scripts editing

8.6 R programming

clean up the messy R scripts buffer. it will

1. remove comments lines start with '## '
2. remove blank lines,
3. add one blank lines between sections, which defined by '#### '.

```
(defun yt/clean-R ()
  (interactive)
  (when (string= major-mode "ess-mode")
    (progn
      (goto-char (point-min))
      (flush-lines "^\\(\\|[[:space:]]+\\)[#]\\{1,3\\} ") ;; remove lines with only␣
↪commenst and start with #, ##, or ###, but not #### for it's the section heading.
      (flush-lines "^\\(\\|[[:space:]]+\\)$") ;; blank lines
      (replace-regexp "#### " "\n#### ") ;; add blank lines between sections.
      (while (search-forward-regexp "##[^']" nil t) ;; remove inline comments start␣
↪with ##
    (kill-region (- (point) 3) (line-end-position)))
    (save-buffer))))
```

apply the clean scripts to the tangled file. also, preappend the date and my name on the tangled file.

```
;; add author info
(defun yt/ess-author-date ()
  (interactive)
```

(continues on next page)

(continued from previous page)

```
  (when (string= major-mode "ess-mode")
    (goto-char (point-min))
    (insert "##' @author: Yi Tang\n")
    (insert "##' @date: ")
    (insert (format-time-string "%F %T"))
    (insert "\n\n")
    (save-buffer)))
(add-hook 'org-babel-post-tangle-hook 'yt/ess-author-date)
(add-hook 'org-babel-post-tangle-hook 'yt/clean-R)
```

CHAPTER 9

Writing in Emacs

9.1 Spell and Grammar

Spell checking and correcting are essential in writing. Emacs need third party program do this. There are a couple of programs and I use `aspell`. It is part of GNU and can be easily installed in OS X and Ubuntu. The following snippet tells Emacs where `aspell` is installed and use British dictionary.

```
(if (eq system-type 'darwin)
    (setq ispell-program-name "/usr/local/bin/aspell")
  (setq ispell-program-name "/usr/bin/aspell"))
(setq ispell-dictionary "british"
      ispell-extra-args '() ;; TeX mode "-t"
      ispell-silently-savep t)
```

I have a personal spelling dictionary, most are abbreviations and jargon. I can tell aspell that they are not misspellings.

```
(setq ispell-personal-dictionary "~/git/.emacs.d/local/ispell-dict") ;; add personal␣
↪dictionary
```

```
(add-to-list 'ispell-skip-region-alist '(":\\(PROPERTIES\\|LOGBOOK\\):" . ":END:"))
(add-to-list 'ispell-skip-region-alist '("#\\+BEGIN_SRC" . "#\\+END_SRC"))
```

`Flyspell` depends on ispell mode and enables on-the-fly spell checking/correcting. I enable the flyspell mode for text-mode and org-mode.

By default, I use C-, to move the cursor to the next misspelled word, and `flycheck` will provide a list of candidates for auto-correlection. I press `C-.` select the first one, and press it again to select the next one.

```
(require 'flyspell)
(add-hook 'text-mode-hook 'flyspell-mode)
(add-hook 'org-mode-hook 'flyspell-mode)
(define-key flyspell-mode-map (kbd "C-.") 'helm-flyspell-correct)
```

I need an grammar check to let me know that

```
Have you do ...
```

is wrong, and also tell me to change *do* to *done*, and also why. `langtool` can do be the job, but currently I don't understand how to get it works, so I am not using it anymore.

```
;; check grammar
(require 'langtool)
(setq langtool-language-tool-jar "~/java/LanguageTool-2.8/languagetool-commandline.jar
↪")
(setq langtool-mother-tongue "en")
```

9.2 Abbreviation

I have been writing in Emacs/org-mode a lot, have been really tired of capitalise i to I, so I use abbrevitation table.

name	expand	Category
i	I	write
amax	annual maximum	stat
gmap	google map	website
mailme	yi.tang.uk@me.com	aboutme
twitterme	@yi_tang_uk	aboutme
eqt	equivalent to	english
iif	if and only if	maths
wrt	with respect to	English
st	such that	English
d/n	distribution	Stats
obs	observation	stats
obss	observations	stats

```
(defun my-text-abbrev-expand-p ()
  "Return t if the abbrev is in a text context, which is: in
   comments and strings only when in a prog-mode derived-mode or
   src block in org-mode, and anywhere else."
  (if (or (derived-mode-p 'prog-mode)
      (and (eq major-mode 'org-mode)
           (org-in-src-block-p 'inside)))
      (nth 8 (syntax-ppss))
    t))

(define-abbrev-table 'my-text-abbrev-table ()
  "Abbrev table for text-only abbrevs. Expands only in comments and strings."
  :enable-function #'my-text-abbrev-expand-p)

(dolist (table (list text-mode-abbrev-table
                 prog-mode-abbrev-table))
  (abbrev-table-put table
                :parents (list my-text-abbrev-table)))

(defun my-text-abbrev-table-init (abbrevs-org-list)
  "Parse 'name: expansion' pairs from an org list and insert into abbrev table."
  (message "Creating text-abbrev table...")
  (dolist (abbrev abbrevs-org-list)
```

(continues on next page)

(continued from previous page)

```
    (let ((name (nth 0 abbrev))
      (expansion (nth 1 abbrev)))
      ;; (print (cons name expansion))
      (define-abbrev my-text-abbrev-table name expansion nil :system t))))
(my-text-abbrev-table-init my-text-abbrevs)
```

9.3 Style

English is my second language, and I am trying to avoid certain guarding term in writing. The following snipts I get it from Sachua will highlight the word like *shuold* or *I think*, which reminds to confirm with what I am not sure about, and have more confidence in what I am saying.

```
(require 'artbollocks-mode)
(add-hook 'text-mode-hook 'artbollocks-mode)
(setq artbollocks-weasel-words-regex
      (concat "\\b" (regexp-opt
                '("should"
                  "just"
                  "sort of"
                  "a lot"
                  "probably"
                  "maybe"
                  "perhaps"
                  "I think"
                  "really"
                  "nice") t) "\\b"))
```

add synosaurus

```
;; [2015-02-12 Thu 21:14]
;; https://github.com/rootzlevel/synosaurus
;; synosaurus-lookup
;; synosaurus-choose-and-replace
;; brew install wordnet
(require 'synosaurus)
(setq synosaurus-choose-method "popup")

;; synosaurus-lookup C-c s l
;; synosaurus-choose-and-replace C-c s r
(setq synosaurus-backend 'synosaurus-backend-wordnet)
(setq synosaurus-choose-method 'popup)
```

CHAPTER 10

Org mode

I started to learn Emacs by reading Bernt Hansen's Org Mode - Organize Your Life In Plain Text! .My settings based on Bernt's

associate org-mode with file with .org, .org_archive, and .txt extension.

```
(add-to-list 'auto-mode-alist '("\\.\\(org\\|org_archive\\|txt\\)$" . org-mode))
```

10.1 org-todos

First, define the TODO keywords.

```
(setq org-todo-keywords
      (quote ((sequence "TODO(t)" "NEXT(n)" "SOMEDAY" "ORG(o@/!)" "|" "DONE(d)")
          (sequence "WAITING(w@/!)" "HOLD(h@/!)" "RUNNING(r)" "|" "CANCELLED(c@/!)"
↪"MEETING"))))
```

Then highlight the keywords using different colours.

```
(setq org-todo-keyword-faces
      (quote (("TODO" :foreground "red" :weight bold)
          ("NEXT" :foreground "red" :weight bold)
          ("ORG" :foreground "blue" :weight bold)
          ("DONE" :foreground "forest green" :weight bold)
          ("WAITING" :foreground "orange" :weight bold)
          ("RUNNING" :foreground "orange" :weight bold)
          ("HOLD" :foreground "magenta" :weight bold)
          ("CANCELLED" :foreground "forest green" :weight bold)
          ("MEETING" :foreground "forest green" :weight bold))))
```

Define an event when a TODO status changes, for example, if changed to HOLD, add HOLD tag and remove WAITING tag. If changed to DONE, remove both HOLD and WAITING tags.

```
(setq org-todo-state-tags-triggers
      (quote (("CANCELLED" ("CANCELLED" . t))
          ("WAITING" ("WAITING" . t))
          ("HOLD" ("WAITING") ("HOLD" . t))
          (done ("WAITING") ("HOLD"))
          ("TODO" ("WAITING") ("CANCELLED") ("HOLD"))
          ("NEXT" ("WAITING") ("CANCELLED") ("HOLD"))
          ("DONE" ("WAITING") ("CANCELLED") ("HOLD")))))
```

Especially, when a task is marked as DONE, a timestamp is added to the LOGBOOK drawer.

```
;; (setq org-log-done (quote time))
;; (setq org-log-into-drawer t)
;; (setq org-log-state-notes-insert-after-drawers nil)
```

Add a cross line for the headline with DONE status. Note currently it is disabled before of the performance issues in OS X.

```
(defun my/modify-org-done-face ()
  (setq org-fontify-done-headline t)
  (set-face-attribute 'org-done nil :strike-through t)
  (set-face-attribute 'org-headline-done nil
                      :strike-through t
                      :foreground "light gray"))
(add-hook 'org-mode-hook 'my/modify-org-done-face)
;; (setq org-fontify-done-headline t)
;; (set-face-attribute 'org-done nil :strike-through t)
;; (set-face-attribute 'org-headline-done nil :strike-through t)
```

10.2 org-capture

Use `C-c c` anywhere to quickly create a org headline and save it to a default place.

```
(global-set-key (kbd "C-c c") 'org-capture)
```

The capture mode templates.

```
(setq org-capture-templates
      (quote (("t" "todo" entry (file "~/git/org/life//refile.org")
           "* TODO %?\n\n" :clock-in t :clock-resume t) ;; TODO: %? %U %a, what does␣
↪these means??? %: %c
          ("o" "org" entry (file "~/git/org/life//refile.org")
           "* TODO %?General Org\n%U\n" :clock-in t :clock-resume t) ;; TODO: %? %U
↪%a, what does these means??? %: %c
          ;; ("r" "respond" entry (file "~/git/org/life//refile.org")
          ;;  "* To %? about :RESPONSE:  \nSCHEDULED: %t\n%U\n%a" :clock-in t :clock-
↪resume t)
          ("r" "read" entry (file "~/git/org/life//refile.org")
           "* TODO %? :READ:\n%U\n" :clock-in t :clock-resume t)
          ("n" "note" entry (file "~/git/org/life//refile.org")
           "* %? :NOTE:\n%U\n" :clock-in t :clock-resume t)

          ("h" "Habit" entry (file "~/git/org/habits.org")
           "* NEXT %?\nSCHEDULED: %(format-time-string \"<%Y-%m-%d .+1d/3d>\
↪")\n:PROPERTIES:\n:STYLE: habit\n:REPEAT_TO_STATE: NEXT\n:END:\n%U\n")
```

(continues on next page)

(continued from previous page)

```
          ("v" "Vocabulary" entry (file "~/git/org/vocabulary.org")
           "* %? :VOCA:\n%U" :clock-in t :clock-resume t)

          ("j" "Journalsing")
          ("jd" "diary" entry (file+datetree "~/git/org/diary/2020.org")
           "* %?\n%U\n" :clock-in t :clock-resume t)
          ("jk" "Kaggle Competition" entry (file+datetree "~/git/org/diary/Kaggle.org
↪")
           "* %?\n%U\n" :clock-in t :clock-resume t)
          ("js" "Statistician" entry (file+datetree "~/git/org/diary/Statistics.org")
           "* %?\n%U\n" :clock-in t :clock-resume t)
          ("jo" "Office" entry (file+datetree "~/git/org/diary/2020.org")
           "* %?:office:\n%U\n" :clock-in t :clock-resume t)
          ("jf" "Finance" entry (file+datetree "~/git/org/diary/2020.org")
           "* %?:finance:\n%U\t\n" :clock-in t :clock-resume t)
          ("jc" "Career" entry (file+datetree "~/git/org/diary/Career.org")
           "* %?\n%U\n" :clock-in t :clock-resume t)
          )))

;; ledger entries

(push '("l" "Ledger Journal" plain (file "~/git/data_finance/ledger/refile.ledger")
       "%(org-read-date) * %(yt/helm-ledger-payee)
    %(yt/helm-ledger-account)    £ %(yt/helm-ledger-amount)
    %(yt/helm-ledger-account)")
     org-capture-templates
     )
```

Speed up the process by using cache.

```
(setq org-refile-use-cache t)
```

10.3 org-refile

Set the refile targets, they are all level 1 2 3 in current buffer and all the files in *org-agenda-files*.

```
(setq org-refile-targets
      '((nil :maxlevel . 3)
    (org-agenda-files :maxlevel . 3)))
(setq org-outline-path-complete-in-steps nil)
```

but exclude DONE state tasks from refile targets

```
(defun bh/verify-refile-target ()
  "Exclude todo keywords with a done state from refile targets"
  (not (member (nth 2 (org-heading-components)) org-done-keywords)))
(setq org-refile-target-verify-function 'bh/verify-refile-target)
```

Provide refile targets as paths. So a level 3 headline will be available as level1/level2/level3.

```
(setq org-refile-use-outline-path t)
```

Use helm for matching the target path. a low easier.

```
(setq org-completion-handler 'helm)
```

10.4 org-clock

Save the running clock and all clock history when exiting Emacs, load it on startup

```
(setq org-clock-persist t)
```

Resume clocking task when emacs is restarted, and if continue to count on this task.

```
(org-clock-persistence-insinuate)
(setq org-clock-in-resume t)

;; Do not prompt to resume an active clock
;; (setq org-clock-persist-query-resume nil)
```

```
;; Save clock data and state changes and notes in the LOGBOOK drawer
(setq org-clock-into-drawer t)
;; Sometimes I change tasks I'm clocking quickly - this removes clocked tasks with␣
↪0:00 duration
(setq org-clock-out-remove-zero-time-clocks t)
;; Clock out when moving task to a done state
(setq org-clock-out-when-done t)

;; Enable auto clock resolution for finding open clocks
(setq org-clock-auto-clock-resolution (quote when-no-clock-is-running))
;; Include current clocking task in clock reports
(setq org-clock-report-include-clocking-task t)
```

highlight the clocking info in mode line.

```
(set-face-attribute 'org-mode-line-clock nil
                    :weight 'bold :box '(:line-width 1 :color "#FFBB00") :foreground
↪"white" :background "#FF4040")
```

List recently clocked headline and clock in.

```
;; Show lot of clocking history so it's easy to pick items off the C-F11 list
(setq org-clock-history-length 23)
;; http://stackoverflow.com/questions/6156286/emacs-lisp-call-function-with-prefix-
↪argument-programmatically
(defun yt/org-clock-in-select ()
  (interactive)
  (setq current-prefix-arg '(4)) ;; C-u,
  (call-interactively 'org-clock-in))
(global-set-key (kbd "S-<f11>") 'yt/org-clock-in-select)
(global-set-key (kbd "<f11>") 'org-clock-jump-to-current-clock)
```

When clock in to a TODO headline, turn the keywords into NEXT.

```
;; Change tasks to NEXT when clocking in
(setq org-clock-in-switch-to-state 'bh/clock-in-to-next)
(defun bh/clock-in-to-next (kw)
  "Switch a task from TODO to NEXT when clocking in.
```

(continues on next page)

(continued from previous page)

```
Skips capture tasks"
  (when (not (and (boundp 'org-capture-mode) org-capture-mode))
    (if (member (org-get-todo-state) (list "TODO"))
    "NEXT")))
```

punch-in into a default org-mode headline.

```
(defun yt/punch-in ()
  (interactive)
    (org-with-point-at (org-id-find "1b586ec1-fa8a-4bd1-a44c-faf3aa2adf51" 'marker)
    (org-clock-in)
     ))
(global-set-key (kbd "<f9> I") 'yt/punch-in)
```

use hydra to define a function that use most frequently

```
;; https://github.com/abo-abo/hydra/wiki/Org-clock
(defhydra hydra-org-clock (:color blue :hint nil)
  "
Clock        In/out^     ^Edit^   ^Summary      (_?_)
-----------------------------------------------------
        _i_n         _e_dit   _g_oto entry
_h_istory   _c_ontinue   _q_uit   _d_isplay
_j_ump      _o_ut        ^ ^      _r_eport
      "
  ("i" org-clock-in)
  ("o" org-clock-out)
  ("c" org-clock-in-last)
  ("e" org-clock-modify-effort-estimate)
  ("q" org-clock-cancel)
  ("g" org-clock-goto)
  ("d" org-clock-display)
  ("r" org-clock-report)
  ("j" org-clock-jump-to-current-clock)
  ("h" yt/org-clock-in-select)
  ("?" (org-info "Clocking commands")))

(global-set-key (kbd "<f11>") 'hydra-org-clock/body)
```

remove empty clock entrys at checkout

```
(add-hook 'org-clock-out-hook 'org-clock-remove-empty-clock-drawer 'append)
```

10.5 org-tags

```
(setq org-tag-alist (quote ((:startgroup)
                            ("@office" . ?O)
                            ("@home" . ?H)
                            (:endgroup)
                            ("WAITING" . ?w)
                            ("HOLD" . ?h)
                            ("PERSONAL" . ?P)
                            ("WORK" . ?W)
                            ("NOTE" . ?n)
```

(continues on next page)

(continued from previous page)

```
                          ("READ" .?r)
                          ("CANCELLED" . ?c)
                          )))
;; Allow setting single tags without the menu
(setq org-fast-tag-selection-single-key (quote expert))
(setq org-agenda-tags-todo-honor-ignore-options t)
```

10.6 Others

```
;;;; * Custom Key Bindings

(setq org-agenda-clockreport-parameter-plist
      (quote (:link t :maxlevel 5 :fileskip0 t :compact t :narrow 80)))
;; Set default column view headings: Task Effort Clock_Summary
(setq org-columns-default-format "%80ITEM(Task) %10Effort(Effort){:} %10CLOCKSUM")
;; global Effort estimate values
;; global STYLE property values for completion
(setq org-global-properties (quote (("Effort_ALL" . "0:15 0:30 0:45 1:00 2:00 3:00␣
↪4:00 5:00 6:00 0:00")
                                    ("STYLE_ALL" . "habit"))))
(setq org-agenda-log-mode-items (quote (closed clock)))

(setq org-use-speed-commands t)
(defun bh/insert-inactive-timestamp ()
  (interactive)
  (org-insert-time-stamp nil t t nil nil nil))
(global-set-key (kbd "<f9> t") 'bh/insert-inactive-timestamp)

(defun yt/insert-ts-as-file ()
    (interactive)
  (insert (format-time-string "%Y-%m-%d--%H-%M-%S"))
  )

(global-set-key (kbd "<f9> T") 'yt/insert-ts-as-file)

(defun bh/insert-heading-inactive-timestamp ()
  (save-excursion
    (org-return)
    (org-cycle)
    (bh/insert-inactive-timestamp)))
(add-hook 'org-insert-heading-hook 'bh/insert-heading-inactive-timestamp 'append)
(setq org-file-apps (quote ((auto-mode . emacs)
                          ("\\.png\\'" . emacs)
                          ("\\.svg\\'" . system)
                          ("\\.mm\\'" . system)
                          ("\\.x?html?\\'" . system)
                          ("\\.pdf\\'" . system))))
                                    ; Overwrite the current window with the agenda
(setq org-agenda-window-setup 'current-window)

(setq org-time-clocksum-format
      '(:hours "%d" :require-hours t :minutes ":%02d" :require-minutes t))

;; (setq org-agenda-span 'day)
```

(continues on next page)

(continued from previous page)

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

(add-hook 'org-mode-hook (lambda () (abbrev-mode 1)))
```

10.7 Agenda

```
  ;; recursively find .org files in provided directory
  ;; modified from an Emacs Lisp Intro example
  (defun sa-find-org-file-recursively (&optional directory filext)
    "Return .org and .org_archive files recursively from DIRECTORY.
  If FILEXT is provided, return files with extension FILEXT instead."
    (interactive "DDirectory: ")
    (let* (org-file-list
       (case-fold-search t)         ; filesystems are case sensitive
       (file-name-regex "^[^.#].*") ; exclude dot, autosave, and backup files
       (filext (or filext "org$\\|org_archive"))
       (fileregex (format "%s\\.\\(%s$\\)" file-name-regex filext))
       (cur-dir-list (directory-files directory t file-name-regex)))
      ;; loop over directory listing
      (dolist (file-or-dir cur-dir-list org-file-list) ; returns org-file-list
    (cond
     ((file-regular-p file-or-dir) ; regular files
      (if (string-match fileregex file-or-dir) ; org files
          (add-to-list 'org-file-list file-or-dir)))
     ((file-directory-p file-or-dir)
      (dolist (org-file (sa-find-org-file-recursively file-or-dir filext)
                        org-file-list) ; add files found to result
        (add-to-list 'org-file-list org-file)))))))

  (setq org-agenda-files (append (sa-find-org-file-recursively "~/git/org")
                                 (sa-find-org-file-recursively "~/git/career")))

  (setq org-list-allow-alphabetical t)

  (defun yt/org-agenda-files--choose (candidate)
    (mapc 'identity (helm-marked-candidates)))


(defun yt/org-agenda-files-set-helm ()  ;; FIXME: path broken.
    (helm :sources '(((name . "Add directories to org-agenda-files variable")
                  (candidates . ("~/git/org/" "~/git/career" "~/git/org/finance"))
                  (action . yt/org-agenda-files--choose)))))

(defun yt/org-agenda-files-set ()
  (interactive)
  (setq org-agenda-files (list))
  (dolist (dir (yt/org-agenda-files-set-helm))
    (mapcar (lambda (arg)
          (add-to-list 'org-agenda-files arg))
        (sa-find-org-file-recursively dir))
        ))


        ;; (defun yt/org-agenda-files-set ()
```

(continues on next page)

(continued from previous page)

```
  ;; (interactive)
    ;; (setq org-agenda-files (yt/org-agenda-files-set-helm)))
  ;; (yt/org-agenda-files-set)

  (global-set-key (kbd "<C-f12>") 'org-agenda)

  ;; Do not dim blocked tasks
  (setq org-agenda-dim-blocked-tasks nil)

  ;; Compact the block agenda view
  (setq org-agenda-compact-blocks nil)



  ;; Custom agenda command definitions
  (defvar bh/hide-scheduled-and-waiting-next-tasks t)
  (setq org-agenda-custom-commands
    (quote (("N" "Notes" tags "NOTE"
             ((org-agenda-overriding-header "Notes")
              (org-tags-match-list-sublevels t)))
            ("h" "Habits" tags-todo "STYLE=\"habit\""
             ((org-agenda-overriding-header "Habits")
              (org-agenda-sorting-strategy
               '(todo-state-down effort-up category-keep))))
            ("d" "deadline" agenda ""
             (
              (org-agenda-entry-types '(:deadline))
              (org-agenda-start-day "2016-01-01")
              (org-agenda-span 'year)
              (org-agenda-include-diary nil)
              (org-agenda-show-all-dates nil)))
            ("s" "scheduled" agenda ""
             (
              (org-agenda-entry-types '(:scheduled))
              (org-agenda-start-day "2016-01-01")
              (org-agenda-span 'year)
              (org-agenda-include-diary nil)
              (org-agenda-show-all-dates nil)))
            (" " "Agenda"
             ((agenda "" nil)
              (tags-todo "-CANCELLED+WAITING|HOLD/!"
                         ((org-agenda-overriding-header (concat "Waiting and␣
↪Postponed Tasks (Ask them)"
                                                                (if bh/hide-scheduled-
↪and-waiting-next-tasks
                                                                    ""
                                                                  " (including␣
↪WAITING and SCHEDULED tasks)")))
                          (org-agenda-skip-function 'bh/skip-non-tasks)
                          (org-tags-match-list-sublevels nil)
                          (org-agenda-todo-ignore-scheduled bh/hide-scheduled-and-
↪waiting-next-tasks)
                          (org-agenda-todo-ignore-deadlines bh/hide-scheduled-and-
↪waiting-next-tasks)))
              (tags "RESPONSE"
                    ((org-agenda-overriding-header "Response (Make other's life␣
↪easier)")
```

(continues on next page)

(continued from previous page)

```
                    (org-tags-match-list-sublevels nil)))
              (tags-todo "-CANCELLED/!NEXT"
                         ((org-agenda-overriding-header (concat "Project Next Tasks␣
↪(Running out of things to do? pick one)"
                                                                (if bh/hide-scheduled-
↪and-waiting-next-tasks
                                                                    ""
                                                                  " (including␣
↪WAITING and SCHEDULED tasks)")))
                          (org-agenda-skip-function 'bh/skip-projects-and-habits-and-
↪single-tasks)
                          (org-tags-match-list-sublevels t)
                          (org-agenda-todo-ignore-scheduled bh/hide-scheduled-and-
↪waiting-next-tasks)
                          (org-agenda-todo-ignore-deadlines bh/hide-scheduled-and-
↪waiting-next-tasks)
                          (org-agenda-todo-ignore-with-date bh/hide-scheduled-and-
↪waiting-next-tasks)
                          (org-agenda-sorting-strategy
                           '(todo-state-down effort-up category-keep))))
              (tags-todo "-CANCELLED/!"
                         ((org-agenda-overriding-header "Stuck Projects (Make the␣
↪project flows, assign Next)")
                          (org-agenda-skip-function 'bh/skip-non-stuck-projects)
                          (org-agenda-sorting-strategy
                           '(category-keep))))
              (tags-todo "-HOLD-CANCELLED/!"
                         ((org-agenda-overriding-header "Projects (on-going)")
                          (org-agenda-skip-function 'bh/skip-non-projects)
                          (org-tags-match-list-sublevels 'indented)
                          (org-agenda-sorting-strategy
                           '(category-keep))))
              (tags-todo "-REFILE-CANCELLED-WAITING-HOLD/!"
                         ((org-agenda-overriding-header (concat "Project Subtasks␣
↪(Will do in the furture)"
                                                                (if bh/hide-scheduled-
↪and-waiting-next-tasks
                                                                    ""
                                                                  " (including␣
↪WAITING and SCHEDULED tasks)")))
                          (org-agenda-skip-function 'bh/skip-non-project-tasks)
                          (org-agenda-todo-ignore-scheduled bh/hide-scheduled-and-
↪waiting-next-tasks)
                          (org-agenda-todo-ignore-deadlines bh/hide-scheduled-and-
↪waiting-next-tasks)
                          (org-agenda-todo-ignore-with-date bh/hide-scheduled-and-
↪waiting-next-tasks)
                          (org-agenda-sorting-strategy
                           '(category-keep))))
              (tags-todo "-REFILE-CANCELLED-WAITING-HOLD/!"
                         ((org-agenda-overriding-header (concat "Standalone Tasks␣
↪(One-off/Small Tasks to pick)"
                                                                (if bh/hide-scheduled-
↪and-waiting-next-tasks
                                                                    ""
                                                                  " (including␣
↪WAITING and SCHEDULED tasks)")))
```

(continues on next page)

(continued from previous page)

```
                          (org-agenda-skip-function 'bh/skip-project-tasks)
                          (org-agenda-todo-ignore-scheduled bh/hide-scheduled-and-
↪waiting-next-tasks)
                          (org-agenda-todo-ignore-deadlines bh/hide-scheduled-and-
↪waiting-next-tasks)
                          (org-agenda-todo-ignore-with-date bh/hide-scheduled-and-
↪waiting-next-tasks)
                          (org-agenda-sorting-strategy
                           '(category-keep))))
              (tags "-REFILE/"
                    ((org-agenda-overriding-header "Tasks to Archive (Two month old)")
                     (org-agenda-skip-function 'bh/skip-non-archivable-tasks)
                     (org-tags-match-list-sublevels nil)))
              (tags "REFILE"
                    ((org-agenda-overriding-header "Tasks to Refile")
                     (org-tags-match-list-sublevels nil)))
              nil)))))

  ;; Limit restriction lock highlighting to the headline only
  (setq org-agenda-restriction-lock-highlight-subtree nil)

  ;; Always hilight the current agenda line
  (add-hook 'org-agenda-mode-hook
        '(lambda () (hl-line-mode 1))
        'append)

  ;;;; * agenda ignore items
  ;; Keep tasks with dates on the global todo lists
  (setq org-agenda-todo-ignore-with-date nil)

  ;; Keep tasks with deadlines on the global todo lists
  (setq org-agenda-todo-ignore-deadlines nil)

  ;; Keep tasks with scheduled dates on the global todo lists
  (setq org-agenda-todo-ignore-scheduled nil)

  ;; Keep tasks with timestamps on the global todo lists
  (setq org-agenda-todo-ignore-timestamp nil)

  ;; Remove completed deadline tasks from the agenda view
  (setq org-agenda-skip-deadline-if-done t)

  ;; Remove completed scheduled tasks from the agenda view
  (setq org-agenda-skip-scheduled-if-done t)

  ;; Remove completed items from search results
  (setq org-agenda-skip-timestamp-if-done t)

  (setq org-agenda-include-diary nil)
  (setq org-agenda-diary-file "~/git/org/diary.org")


  (setq org-agenda-insert-diary-extract-time t)

  ;; Include agenda archive files when searching for things
  (setq org-agenda-text-search-extra-files (quote (agenda-archives)))
```

(continues on next page)

(continued from previous page)

```
  ;; Show all future entries for repeating tasks
  (setq org-agenda-repeating-timestamp-show-all t)

  ;; Show all agenda dates - even if they are empty
  (setq org-agenda-show-all-dates t)

  ;; Sorting order for tasks on the agenda
  (setq org-agenda-sorting-strategy
    (quote ((agenda habit-down time-up user-defined-up effort-up category-keep)
            (todo category-up effort-up)
            (tags category-up effort-up)
            (search category-up))))



  ;; (setq org-agenda-tags-column -102)
  ;; Use sticky agenda's so they persist
  ;; (setq org-agenda-sticky t)
```

Enable display of the time grid so we can see the marker for the current time

```
(setq org-agenda-time-grid (quote ((daily today require-timed)
 (800 1000 1200 1400 1600 1800 2000)
 "......" "----------------")))

;; (quote ((daily today remove-match)
;;                                    #("----------------" 0 16 (org-heading t))
;;                                    (0700 0800 0900 1000 1100 1200 1200 1300 1400␣
↪1500 1600 1700))))
```

Start the weekly agenda on Monday.

```
(setq org-agenda-span 'week)
(setq org-agenda-start-on-weekday 1)
```

use 30 days.

```
(setq org-deadline-warning-days 30)
```

check clock entries if some are too long/short.

```
(setq org-agenda-clock-consistency-checks
      (quote (:max-duration "4:00"                ;; highligh clock entries longer␣
↪than 5 hours.
                        :min-duration "00:05"  ;; highlight clock smaller than 5 mins
                        :max-gap "00:05"       ;; highlight clock gap loger than 5␣
↪mins.
                        :gap-ok-around ("4:00"))))
(setq org-read-date-prefer-future 'time)
```

agenda reminder

```
;; Erase all reminders and rebuilt reminders for today from the agenda
(defun bh/org-agenda-to-appt ()
  (interactive)
  (setq appt-time-msg-list nil)
```

(continues on next page)

(continued from previous page)

```
  (setq appt-display-format 'window) ;; YT: show notification in separate window
  (org-agenda-to-appt))

                                     ; Rebuild the reminders everytime the agenda is␣
↪displayed
(add-hook 'org-finalize-agenda-hook 'bh/org-agenda-to-appt 'append)

                                     ; This is at the end of my .emacs - so␣
↪appointments are set up when Emacs starts
(bh/org-agenda-to-appt)
```

10.8 customised

```
(setq org-reverse-note-order t) ;; refiled headline will be the first under the taget

(setq org-archive-location "::* Archived Tasks") ;;in-file archive

(require 'org-habit)
(setq org-habit-show-all-today t)
(setq org-habit-show-habits nil)
(setq org-habit-graph-column 80)
;; add the following
(setq org-time-stamp-custom-formats '("<%A %d %B %Y>" . "<%A %d %B %Y %H:%M>"))
(setq org-agenda-tags-column 120)

(setq org-columns-default-format "%80ITEM(Task) %10Effort(Effort){:} %10CLOCKSUM
↪%10Mindfullness")
```

Start up options

```
(setq org-startup-folded t
      org-hide-block-startup t
      org-startup-indented nil)
```

10.9 Deep Configuration

Remove keys

```
;; remove C-TAB
(define-key org-mode-map (kbd "C-S-<right>") 'mc/mark-next-like-this)
(define-key org-mode-map (kbd "C-S-<left>") 'mc/mark-previous-like-this)
(org-defkey org-mode-map (kbd "C-c [") nil)
(org-defkey org-mode-map (kbd "C-c ]") nil)
(org-defkey org-mode-map (kbd "C-TAB") nil)
(org-defkey org-mode-map (kbd "<f8>") nil)
;; use helm iwth org
;; (setq org-completion-handler 'helm)
```

Show org-mode bullets as UTF-8 characters.

```
(require 'org-bullets)
(setq org-bullets-bullet-list '(;;; Large
                                "" "" ;"" ""
                                "" ""
                                "" ""
                                "" ""
                                "" "" ;""
                                "" ""
                                "" "" ;""
                                "" "" "" ""))
(setq org-bullets-bullet-list '("" "" "" "" "" "" "" "†" "" "" "" "" "" "" "" ""))
;; org ellipsis options, other than the default Go to Node...
;; not supported in common font, but supported in Symbola (my fall-back font) , ,
(setq org-ellipsis "");;
(add-hook 'org-mode-hook (lambda () (org-bullets-mode 1)))
```

org-download

```
(require 'org-download)
(setq-default org-download-image-dir "~/Downloads/img")
(setq-default org-download-heading-lvl nil)
;; (if (eq system-type 'darwin)
;;     "org-download: default download method"
;;     (setq org-download-screenshot-method "gnome-screens
;; hot -w --delay=1 -f %s"))
(setq org-download-image-wdith 400)

;; (setq org-download-screenshot-method "gnome-screenshot -a -f %s")
```

Add markup wrapper for org-mode. to turn a word into bold, wrapper in a selected region, by using expand-region, which is bound to C-= then type *.

```
(sp-local-pair 'org-mode "=" "=") ; select region, hit = then region -> =region= in␣
↪org-mode
(sp-local-pair 'org-mode "*" "*") ; select region, hit * then region -> *region* in␣
↪org-mode
(sp-local-pair 'org-mode "/" "/") ; select region, hit / then region -> /region/ in␣
↪org-mode
(sp-local-pair 'org-mode "_" "_") ; select region, hit _ then region -> _region_ in␣
↪org-mode
(sp-local-pair 'org-mode "+" "+") ; select region, hit + then region -> +region+ in␣
↪org-mode
(sp-local-pair 'org-mode "$" "$") ; select region, hit $ then region -> $region$ in␣
↪org-mode
```

10.10 External Links

```
(global-set-key (kbd "C-c l") 'org-store-link)
```

10.11 Babel

```
;;;; * org-babel
(setq org-src-window-setup 'current-window)
(setq org-src-fontify-natively nil)
(setq org-src-preserve-indentation nil)
(setq org-edit-src-content-indentation 0)
(setq org-catch-invisible-edits 'error)
(setq org-export-coding-system 'utf-8)
(prefer-coding-system 'utf-8)
(set-charset-priority 'unicode)
(setq default-process-coding-system '(utf-8-unix . utf-8-unix))
```

```
(defun bh/display-inline-images ()
  (condition-case nil
      (org-display-inline-images)
    (error nil)))


(add-hook 'org-babel-after-execute-hook 'bh/display-inline-images 'append)

(setq org-babel-results-keyword "results")
(org-babel-do-load-languages
 (quote org-babel-load-languages)
 (quote ((emacs-lisp . t) ;; TODO: simplifiy this list
     (R . t)
     (shell . t)
     (ledger . t)
     (org . t)
     (plantuml . t)
     (dot . t)
     (python .t)
     (ipython .t)
     ;; (bibtex .t)
     (octave . t)
     (latex . t)
     ;; (jupyter . t)
     (shell . t)
     (ledger . t)
     (sql . t))))

(setq org-babel-default-header-args (append org-babel-default-header-args
↪'((:colnames . "yes"))))

;; (add-to-list 'org-babel-default-header-args:R
;;              ;; '(:session . "*R-main*")
;;              '((:width . 640) (:height . 640)))


(setq org-confirm-babel-evaluate nil)

(setq org-plantuml-jar-path "~/git/.emacs.d/java/plantuml.jar") ;; TODO: change the␣
↪location..
;; Use fundamental mode when editing plantuml blocks with C-c '
(setq plantuml-jar-path "~/git/.emacs.d/java/plantuml.jar")
(add-to-list 'org-src-lang-modes (quote ("plantuml" . plantuml)))
```

10.12 Export

Add export back-end, I need HTML, PDF, MarkDown, and Ascii.

```
(require 'ox-html)
(require 'ox-latex)
(require 'ox-ascii)
(require 'ox-md)
(require 'htmlize)
(require 'ox-gfm)
```

General export options, it applys to all the export-backend.

```
(setq org-export-with-toc nil
      org-export-with-todo-keywords t
      org-export-with-sub-superscripts nil
      org-export-with-planning nil
      org-export-with-author t
      org-export-with-timestamps nil
      org-export-babel-evaluate t
      org-export-with-drawers nil)
```

```
(setq org-image-actual-width '(400))
```

Set the default format when exporting table to CSV.

```
(setq org-table-export-default-format "orgtbl-to-csv")
```

define the markups.

```
(setq org-emphasis-alist (quote (("*" bold "<b>" "</b>")
                                 ("/" italic "<i>" "</i>")
                                 ("_" underline "<span style=\"text-decoration:underline;\
↪">" "</span>")
                                 ("=" org-code "<code>" "</code>" verbatim)
                                 ("~" org-verbatim "<code>" "</code>" verbatim))))
```

10.13 PDF Export

```
;; http://emacs-fu.blogspot.co.uk/2011/04/nice-looking-pdfs-with-org-mode-and.html
;; 'djcb-org-article' for export org documents to the LaTex 'article', using
;; XeTeX and some fancy fonts; requires XeTeX (see org-latex-to-pdf-process)
(add-to-list 'org-latex-classes
         '("yt/org-article"
           "
\\documentclass[11pt,a4paper]{article}
\\usepackage{graphicx}    %% demo mode is a must when .img does not exists.
\\usepackage[T1]{fontenc}
\\usepackage{fontspec}
\\usepackage{hyperref}
\\hypersetup{
     colorlinks   = true,
     citecolor    = gray
}
```

(continues on next page)

(continued from previous page)

```
\\usepackage{amsmath}
\\usepackage{amstext}
\\usepackage{amssymb} %% checkbox
\\usepackage{commath}
\\DeclareMathOperator*{\\argmin}{\\arg\\!\\min} %% use $\argmin_{b}$
\\DeclareMathOperator*{\\argmax}{\\arg\\!\\max}
%% \\DeclareMathOperator{\\E}{\\mathbb{E}}
\\newcommand{\\E}[1]{{\\mathbb E}\\left[ #1 \\right]}
\\newcommand{\\Var}{\\mathrm{Var}}
\\DeclareMathOperator{\\P}{\\mathbb{Pr}}

\\usepackage{minted}
\\defaultfontfeatures{Mapping=tex-text}
% \\setromanfont[BoldFont={Gentium Basic Bold},
%                 ItalicFont={Gentium Basic Italic}]{Gentium Plus}
\\setsansfont{Charis SIL}
\\setmonofont[Scale=0.8]{DejaVu Sans Mono}
\\usepackage{geometry}
%% \\geometry{a4paper, textwidth=6.5in, textheight=10in,
 %%  marginparsep=7pt,
 %%  marginparwidth=1.2in, %% make sure it less than right=1.5in,
  %% otherwise, will go out of the paper
 %% right=1.5in, left=0.6in}

\\geometry{a4paper, textwidth=6.5in, textheight=10in,
        marginparsep=7pt, marginparwidth=.6in}
\\pagestyle{empty}

%% package from org-latex-default-packages-alist
\\usepackage{setspace}
\\onehalfspacing
\\usepackage{textcomp}
\\usepackage{marvosym}
\\usepackage{wasysym}
\\usepackage{ulem}
\\usepackage{amsthm}

\\theoremstyle{definition}
\\newtheorem{definition}{Definition}[section]% Conjecture is numbered
                           % within \section
\\newtheorem{lemma}[definition]{Lemma}
\\newtheorem{theorem}[definition]{Theorem}

\\newcommand{\\twodots}{\\mathinner {\\ldotp \\ldotp}}

%% \\renewcommand\\texttt[1]{{\\mint{cl}|#1|}}


\\usepackage{environ}
\\NewEnviron{note}{\\marginpar{\\footnotesize \\BODY}}

%% algorithm
\\usepackage{xcolor}
\\usepackage[linesnumbered]{algorithm2e}
\\newcommand\\mycommfont[1]{\\footnotesize\\ttfamily\\textcolor{blue}{#1}}
\\makeatletter
\\renewcommand{\\@algocf@capt@plain}{above}% formerly {bottom}
```

(continues on next page)

(continued from previous page)

```
\\makeatother


\\title{}
      [NO-DEFAULT-PACKAGES]
      [NO-PACKAGES]"
           ("\\section{%s}" . "\\section*{%s}")
           ("\\subsection{%s}" . "\\subsection*{%s}")
           ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
           ("\\paragraph{%s}" . "\\paragraph*{%s}")
           ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
(setq org-latex-default-class "yt/org-article")

(add-to-list 'org-latex-classes
         '("beamer"
           "\\documentclass\[presentation\]\{beamer\}
\\usepackage[T1]{fontenc}
\\usepackage{fontspec}
\\defaultfontfeatures{Mapping=tex-text}
% \\setromanfont[BoldFont={Gentium Basic Bold},
%                 ItalicFont={Gentium Basic Italic}]{Gentium Plus}
\\setsansfont{Charis SIL}
\\setmonofont[Scale=0.8]{DejaVu Sans Mono}
\\usepackage{minted}
[NO-DEFAULT-PACKAGES]
[NO-PACKAGES]
"
           ("\\section\{%s\}" . "\\section*\{%s\}")
           ("\\subsection\{%s\}" . "\\subsection*\{%s\}")
           ("\\subsubsection\{%s\}" . "\\subsubsection*\{%s\}")))

(require 'ox-beamer)

;; code highlights using minted package
(add-to-list 'org-latex-packages-alist '("" "minted"))
(setq org-latex-listings 'minted)
(setq org-latex-minted-options
      '(("frame" "lines")
    ("fontsize" "\\scriptsize")))
;; ("linenos" "")))

;;;; comple pdf
(setq org-latex-pdf-process
      '("xelatex -shell-escape -interaction=nonstopmode -output-directory %o %f"
    "xelatex -shell-escape -interaction=nonstopmode -output-directory %o %f"
    "xelatex -shell-escape -interaction=nonstopmode -output-directory %o %f"))
```

CHAPTER 11

Hydra

```
(defhydra yt-hydra/help (:color blue :hint nil)
  "
_f_unction: Documentation for a function
_v_ariable: Documentation for a variable
_i_nfo: info mode
_G_oogle: search google
_d_ictionary: search meaning of a word"
  ("f" describe-function)
  ("v" describe-variable)
  ("i" helm-info-org)
  ("G" helm-google-suggest)
  ("d" voca-builder/search-popup))
(global-set-key (kbd "<f1>") 'yt-hydra/help/body)
```

hydra

```
(require 'hydra)

(defhydra hydra-search (:color blue
                            :hint nil)
  "
Current Buffer   : _i_search helm-_s_woop _a_ce-jump-word
Multiple Buffers : helm-multi-_S_woop
Project Directory: projectile-_g_rep  helm-projectile-_G_rep
"
  ("i" isearch-forward)
  ("s" helm-swoop)
  ("a" ace-jump-word-mode)
  ("S" helm-multi-swoop)
  ("g" projectile-grep)
  ("G" helm-projectile-grep))
(global-set-key [f5] 'hydra-search/body)
```

CHAPTER 12

Emacs Lisp Programming

12.1 Org-Mode API

Get the link to current headline as an external link.

```
(defun yt/org-get-heading-link ()
  (interactive)
  (let* ((file-name (file-truename buffer-file-name))
     (headline (org-heading-components))
     (level (nth 0 headline))
     (title (nth 4 headline))
     (link (concat file-name
                   "::"
                   "*" ;; (yt/lisp-rep "*" level)
                   title)))
    (kill-new (concat "[["
                  link
                  "]["
                  (concat "headline: " title)
                  "]]"))))
```

CHAPTER 13

Refile

quickly filter out non-work tasks in org-agenda.

```
(defun yt/filter-life-agenda (tag)
  (concat "-" "life"))
(defun yt/filter-office-agenda (tag)
  (concat "-" "@office"))
(if (eq system-type 'darwin)
    (setq org-agenda-auto-exclude-function 'yt/filter-office-agenda)
  (setq org-agenda-auto-exclude-function 'yt/filter-life-agenda))
```

open this gnome-terminal here

```
(defun yt/open-terminal ()
  (interactive)
  (shell-command (concat "gnome-terminal "
                         "--working-directory="
                         (file-truename default-directory)
                         )))
;; (global-set-key (kbd "<f5>") 'yt/open-terminal)
```

use swiper to replace default `isearch`

```
(global-set-key "\C-s" 'swiper)
```

use snakemake-mode for smake file.

```
(add-to-list 'auto-mode-alist '("sfile" . snakemake-mode))
```

```
(defun yt/sh-chunk-args ()
(interactive)
(replace-string " -" " \\\n -")
)
```

insert git sha1 value into current point.

```
(defun yt/insert-git-hash-value ()
  (interactive)
  (insert (shell-command-to-string (concat "git rev-parse HEAD"))))
(global-set-key (kbd "<f9> s") 'yt/insert-git-hash-value)
```