Emacs Documentation

Release latest

Jan 08, 2018

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 do most of productive work.

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

my configration file is initally seaprated by differnt purpose or mode, but as it gorwths, it bcaem inconicient in tracking, and, As for other Emacs user, my configration is keep growthing, and This documents is first combined by 5 configration files, and it keep expanding, I use liteature programming to includes all the notes, and keep a log of how I use themn

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

Normally you could tangle a org file to extract all the source code into one file, that you could use. But I would like to push liteartpr programming furture 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 particullary 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*

```
(package-initialize)
(require 'org)
(setq org-confirm-babel-evaluate nil)  ;; evaluate src block without confirmation

;;;;;;; [2015-01-22 Thu 21:27]
(defvar endless/init.org-message-depth 5
  "What depth of init.org headers to message at startup.")

(with-temp-buffer
  (insert-file "~/git/.emacs.d/yt/init.org")
  (goto-char (point-min))

  ;; org babels
  (search-forward "\n* Babel Library")
  (org-copy-subtree)
  (let ((tmp-file (make-temp-file "tmp")))
    (with-temp-file tmp-file (yank))
    (org-babel-lob-ingest tmp-file))

  ;; emacs lisp functions
  (goto-char (point-min))
  (search-forward "\n* Emacs Configuration")
  (while (not (eobp))
    (forward-line 1)
    (cond
     ;; Report Headers
     ((looking-at
       (format "\\*\\{2,%s\\} +.*$"
               endless/init.org-message-depth))
      (message "%s" (match-string 0)))
     ;; (message (format (current-time-string))))
     ;; Evaluate Code Blocks
     ((looking-at "[\s]*\\#\\+begin_src\semacs-lisp")
```

```
      ;; ((looking-at "#\\+BEGIN_SRC +emacs-lisp.*$")
      ;; ((looking-at "^#\\+BEGIN_SRC +.*$")
       (org-babel-execute-src-block))
     ;; Finish on the next level-1 header
     ((looking-at "^\\* End")
      (goto-char (point-max))))))
```

CHAPTER 3

install packages

```
(setq my-package-list '(ess
                        ssh
                        auto-complete
                        nyan-mode
                        yasnippet
                        projectile
                        magit
                        helm-swoop
                        nyan-mode
                        org-jekyll
                        org-plus-contrib
                        helm-projectile
                        rainbow-delimiters
                        zenburn-theme
                        htmlize
                        nanowrimo
                        golden-ratio
                        artbollocks-mode
                        langtool
                        flycheck
                        expand-region
                        guide-key
                        exec-path-from-shell
                        smart-mode-line
                        smart-mode-line-powerline-theme
                        powerline
                        synosaurus
                        hydra
                        w3m
                        ace-window
                        calfw
                        multiple-cursors
                        org-download
                        paradox
                        smartparens
```

```
                        ace-jump-mode
                        voca-builder
                        org-download
                        gist
                        sunshine
                        keyfreq
                        pretty-mode
                        f
                        olivetti
                        helm-mu
                        ))

(require 'package)
(setq package-archives '(("melpa" . "http://melpa.milkbox.net/packages/")
                         ("org" . "http://orgmode.org/elpa/")
                         ("gnu" . "http://elpa.gnu.org/packages/")
                         ("marmalade" . "http://marmalade-repo.org/packages/")))
(package-initialize)

;; fetch the list of packages available
(unless package-archive-contents
  (package-refresh-contents))
;; install
(dolist (i-package my-package-list)
  (unless (package-installed-p i-package)
    (package-install i-package)))
```

CHAPTER 4

General

4.1 Remove Keybind

```
(global-unset-key (kbd "C-x b"))
(global-unset-key (kbd "C-x C-b"))
```

4.2 Assorted Pieces

Define an auto-save feature, the backup files are saved in ~.emacs.d/backup/ folder. I've used it several times to recover the most recent, but un-saved version, after power off.

```
;; 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)

;; start auto-complete with emacs
(require 'auto-complete)
```

```
;; do default config for auto-complete
(require 'auto-complete-config)
(ac-config-default)
```

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

Shows an notication for invalid operations.

```
(setq visible-bell 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)
;; (setq yas-snippet-dirs '("~/git/.emacs.d/my-snippets"
;;                          "~/git/.emacs.d/.cask/24.4.2/elpa/yasnippet-20141102.1554/
↪snippets"
;;                          "~/git/.emacs.d/.cask/25.0.50.1/elpa/yasnippet-20141102.
↪1554/snippets"))
```

4.3 Window Layout/Navigation

Quickly jump between windows using `ace-window`, I used it frequently and bind it `F1`.

```
(require 'ace-window)
(global-set-key (kbd "<f1>") '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)
```

4.4 Utilities

```
;; Change "yes or no" to "y or n"
(fset 'yes-or-no-p 'y-or-n-p)

(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!"))
```

Use keyfreq package to record the commands I use in Emacs.

```
(require 'keyfreq)
(keyfreq-mode 1)
(keyfreq-autosave-mode 1)
```

4.5 System Path/Keyboard

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

24. 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)))
```

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)
```

It is a terrible idea to have lines of context that expand the whole screen, especially nowadays we have wide-screens, which just make it is hard to read. 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.

```
(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

`savehist` is an very powerful mode.

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

CHAPTER 5

UI - Emacs Looks Cool

I never click any buttons in the menu-bar/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.

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

Mode-line contains most of the relevant information about the buffer, Git status, Major mode, clock info, and current time. I disable the display of minor modes, there are just too many and almost all are irrelevant.

```
(display-time-mode)
(require 'smart-mode-line)
(setq powerline-arrow-shape 'curve)
(setq powerline-default-separator-dir '(right . left))
(setq sml/theme 'powerline)
(setq sml/mode-width 0)
(setq sml/name-width 20)
(rich-minority-mode 1)
(setf rm-blacklist "")
(sml/setup)
```

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

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.

```
;; (when window-system
;;   (load-theme 'zenburn t))
```

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
      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)
```

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-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
;; (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 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)))))))

(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)))))

(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)))))

(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)))


(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"
  ("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)
  )
;; (global-set-key [f3] 'hydra-file-management/body)
```

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

```
;; 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
    (kill-new (file-truename buffer-file-name))))
```

7.2 Projectile - Directory Access

I only use *projectile* to jump between different git folders, so there isn't much configuration except using `helm` for selection.

```
(require 'projectile)
(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)
```

7.3 Remote (SSH)

Sometimes, I need to sync between local and remote machine, I can do it in termnial or in Emacs.

```
(require 'tramp)
(require 'ssh)
(setq password-cache-expiry nil)

;; (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:JBA28:/home/local/JBANORTHWEST/yitang/LR_share")
;;   ;; (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)
```

I use Emacs/org-mode as a unified system to do everything, at home and in the office. I used to use Dropbox/Copy to automatically sync the files on multiple machines, but this work-flow can be dangerous. Image if I was editing same files on two machines at the same time, then I can't track which is which.

The good thing about Git is that you can see what exactlly has been changed by each version, and auto log, with commit information and timesatmp. Magit helps me to do it conviently but I need 3 more features:

1. automatically save all the buffers

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)))
```

1. quick rush

I shared lift with my colluge, and someimes I left 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/save-git-backup` 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 snippts.

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

REPOS=\"org jbarm\"
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/save-git-backup ()
  (interactive)
  (yt/save-all-buffers)
  (yt/git-backup))
```

1. automatically commit

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.

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

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 t)
(setq comint-move-point-for-output t)
(setq ess-eval-visibly-p 'nowait) ;; no waiting while ess evalating
(defun my-ess-start-R ()
  (interactive)
  (if (not (member "*R-main*" (mapcar (function buffer-name) (buffer-list))))
      (progn
        (delete-other-windows)
        (setq w1 (selected-window))
        (setq w1name (buffer-name))
        (setq w2 (split-window w1 nil t))
        (R)
        (set-window-buffer w2 "*R*")
        (rename-buffer "*R-main*")
        (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")
```

8.1 Start-up

when R start, it will load few local settings, one of them is the user-setting, which is R scripts saved in ~/RProfile. I'd like to have same settings on both my local, and remote server. and this can be achieved by using `ess-post-run-hook`.

```
(setq yt/ess--RProfile-string "
#### change this file name to .Rprofile and place to ~/userName so when R starts, the␣
↪following command will be processed automatically

## http://stackoverflow.com/questions/1189759/expert-r-users-whats-in-your-rprofile
options(\"width\"=160)                # wide display with multiple monitors
options(\"digits.secs\"=3)            # show sub-second time stamps
options(\"repos\" = c(CRAN = \"http://www.stats.bris.ac.uk/R/\")) # hard code the UK␣
↪repo for CRAN
options(\"max.print\" = 200)
## from the AER book by Zeileis and Kleiber
options(prompt=\"R> \", digits=4, show.signif.stars=FALSE)

.Libs <- function(){
    library(data.table)
    library(ggplot2)
    library(gridExtra)
##    library(sp)
##    library(geosphere)
##    library(rgeos)
##    library(sp)
##    library(dragonfly)
}

.libPaths(\"~/R_libs\")
## improved list of objects
.ls.objects <- function (pos = 1, pattern, order.by,
                 decreasing=FALSE, head=FALSE, n=5)
    {
        napply <- function(names, fn) sapply(names, function(x)
            fn(get(x, pos = pos)))
        names <- ls(pos = pos, pattern = pattern)
        obj.class <- napply(names, function(x) as.character(class(x))[1])
        obj.mode <- napply(names, mode)
        obj.type <- ifelse(is.na(obj.class), obj.mode, obj.class)
        obj.prettysize <- napply(names, function(x) {
                                    capture.output(print(object.size(x), units = \
↪\"auto\")) })
        obj.size <- napply(names, object.size)
        obj.dim <- t(napply(names, function(x)
            as.numeric(dim(x))[1:2]))
        vec <- is.na(obj.dim)[, 1] & (obj.type != \"function\")
        obj.dim[vec, 1] <- napply(names, length)[vec]
        out <- data.frame(obj.type, obj.size, obj.prettysize, obj.dim)
        names(out) <- c(\"Type\", \"Size\", \"PrettySize\", \"Rows\", \"Columns\")
        if (!missing(order.by))
            out <- out[order(out[[order.by]], decreasing=decreasing), ]
        if (head)
            out <- head(out, n)
        out
    }
## shorthand
```

```
lsos <- function(..., n=10) {
    .ls.objects(..., order.by=\"Size\", decreasing=TRUE, head=TRUE, n=n)
}")

(add-hook 'ess-post-run-hook
          (lambda ()
            (goto-char (point-max))
            (insert yt/ess--RProfile-string)
            (inferior-ess-send-input) ;; execuate the R scripts
            ;; clean up
            (search-backward "Type 'q()' to quit R.")
            (next-line)
            (delete-region (point) (point-max))
            (inferior-ess-send-input)
            ))
```

8.2 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)
```

8.3 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 `org-strct-mode`, to show/hide the whole section, I havne;t tryied it before, but it sounds a good idea.

```
(add-hook 'prog-mode-hook 'hs-minor-mode)
(global-set-key (kbd "<f3>") 'hs-toggle-hiding)
(global-set-key (kbd "S-<f3>") 'hs-show-all) ;; S->show
(global-set-key (kbd "C-<f3>") 'hs-hide-all)
;;   hs-hide-block                      C-c @ C-h
;;   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)
```

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 80) ;; 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
(require 'cl-lib)
(require 'color)
(cl-loop
 for index from 1 to rainbow-delimiters-max-face-count
 do
 (let ((face (intern (format "rainbow-delimiters-depth-%d-face" index))))
   (cl-callf color-saturate-name (face-foreground face) 30)))
```

8.4 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")))
```

8.5 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)))
```

8.6 Scripts editing

8.7 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)
```

```
  (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)
```

clean R console

```
;;;; * clean up ESS or sh 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)
```

```
(add-hook 'ess-mode-hook '(lambda ()
                            (turn-on-orgstruct)
                            (setq-local orgstruct-heading-prefix-regexp "#### ")))
```

CHAPTER 9

Writing in Emacs

9.1 English Language

I type quit fast with lots of misspell and in writing, I don't need to correct every single one when writing, which will stop the flow. I will do it afterwards in editing, I will press C-, to move the cursor to next misspelled word, and press `C-.` to correcct it, press it again, to correct it to another words.

```
;; check spelling
(add-hook 'text-mode-hook 'flyspell-mode)
(add-hook 'org-mode-hook 'flyspell-mode)
(setq ispell-dictionary "british"
      ispell-extra-args '() ;; TeX mode "-t"
      ispell-silently-savep t)
(if (eq system-type 'darwin)
    (setq ispell-program-name "/usr/local/bin/aspell")
  (setq ispell-program-name "/usr/bin/aspell"))
(setq ispell-personal-dictionary "~/git/.emacs.d/personal/ispell-dict") ;; add␣
↪personal dictionary
```

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")
```

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"))
```

For about one month, I tried to write at least 500 words per day. I also set up a special `write-mode` that has different color scheldules that helps me to set the moode.

```
;; [2014-12-25 Thu 22:21]
(defun yt/write-mode ()
  (interactive)
  (hl-sentence-mode)
  (variable-pitch-mode)
  (nanowrimo-mode))

;; word count
;; https://bitbucket.org/gvol/nanowrimo.el
(require 'org-wc)
(require 'nanowrimo)
(setq nanowrimo-today-goal 500)


;; [2014-12-23 Tue 22:06]
;; Highlight sentence
;; https://www.gnu.org/software/emacs/manual/html_node/elisp/Attribute-Functions.html
(require 'hl-sentence)
(add-hook 'nanowrimo-mode 'hl-sentence-mode)
(set-face-attribute 'hl-sentence-face nil
                    ;; :foreground "black")
                    :foreground "white")
(add-hook 'nanowrimo-mode 'variable-pitch-mode)
(set-face-attribute 'variable-pitch nil
                    :foreground "gray40")
```

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)
```

9.2 Random Quotes

If I run out of idea, and I didn't write anything for 1 minutes, Emacs will pop a random quote that I collected in the echo area. The random quotes can inspire me sometimes.

```
(defconst yt/quotes
  '("You can't see paradise, if you don't pedal.  - Chicken Run "
    "He who who says he can and he who says he can't are both usually right  Confucius
↪"
    "Why waste time proving over and over how great you are when you could be getting␣
↪better? - Dweck The Mindset"
    "You're not a failure until you start to assign blame. - The legendary basketball␣
↪coach John Wooden"
    "I could hear my heart beating. I could hear everyone's heart. I could hear the␣
↪human noise we sat there making, not one of us moving, not even when the room went␣
↪dark. - Raymond Carver"
    "A writer is a sum of their experiences. Go get some - Stuck in Love (2012)"
    "If there is any one secret of success, it lies in the ability to get the other␣
↪person's point of view and see things from that person's angle as well as from your␣
↪own. - Henry Ford"
    "People who can put themselves in the place of other people who can understand␣
↪the workings of their minds, need never worry about what the future has in store␣
↪for them. - Owen D. Young"
    )
  "Good quotes
   they can be useful for creative writers as well.")
(defun yt/show-random-quotes ()
  "Show random quotes to minibuffer"
  (interactive)
  (message "%s"
           (nth (random (length yt/quotes))
                yt/quotes)))
(run-with-idle-timer 60 t 'yt/show-random-quotes)
```

9.3 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_{tanguk}	aboutme
eqt	equivalent to	english
iif	if and only if	maths
wrt	with respect to	english

```
(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)
    (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)
```

9.4 Style

CHAPTER 10

Browser - eww

Emacs has build-in browser *eww* and I used it for reading articles for its inability to process multi-media content and sometimes I turn off the display of image, so I can fully focus on the context.

```
;; http://emacs.stackexchange.com/questions/561/how-can-i-toggle-displaying-images-in-
↪eww-without-a-page-refresh
(defvar-local endless/display-images t)

(defun endless/toggle-image-display ()
  "Toggle images display on current buffer."
  (interactive)
  (setq endless/display-images
        (null endless/display-images))
  (endless/backup-display-property endless/display-images))

(defun endless/backup-display-property (invert &optional object)
  "Move the 'display property at POS to 'display-backup.
Only applies if display property is an image.
If INVERT is non-nil, move from 'display-backup to 'display
instead.
Optional OBJECT specifies the string or buffer. Nil means current
buffer."
  (let* ((inhibit-read-only t)
         (from (if invert 'display-backup 'display))
         (to (if invert 'display 'display-backup))
         (pos (point-min))
         left prop)
    (while (and pos (/= pos (point-max)))
      (if (get-text-property pos from object)
          (setq left pos)
        (setq left (next-single-property-change pos from object)))
      (if (or (null left) (= left (point-max)))
          (setq pos nil)
        (setq prop (get-text-property left from object))
        (setq pos (or (next-single-property-change left from object)
                      (point-max)))
```

```
        (when (eq (car prop) 'image)
          (add-text-properties left pos (list from nil to prop) object))))))
```

CHAPTER 11

Org mode

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

11.1 customised

punch-in

```
(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)
```

```
(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)
```

```
(setq org-archive-location "::* Archived Tasks") ;;in-file archive
(setq org-habit-show-habits nil)
(setq org-agenda-span 'week)
```

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
```

clock-in tasks in mode-line

```
(set-face-attribute 'org-mode-line-clock nil
                    :weight 'bold :box '(:line-width 1 :color "#FFBB00") :foreground
↪"white" :background "#FF4040")
(setq org-reverse-note-order t) ;; refiled headline will be the first under the taget
```

CHAPTER 12

mu4e - Email Client

The advantage of use Emacs as an email client:

1. communication happens at the point where the content is generated. as a statisician/programmer, most likely I need to communicate with numbers, table, graphs, or snippet. I could just copy these results from to email, do a quick editing.
2. HTML email with CSS style. I like to format my email use headingline, fonts, and highlight the code, I used to be write a report in Word/LatEx and write an email with only one line, please see the attachment. which I don't like.
3. search properly I use search all the time, and this functionality is not working at all in outlook 2013, it also shows up random info (mail.app in osx did a great job).

Disadvantage and the things it can't do:

1. book Meeting/Appointment I am not aware if you can do it in emacs, and we need an iterative way to do. Outlook Schedule Asistant does a good job, it lists agenda of all attendence, and I could spot one time slot that suit for all or most people.
2. don't expect me to reply immediately this is how it works: 1. download the email from server, 2) index with head, body, attachment, user name etc. 3) Emacs talk to and show in GUI. I usually update the email about 30 minutes. but it helps me out of disrupts.
3. calendar I don't know how to integrate Office 365 calendar with Org-mode calandar yet. even it can, I doubt I can download and see other people's agenda.

This setting need two programs to work: 1) mu, 2) offlineimap.

to install *mu* on osx,

```
brew install mu
```

```
;; usage:
;; $ offlineimap
;; $ mu index
;; M-x mu4e
;; from mu's official manual
```

```
;;----------------------------------------------------------
(add-to-list 'load-path "~/git/.emacs.d/elpa/mu4e")
(require 'mu4e)
(setq mu4e-mu-binary "/usr/local/bin/mu")
;; default

;; don't save message to Sent Messages, Gmail/IMAP takes care of this
(setq mu4e-sent-messages-behavior 'sent)

;; allow for updating mail using 'U' in the main view:
(setq mu4e-get-mail-command "offlineimap -a JBA")
(setq mu4e-update-interval 300)

;; sending mail -- replace USERNAME with your gmail username
;; also, make sure the gnutls command line utils are installed
;; package 'gnutls-bin' in Debian/Ubuntu
(setq smtpmail-default-smtp-server "smtpserver") ; needs to be specified before the␣
↪(require)
(require 'smtpmail)


;; don't keep message buffers around
(setq message-kill-buffer-on-exit t)


;; attempt to show images when viewing messages
(setq mu4e-view-show-images t
      mu4e-show-images t
      mu4e-view-image-max-width 800)


;; (setq mu4e-html2text-command "html2text -utf8 -width 72") ;; nil "Shel command␣
↪that converts HTML
;; ref: http://emacs.stackexchange.com/questions/3051/how-can-i-use-eww-as-a-renderer-
↪for-mu4e
(defun my-render-html-message ()
  (let ((dom (libxml-parse-html-region (point-min) (point-max))))
    (erase-buffer)
    (shr-insert-document dom)
    (goto-char (point-min))))
(setq mu4e-html2text-command 'my-render-html-message)

;; yt
(setq mu4e-view-prefer-html t) ;; try to render
(add-to-list 'mu4e-view-actions
             '("ViewInBrowser" . mu4e-action-view-in-browser) t) ;; read in browser
;; mu4e as default email agent in emacs
(setq mail-user-agent 'mu4e-user-agent)
(require 'org-mu4e)
                                        ;== M-x org-mu4e-compose-org-mode==
(setq org-mu4e-convert-to-html t) ;; org -> html
                                        ; = M-m C-c.=

;; give me ISO(ish) format date-time stamps in the header list
(setq  mu4e-headers-date-format "%Y-%m-%d %H:%M")

;; customize the reply-quote-string
;; M-x find-function RET message-citation-line-format for docs
```

```
(setq message-citation-line-format "%N @ %Y-%m-%d %H:%M %Z:\n")
(setq message-citation-line-function 'message-insert-formatted-citation-line)

;; the headers to show in the headers list -- a pair of a field
;; and its width, with `nil' meaning 'unlimited'
;; (better only use that for the last field.
;; These are the defaults:
(setq mu4e-headers-fields
    '( (:date          .  25)
       (:flags         .   6)
       (:from          .  22)
       (:subject       .  nil)))

;; don't keep message buffers around
(setq message-kill-buffer-on-exit t)
;; attachments go here
(setq mu4e-attachment-dir  "~/Downloads")

;; should mu4e use fancy utf characters? NO. they're ugly.
;;(setq mu4e-use-fancy-chars 't)
```

use helm-mu to search emails

```
(global-set-key (kbd "<f9> e") 'helm-mu)
```

12.1 Account

set up email account, use office 365 in the office, and iCloud at macbook pro.

```
(setq  mu4e-maildir-shortcuts  '(("/JBA/INBOX"     . ?j)
                                 ("/iCloud/INBOX" . ?i)
                                 ("/Gmail/INBOX" . ?g)
                                 ("/Sent Items"   . ?s)
                                 ("/Trash"        . ?t)
                                 ("/All Mail"     . ?a)))


;; (defun yt/email-jba ()
;;   (interactive)
;;   ;; setup for smtp
;;   (setq message-send-mail-function 'smtpmail-send-it
;;         smtpmail-stream-type 'starttls
;;         smtpmail-default-smtp-server "smtp.office365.com"
;;         smtpmail-smtp-server "smtp.office365.com"
;;         smtpmail-smtp-service 587
;;         smtpmail-smtp-user "yi.tang@jbarisk.com"
;;         ;; account info
;;         user-mail-address "yi.tang@jbarisk.com"
;;         user-full-name  "Yi Tang"
;;         ;; mu4e
;;         mu4e-drafts-folder "/JBA/Drafts"
;;         mu4e-sent-folder   "/JBA/Sent Items"
;;         mu4e-trash-folder  "/JBA/Trash"
;;         mu4e-maildir-shortcuts  '(("/JBA/INBOX"     . ?i)
;;                                   ("/Sent Items"   . ?s)
```

```
;;                                    ("/Trash"        . ?t)
;;                                    ("/All Mail"     . ?a))
;;         mu4e-compose-signature (concat
;;                                 "Yi Tang\n"
;;                                 "Statistician\n"
;;                                 "T: +44 (0) 1756 799919\n")))

(defun yt/email-icloud ()
  (setq  ;; account info
   user-mail-address "yi.tang.uk@me.com"
   user-full-name  "Yi Tang"
   message-send-mail-function 'smtpmail-send-it
   smtpmail-stream-type 'starttls
   smtpmail-default-smtp-server "smtp.mail.me.com"
   smtpmail-smtp-server "smtp.mail.me.com"
   smtpmail-smtp-service 587
   smtpmail-smtp-user "yi.tang.uk@me.com"
   mu4e-maildir "~/Maildir"
   mu4e-drafts-folder "/iCloud/Drafts"
   mu4e-sent-folder   "/iCloud/Sent Messages"
   mu4e-trash-folder  "/iCloud/Deleted Messages"

   mu4e-maildir-shortcuts  '(("/iCloud/INBOX"    . ?i)
                             ("/Sent Items"   . ?s)
                             ("/Trash"        . ?t)
                             ("/All Mail"     . ?a))


   mu4e-compose-signature (concat
                           " (Yi Tang)\n"
                           "Email: yi.tang.uk@me.com\n"
                           "\n")))

(defun yt/email-gmail ()

  ;; sent emails
  (setq message-send-mail-function 'smtpmail-send-it
        starttls-use-gnutls t
        smtpmail-starttls-credentials '(("smtp.gmail.com" 587 nil nil))
        ;; smtpmail-auth-credentials
        ;;   '(("smtp.gmail.com" 587 "yi.tang.uni@gmail.com" nil))
        smtpmail-default-smtp-server "smtp.gmail.com"
        smtpmail-smtp-server "smtp.gmail.com"
        smtpmail-smtp-service 587
        smtpmail-smtp-user "yi.tang.uni@gmail.com")



  (setq user-mail-address "yi.tang.uni@gmail.com" )
  (setq user-full-name  "Yi Tang" )
  (setq mu4e-drafts-folder "/Gmail/Drafts" )
  (setq mu4e-sent-folder   "/Gmail/Sent Items" )
  (setq mu4e-trash-folder  "/Gmail/Trash" )
  (setq mu4e-maildir-shortcuts  '(("/Gmail/INBOX"    . ?i)
                                  ("/Sent Items"   . ?s)
                                  ("/Trash"        . ?t)
                                  ("/All Mail"     . ?a)))
  (setq mu4e-compose-signature (concat
```

```
                                "Yi Tang\n"
                                "Statistician\n"
                                "T: 07445510033\n")))



(defun yt/mu4e-jba ()
  (interactive)
  (yt/email-jba)
  (mu4e)
  )
(defun yt/mu4e-gmail()
  (interactive)
  (yt/email-gmail)
  (mu4e))
(defun yt/mu4e-icloud()
  (interactive)
  (yt/email-icloud)
  (mu4e))

(defhydra hydra-email (:color blue :hint nil)
  "
Mu4e: _g_mail _j_ba _i_Cloud"
  ("g" yt/mu4e-gmail)
  ("j" yt/mu4e-jba)
  ("i" yt/mu4e-icloud))
(global-set-key [f2] 'hydra-email/body)

;; (if (string= WhereAmI "Office")
;;     (yt/email-jba)
;;   (yt/email-icloud))

(yt/email-icloud)
```

when I send an email, it will prompt and ask for email address, I only need to type once per Emacs session, also, I can save the password to an *.authoty* file.

12.2 Contacts

have problem with BBDB installtion, and use org-contact.el to manage contact. adding contact is very easy. I can use tab to complete contacts, which is really handy.

```
(require 'org-contacts)
(setq org-contacts-files '("~/git/org/contacts.org"))
;; (setq mu4e-org-contacts-file  "~/git/org/contacts.org")
(add-to-list 'mu4e-headers-actions
             '("org-contact-add" . mu4e-action-add-org-contact) t)
(add-to-list 'mu4e-view-actions
             '("org-contact-add" . mu4e-action-add-org-contact) t)
```

12.3 Workflow

I am trying to avoid use `C-x m` to write/sent email directy, unless it is really short. otherwise, if it relates to a project, I will make an org headline to keep track the project communciation, to do that, I compose email/message in org mode, then sent the whole subtree by `C-c M-o`.

```
(require 'org-mime)
(setq org-mime-library 'mml)
(add-hook 'message-mode-hook
          (lambda ()
            (local-set-key "\C-c\M-o" 'org-mime-htmlize)))
(add-hook 'org-mode-hook
          (lambda ()
            (local-set-key "\C-c\M-o" 'org-mime-org-buffer-htmlize)))
(add-hook 'org-mime-html-hook
          (lambda ()
            (insert-file-contents "~/git/.emacs.d/personal/css/office.css")
            ;; (goto-char 5)
            )
          t)


(add-hook 'org-mode-hook
          (lambda ()
            (local-set-key (kbd "C-c M-o") 'org-mime-subtree))
          'append)
```

just in case I didn't get the format right at the first place and need some quick fix in message mode, save me few seconds in going back to org-mode.

```
(add-hook 'message-mode-hook 'orgstruct++-mode 'append)
(add-hook 'message-mode-hook 'turn-on-auto-fill 'append)
;; (add-hook 'message-mode-hook 'bbdb-define-all-aliases 'append)
(add-hook 'message-mode-hook 'orgtbl-mode 'append)
(add-hook 'message-mode-hook 'turn-on-flyspell 'append)
;; (add-hook 'message-mode-hook
;;           '(lambda () (setq fill-column 270))
;;           'append)
```

12.4 TODO Comprehensive Email

CHAPTER 13

Refile

voca-builde

```
(require 'voca-builder)
(setq voca-builder/voca-file "~/git/org/vocabulary.org")
(setq voca-builder/current-tag "General")
(global-set-key (kbd "<f4>") 'voca-builder/search-popup)
(setq sentence-end-double-space nil)
```

weather forecasting

```
;; sunshine: weather forcaste service
(setq sunshine-units 'metric)
(setq sunshine-location "Keighley, GB")
```

Add ad wrapper function `yt/qs-bakcup-log` for auto$_{\text{bakc.el}}$. call this function at the end of the day.

```
(defun yt/qs-backup-log ()
  (interactive)
  (load "~/git/qs/auto_back.el"))
```

CHAPTER 14

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-zoom (global-map "<f2>")
  "zoom"
  ("g" text-scale-increase "in")
  ("l" text-scale-decrease "out"))

(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 15

Emacs Lisp Programming

use org-struct mode for emacs-lisp.

```
(add-hook 'emacs-lisp-mode-hook '(lambda ()
                                   (turn-on-orgstruct)
                                   (setq-local orgstruct-heading-prefix-regexp ";;;; ")))
```

(rep “a” 3) –> “aaa”

```
(defun yt/lisp-rep (arg n)
  (apply 'concat (make-list n arg)))
```

15.1 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
                      "]["
                      title
                      "]]"))))
```

CHAPTER 16

Babel Library

to load a

```
(org-babel-lob-ingest "~/git/.emacs.d/yi.babel.org)
```

16.1 org-contact

```
(defun GetContactGroup (contact-file search-tag contact-type)
  (interactive)
  (let ((name-address (org-map-entries (lambda()
                                         (cons (elt (org-heading-components) 4)  ;; get␣
↪headline (name)
                                               (org-entry-get nil contact-type))) ;; get␣
↪Email/phone etc property
                                       search-tag
                                       contact-file)))
    (mapconcat (lambda (a-list)
                 (concat (car a-list) " <" (cdr a-list) ">"))
               name-address ", ")))
(GetContactGroup contact-file search-tag contact-type)
```

16.2 Ledger

add the following to .ledgerrc

```
--start-of-week=1 -f ~/git/ImportantFiles/ledger
```

```
ledger --period "this week"  org register Exp -S "date, amount"
```

```
ledger --period "this week"  bal Exp
```

```
#+call: Weekly_Transaction()
2015-04-06 GiffGaff              Expenses:Bill:Phone         £ 10.00      £ 10.00
2015-04-06 Tesco                 Expenses:Food:Grocery       £ 15.25      £ 25.25
2015-04-08 HS                    Expenses:Transport           £ 4.20      £ 29.45
2015-04-08 Tesco                 Expenses:Food:Grocery       £ 14.11      £ 43.56
2015-04-10 HS                    Expenses:Food:Lunch          £ 2.50      £ 46.06
2015-04-10 Tesco                 Expenses:Speical:Cig         £ 5.48      £ 51.54
15-Apr-06 GiffGaff               Expenses:Bill:Phone         £ 10.00      £ 10.00
15-Apr-06 Tesco                  Expenses:Food:Grocery       £ 15.25      £ 25.25
15-Apr-08 HS                     Expenses:Transport           £ 4.20      £ 29.45
15-Apr-08 Tesco                  Expenses:Food:Grocery       £ 14.11      £ 43.56
15-Apr-10 HS                     Expenses:Food:Lunch          £ 2.50      £ 46.06
15-Apr-10 Tesco                  Expenses:Speical:Cig         £ 5.48      £ 51.54
```

CHAPTER 17

Archive

17.1 open stree map

```
(defun omap_bbox (minlon minlat maxlon maxlat)
  "generate to hyperlink and also view an area in openstreemap.
the area is defined by the bounding box. "
;;   (interactive)
  (let ((address (concat  "http://www.openstreetmap.org/?"
           "minlon=" (number-to-string minlon) "&"
           "minlat=" (number-to-string minlat) "&"
           "maxlon=" (number-to-string maxlon) "&"
           "maxlat=" (number-to-string maxlat) "&")))
    (browse-url address)
    (kill-new address)))


(defun omap_point (lon lat)
;;  (interactive)
  (let ((address (concat  "http://www.openstreetmap.org/?"
                          "mlat=" (number-to-string lat) "&"
                          "mlon=" (number-to-string lon))))
    (browse-url address)
    (kill-new address)))

(defun omap_site (site)
;;  (interactive)
  (let ((address (concat "http://nominatim.openstreetmap.org/search.php?q=" site)))
    (browse-url address)
    (kill-new address)))

(defun omap (arg)
;;  (interactive)
  (cond ((= 1 (length arg))
         (omap_site (nth 0 arg)))
```

```
        ((= 2 (length arg))
         (omap_point (nth 0 arg) (nth 1 arg)))
        ((= 4 (length arg))
         (omap_bbox (nth 0 arg)
                    (nth 1 arg)
                    (nth 2 arg)
                    (nth 3 arg)))
        (t "nope")))

;; (omap '(-0.489 51.28 0.236 51.686)) ;; london area
;; http://www.openstreetmap.org/?minlon=-0.489&minlat=51.28&maxlon=0.236&maxlat=51.
↪686&
;; (omap '(-76.3412 38.6710)) ;; New york
;; http://www.openstreetmap.org/?mlat=38.671&mlon=-76.3412
;; (omap '("UK"))  ;; UK, obvs
;; http://nominatim.openstreetmap.org/search.php?q=UK
```

17.2 Office Setting

[2015-03-18 Wed 18:32] In the office, I have differnet settings for

1. org-agenda files
2. org-capture-templates,
3. org-html-export

In my office Ubuntu machine, set a variable WhereAmI to Office, then the setting will be done automatically. wrap all the settings in the `my/all-office-settings` function for I can call it explicity when I work on my Mac.

```
;; (setq WhereAmI "Office") ;; set this variable in ~/.emacs
(defun my/office-org-agenda-file ()
  (setq org-agenda-files '("~/git/jbarm" )))

(defun my/office-org-capture-temp ()
  (setq org-capture-templates
        (quote (("t" "todo" entry (file "~/git/jbarm/re_org_refile.org")
                 "* TODO %?\nSCHEDULED:%t \n %U\n" :clock-in t :clock-resume t) ;;␣
↪TODO: %? %U %a, what does these means??? %: %c
                ("r" "respond" entry (file "~/git/jbarm/re_org_refile.org")
                 "* To %? about :RESPONSE:  \nSCHEDULED: %t\n%U\n" :clock-in t :clock-
↪resume t)
                ("n" "note" entry (file "~/git/jbarm/re_org_refile.org")
                 "* %? :NOTE:\n%U\n" :clock-in t :clock-resume t)
                ("j" "Journal" entry (file+datetree "~/git/jbarm/re_org_diary.org")
                 "* %?\n%U\n" :clock-in t :clock-resume t)
                ("l" "Ledger Journal" plain (file "~/git/org/Finance/ledger")
                 "%(org-read-date) * %^{Payee}\n\t%^{Account}\t£ %^{Amount}
↪\n\tAssets:Checking" :immediate-finish :clock-in t :clock-resume t)
                ("h" "Habit" entry (file "~/git/jbarm/re_org_habits.org")
                 "* NEXT %?\n%U\nSCHEDULED: %(format-time-string \"<%Y-%m-%d .+1d/3d>\
↪")\n:PROPERTIES:\n:STYLE: habit\n:REPEAT_TO_STATE: NEXT\n:END:\n")
                ("v" "Vocabulary" entry (file "~/git/jbarm/re_org_refile.org")
                 "* %? :VOCA:\n%U" :clock-in t :clock-resume t)
                ))))

(defun my/all-office-settings ()
```

```
  (interactive)
  (my/office-org-agenda-file)
  (my/office-org-capture-temp)
  (setq org-html-head-extra "<link rel=\"stylesheet\" href=\"/home/yitang/git/.emacs.
↪d/personal/css/office.css\" type=\"text/css\" />"
        org-html-head "<link rel=\"stylesheet\" href=\"\\SKI-CQV5LZ1\static\yi_report.
↪css\" type=\"text/css\" />"
        ;; voca-builder/voca-file "~/git/org/vocabulary-office.org"
        )

  )

(when (equal "Office" WhereAmI)
  (my/all-office-settings))
```

17.3 Git-Report

```
;; (defun yt/git-repo-info ()
;;   (interactive)
;;   (let* ((sh-num-unpushed-commits "git status | grep \"\'origin/master\'\" | grep -
↪Po \"[0-9]\"")
;;        (sh-num-uncommited-files "git status --porcelain 2>/dev/null| egrep \"^(M|␣
↪M)\" | wc -l")
;;        (unpush (shell-command-to-string sh-num-unpushed-commits))
;;        (uncommit (shell-command-to-string sh-num-uncommited-files)))
;;     (concat "unpushed commits: " unpush "\n" "uncommited files: " uncommit)))

(defun yt/git-check-repo-update (path)
  (interactive)
  (let* ((git-cmd "git status | grep clean")
         (sh-cmd (concat "cd " path "; " git-cmd))
         (sh-result (shell-command-to-string sh-cmd))
         (is-git-repo (not (string= "fatal: Not a git repository (or any of the␣
↪parent directories): .git\n"
                                    sh-result)))
         (repo-clean-p (string= "nothing to commit, working directory clean\n"
                                sh-result)))
    (cond ((not is-git-repo)
           (concat path " is not a git repo"))
          (repo-clean-p
           (concat path " is clean"))
          (t
           (concat path " is dirty")))))

(defun yt/git-status-report (repo-list)
  (interactive)
  (let ((repo-status (mapconcat 'yt/git-check-repo-update repo-list "\n")))
    repo-status))


(defun yt/git-status-report-org-mode (repo-list)
  (interactive)
  (let ((repo-status (mapcar 'yt/git-check-repo-update-return-list repo-list)))
    repo-status))
```

```
(defun yt/git-check-repo-update-return-list (path)
  (let* ((status (yt/git-check-repo-update path))
         (status (car (cdr (split-string status " is "))))
         (magit-link
          (if (string= "not a git repo" status)
              " "
            (concat "[[magit:" path "]]"))))
    (list path
          status
          magit-link)))

(defun yt/list-to-org-table (a-list)
  (let ((a-list-string (mapconcat 'identity a-list " | "))) ;; use | to separate each item
    (concat "| " a-list-string " |" "\n")))

(defun yt/git-check-repo-update-org-table (repo-list)
  (let* ((status-list (mapcar 'yt/git-check-repo-update-return-list repo-list))
         (status-table (mapcar 'yt/list-to-org-table status-list))
         (status-table-string (mapconcat 'identity status-table " "))
         (table-header "| repo | status | magit |\n"))
    (concat table-header " " status-table-string)))

(require 'f)
(defun yt/git-all-git-dir (git-base-dir)
  (f-directories "~/git"
                 '(lambda (arg)
                    (let* ((f (directory-files arg))
                           (f-string (format "%s" f)))
                      (if (string-match "\\.git" f-string)
                          t
                        nil)))
                 t))

(defun yt/git-generate-report ()
  (interactive)
  (let  ((git-report (yt/git-check-repo-update-org-table repo-list)))
    (with-temp-buffer
      (insert (format "%s" git-report))
      (write-region (point-min) (point-max) git-report-temp-file)
      )
    (find-file git-report-temp-file)
    (goto-char (+ 1 (point-min)))
    (org-mode)
    (org-ctrl-c-ret)
    (previous-line 2)
    (search-forward "status")
    (next-line 2)
    ;; (setq current-prefix-arg '(4)) ; C-u
    (org-table-sort-lines t ?a)
    )
  )


(setq git-report-temp-file "~/git-report-buffer")
;; (setq repo-list (yt/git-all-git-dir "~/git"))
(setq re-post '("/Users/yitang/git/voca-builder" "/Users/yitang/git/qs" "/Users/
↪yitang/git/portfolio-statistician" "/Users/yitang/git/portfilio-yitang" "/Users/
↪yitang/git/org" "/Users/yitang/git/myblog" "/Users/yitang/git/jbarm" "/Users/yitang/
↪git/Free-Classic-Books" "/Users/yitang/git/.emacs.d" "/Users/yitang/git/R/yiR" "/
↪Users/yitang/git/R/activitydashboard2" "/Users/yitang/git/R/RExercise" "/Users/
↪yitang/git/R/Archive/yiR" "/Users/yitang/git/R/Archive/orgR" "/Users/yitang/git/R/
↪Archive/learn_shiny")("/Users/yitang/git/voca-builder" "/Users/yitang/git/qs" "/
↪Users/yitang/git/portfolio-statistician" "/Users/yitang/git/portfilio-yitang" "/
↪Users/yitang/git/org" "/Users/yitang/git/myblog" "/Users/yitang/git/jbarm" "/Users/
```

```
;; (yt/git-generate-report)

;; | repo                                       | status | magit
↪                  |
;; |--------------------------------------------+--------+-----------------------------
↪------------------|
;; | /Users/yitang/git/voca-builder             | dirty  | [[magit:/Users/yitang/git/
↪voca-builder]]            |
;; | /Users/yitang/git/qs                       | dirty  | [[magit:/Users/yitang/git/
↪qs]]                      |
;; | /Users/yitang/git/portfolio-statistician | clean  | [[magit:/Users/yitang/git/
↪portfolio-statistician]] |
;; | /Users/yitang/git/portfilio-yitang       | clean  | [[magit:/Users/yitang/git/
↪portfilio-yitang]]        |
;; | /Users/yitang/git/org                      | dirty  | [[magit:/Users/yitang/git/
↪org]]                     |
;; | /Users/yitang/git/myblog                   | dirty  | [[magit:/Users/yitang/git/
↪myblog]]                  |
;; | /Users/yitang/git/jbarm                    | clean  | [[magit:/Users/yitang/git/
↪jbarm]]                   |
;; | /Users/yitang/git/Free-Classic-Books     | dirty  | [[magit:/Users/yitang/git/
↪Free-Classic-Books]]      |
;; | /Users/yitang/git/.emacs.d                 | dirty  | [[magit:/Users/yitang/git/.
↪emacs.d]]                 |
;; | /Users/yitang/git/R/yiR                    | dirty  | [[magit:/Users/yitang/git/R/
↪yiR]]                     |
;; | /Users/yitang/git/R/activitydashboard2   | clean  | [[magit:/Users/yitang/git/R/
↪activitydashboard2]]    |
;; | /Users/yitang/git/R/RExercise              | clean  | [[magit:/Users/yitang/git/R/
↪RExercise]]               |
;; | /Users/yitang/git/R/Archive/yiR            | dirty  | [[magit:/Users/yitang/git/R/
↪Archive/yiR]]             |
;; | /Users/yitang/git/R/Archive/orgR           | dirty  | [[magit:/Users/yitang/git/R/
↪Archive/orgR]]            |
;; | /Users/yitang/git/R/Archive/learn_shiny    | clean  | [[magit:/Users/yitang/git/R/
↪Archive/learn_shiny]]    |
```

17.4 C++

```
;; (global-set-key [(f9)] 'compile)
(setq compilation-window-height 2)
(setq compilation-finish-function
      (lambda (buf str)

        (if (string-match "exited abnormally" str)

            ;;there were errors
            (message "compilation errors, press C-x ` to visit")

          ;;no errors, make the compilation window go away in 0.5 seconds
          (run-at-time 0.5 nil 'delete-windows-on buf)
          (message "NO COMPILATION ERRORS!"))))
```

```
(defun indent-buffer-ask()
   (when (y-or-n-p "Indent buffer before saving? ")
     (indent-region (point-min) (point-max))))

(defun indent-buffer-no-ask()
  (indent-region (point-min) (point-max)))

(setq c++-mode-hook
      '(lambda ()
         (c-set-style "cc-mode")
         (define-key c++-mode-map "\C-c\C-c" 'compile)
         (define-key c++-mode-map "\C-c\C-e" 'next-error)
;         (add-hook 'before-save-hook 'indent-buffer-ask nil t)
         (add-hook 'before-save-hook 'indent-buffer-no-ask nil t)  ;; indent c++␣
↪files after save.
))




(require 'flymake-google-cpplint)
(add-hook 'c++-mode-hook 'flymake-google-cpplint-load)
;; (custom-set-variables
;; '(flymake-google-cpplint-command "/Library/Python/2.7/site-packages/cpplint/
↪cpplint.py"))
;"/usr/local/bin/cpplint"))

; start google-c-style with emacs
(require 'google-c-style)
(add-hook 'c-mode-common-hook 'google-set-c-style)
(add-hook 'c++-mode-common-hook 'google-make-newline-indent)
```

17.5 tryout.el

```
;;change default browser for 'browse-url'  to w3m
(setq browse-url-browser-function 'w3m-goto-url-new-session)
;;change w3m user-agent to android
(setq w3m-user-agent "Mozilla/5.0 (Linux; U; Android 2.3.3; zh-tw; HTC_Pyramid Build/
↪GRI40) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.")
;; (setq w3m-user-agent "Emacs-w3m/1.4.540 w3m/0.5.3+debian-15")


(defun yt/hello ()
  "functon meant to be called first thing in the morning.

It will open four windows:
1. weather of today and the next few days,
2. my weekly calendar, without habits shown,
3. habits,
4. git repo rpeort. "

  ;; switch to *Org Agenda(a)* buffer

  (sunshine-forecast) ;; switch to *Sunshine* buffer
  (yt/git-generate-report) ;; switch to git-report-
```

```
  )


(defun yt/bye ()
  "function meat to be called before I leave

It reminds of me to
1. sync git folder,
2. back up keyfreq file"
  (interactive)
  (yt/git-generate-report)
  (goto-char (point-max))
  (insert "
  (yt/daily-back-keyfreq)")
  )


(defun yt/git-repo-info ()
  (interactive)
  (let* ((sh-num-unpushed-commits "git status | grep \"\'origin/master\'\" | grep -Po␣
↪\"[0-9]\"")
         (sh-num-uncommited-files "git status --porcelain 2>/dev/null| egrep \"^(M|␣
↪M)\" | wc -l")
         (unpush (shell-command-to-string sh-num-unpushed-commits))
         (uncommit (shell-command-to-string sh-num-uncommited-files)))
    (concat "unpushed commits: " unpush "\n" "uncommited files: " uncommit)))
```

17.6 Update Emacs library

```
cask install --path ~/git/.emacs.d
cask upgrade-cask --path ~/git/.emacs.d
git add -A
git commit -m "cask update $(date)"
git push
```

```
#+call: UpdateEmacsPackages()

#+results:
: [master 3f5c750] cask update Sat Nov  8 18:06:08 GMT 2014
:  2 files changed, 7 insertions(+), 3 deletions(-)
:  delete mode 120000 .#yi.babel.org
```

17.7 image process

```
## http://www.linuxjournal.com/content/tech-tip-extract-pages-pdf
function pdfpextr()
{
    # this function uses 3 arguments:
    #     $1 is the first page of the range to extract
    #     $2 is the last page of the range to extract
    #     $3 is the input file
    #     output file will be named "inputfile_pXX-pYY.pdf"
```

```
    # TODO: add format=png option
    gs -sDEVICE=pdfwrite -dNOPAUSE -dBATCH -dSAFER \
       -dFirstPage=${1} \
       -dLastPage=${2} \
       -sOutputFile=${3%.pdf}_p${1}-p${2}.pdf \
       ${3}
}
pdfpextr $pageStart $pageEnd $pdfFile
```

```
#+name: Extract_PDF_Pages[:dir ~/git/org/img](1, 10, "tmp.pdf")
```

17.8 Python

```
(require 'eval-in-repl)
;; Python support
;; (require 'python) ; if not done elsewhere
(require 'eval-in-repl-python)
(define-key python-mode-map (kbd "<C-return>") 'eir-eval-in-python)
```