Author: Timo Lassmann
Date: January 21, 2018

Emacs Configuration File

General Settings

1. Whoami

```
(setq user-full-name "Timo Lassmann"
      user-mail-address "timo.lassmann@telethonkids.org.au")
```

2. Emacs directory

```
(defconst tl/emacs-directory (concat (getenv "HOME") "/.emacs.d/"))
(defun tl/emacs-subdirectory (d) (expand-file-name d tl/emacs-directory))
```

```
(message "%s" (tl/emacs-subdirectory "elisp"))
(add-to-list 'load-path (tl/emacs-subdirectory "elisp"))
```

3. Sensible setting

```
(load-file "~/.emacs.d/elisp/sensible-defaults.el")
(sensible-defaults/increase-gc-threshold)
(sensible-defaults/use-all-settings)
(sensible-defaults/use-all-keybindings)
```

4. Very general

```
(setq scroll-conservatively 10000
      scroll-preserve-screen-position t)
```

```
(let ((path-from-shell (shell-command-to-string "/bin/bash -l -c 'echo
   $PATH'")))
  (setenv "PATH" path-from-shell)
  (setq exec-path (split-string path-from-shell path-separator)))
```

```
(setq temporary-file-directory "/tmp")
```

5. Modernizing Emacs

Found here how to remove the warnings from the GnuTLS library when using HTTPS… increase the minimum prime bits size:

```
(setq gnutls-min-prime-bits 4096)
```

6. misc

Assume that I always want to kill the current buffer when hitting C-x k.

```
(defun tl/kill-current-buffer ()
  "Kill the current buffer without prompting."
  (interactive)
  (kill-buffer (current-buffer)))
(global-set-key (kbd "C-x k") 'tl/kill-current-buffer)
```

Never use tabs. Tabs are the devil's whitespace.

```
(setq-default indent-tabs-mode nil)
```

Tell emacs about the mactex installation…

```
(setenv "PATH" (concat "/Library/TeX/texbin" ":" (getenv "PATH")))
```

This should do it…

7. Start in fullscreen

Start in fullscreen mode

```
(custom-set-variables '(initial-frame-alist (quote ((fullscreen .
   maximized)))))
```

Package Initialization

1. Package Manager

Emacs has become like every other operating system, and now has a package manager with its own collection repository, but since it is so conservative, we need to add more repositories to get all the sweet goodness, I demand.

```
(load-library "url-handlers")
     (require 'package)

     (setq package-archives '(("org" . "http://orgmode.org/elpa/")
                              ("gnu" . "http://elpa.gnu.org/packages/")
                              ("melpa"     . "http://melpa.org/packages/"
                                 )))

     (package-initialize)
     (package-refresh-contents)
```

2. Use-Package

Using use-package to automatically install certain packages, as well as the ease of lazily loading them.

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

(require 'use-package)
```

3. Init File Support

Load up a collection of enhancements to Emacs Lisp, including dash, s for string manipulation, and f for file manipulation.

```
(require 'cl)
```

```
(use-package dash
  :ensure t
  :config (eval-after-load "dash" '(dash-enable-font-lock)))

(use-package s
  :ensure t)

(use-package f
  :ensure t)
```

4. Fill Mode

Automatically wrapping when you get to the end of a line (or the fill-region):

```
(use-package fill
  :bind (("C-c T f" . auto-fill-mode)
         ("C-c T t" . toggle-truncate-lines))
  :init (add-hook 'org-mode-hook 'turn-on-auto-fill)
  :diminish auto-fill-mode)
```

5. Undo / redo

According to this article, I get better functionality than the redo+ plugin (which I can't seem to get working well).

```
(use-package undo-tree
  :ensure t
  :diminish undo-tree-mode
  :init
  (global-undo-tree-mode 1)
  :config
  (defalias 'redo 'undo-tree-redo)
  :bind (("C-z" . undo)     ; Zap to character isn't helpful
         ("C-S-z" . redo)))
```

End.

Backup

```
(setq backup-directory-alist '(("." . "~/Dropbox/backup")))
```

Tramp should do the same:

```
(setq tramp-backup-directory-alist backup-directory-alist)
```

Make backups of files, even when they're in version control:

```
(setq vc-make-backup-files t)
```

And let's make sure our files are saved if we wander off and defocus the Emacs application:

```
(defun save-all ()
  "Save all dirty buffers without asking for confirmation."
  (interactive)
  (save-some-buffers t))

(add-hook 'focus-out-hook 'save-all)
```

End.

Movement

1. SMEX

```
(use-package smex
  :ensure t
  :init (smex-initialize)
  :bind ("M-x" . smex)
  ("M-X" . smex-major-mode-commands))
```

2. IDO

```
(use-package ido
  :ensure t
  :init  (setq ido-enable-flex-matching t
               ido-ignore-extensions t
               ido-use-virtual-buffers t
               ido-everywhere t
                 ido-enable-tramp-completion nil)
  :config
  (ido-mode 1)
  (ido-everywhere 1)
  (add-to-list 'completion-ignored-extensions ".o")
  (add-to-list 'completion-ignored-extensions ".elc")
  (add-to-list 'completion-ignored-extensions "~")
  (add-to-list 'completion-ignored-extensions ".bin")
  (add-to-list 'completion-ignored-extensions ".bak")
  (add-to-list 'completion-ignored-extensions ".obj")
  (add-to-list 'completion-ignored-extensions ".map")
  (add-to-list 'completion-ignored-extensions ".a")
  (add-to-list 'completion-ignored-extensions ".ln")
  (add-to-list 'completion-ignored-extensions ".mod")
  (add-to-list 'completion-ignored-extensions ".gz")
  (add-to-list 'completion-ignored-extensions ".aux")
  (add-to-list 'completion-ignored-extensions ".tdo")
  (add-to-list 'completion-ignored-extensions ".fmt")
```

```
  (add-to-list 'completion-ignored-extensions ".swp")
  (add-to-list 'completion-ignored-extensions ".pdfsync")
  (add-to-list 'completion-ignored-extensions ".pdf")
  (add-to-list 'completion-ignored-extensions ".vrb")
  (add-to-list 'completion-ignored-extensions ".idx")
  (add-to-list 'completion-ignored-extensions ".ind")
  (add-to-list 'completion-ignored-extensions ".bbl")
  (add-to-list 'completion-ignored-extensions ".toc")
  (add-to-list 'completion-ignored-extensions ".blg")
  (add-to-list 'completion-ignored-extensions ".snm")
  (add-to-list 'completion-ignored-extensions ".ilg")
  (add-to-list 'completion-ignored-extensions ".log")
  (add-to-list 'completion-ignored-extensions ".out")
  (add-to-list 'completion-ignored-extensions ".pyc")
  (add-to-list 'completion-ignored-extensions ".DS_Store")
  (add-to-list 'completion-ignored-extensions "-blx.bib")
  (add-to-list 'completion-ignored-extensions ".run.xml")
  (add-to-list 'completion-ignored-extensions ".fls")
  (add-to-list 'completion-ignored-extensions ".fdb_latexmk")
  (add-to-list 'completion-ignored-extensions ".bcf")
  (add-to-list 'completion-ignored-extensions ".rel")
  (add-to-list 'completion-ignored-extensions ".epub")
  )
```

FLX package:

```
(use-package flx-ido
  :ensure t
  :init (setq ido-enable-flex-matching t
              ido-use-faces t)
  :config (flx-ido-mode 1))
```

According to Ryan Neufeld, we could make IDO work vertically, which is much easier to read. For this, I use ido-vertically:

```
(use-package ido-vertical-mode
  :ensure t
  :init                ; I like up and down arrow keys:
  (setq ido-vertical-define-keys 'C-n-C-p-up-and-down)
  :config
  (ido-vertical-mode 1))
```

This sorts an IDO filelist by mtime instead of alphabetically.

```
(defun ido-sort-mtime ()
  "Reorder the IDO file list to sort from most recently modified."
  (setq ido-temp-list
```

```
        (sort ido-temp-list
              (lambda (a b)
                (ignore-errors
                  (time-less-p
                   (sixth (file-attributes (concat ido-current-directory
                      b)))
                   (sixth (file-attributes (concat ido-current-directory
                      a)))))))))
  (ido-to-end  ;; move . files to end (again)
   (delq nil (mapcar
              (lambda (x) (and (char-equal (string-to-char x) ?.) x))
              ido-temp-list))))

(add-hook 'ido-make-file-list-hook 'ido-sort-mtime)
(add-hook 'ido-make-dir-list-hook 'ido-sort-mtime)
```

3. Ace-window

```
(use-package ace-window
  :ensure t
  :init
  (setq aw-keys '(?1 ?2 ?3 ?4 ?5 ?6 ?7 ?8))
  :diminish ace-window-mode)
(global-set-key (kbd "C-x o") 'ace-window)
```

4. Avy

```
(use-package avy
  :ensure t
  :init (setq avy-background t))
(global-set-key (kbd "M-s") 'avy-goto-char-timer)
```

5. Line Numbers

```
(use-package linum
  :ensure t
  :init
  (add-hook 'prog-mode-hook 'linum-mode)
  (add-hook 'linum-mode-hook (lambda () (set-face-attribute 'linum nil :
     height 110)))

  :config
  (defun linum-fringe-toggle ()
    "Toggles the line numbers as well as the fringe."    (interactive)
    (cond (linum-mode (fringe-mode '(0 . 0))
                      (linum-mode -1))
```

```
          (t          (fringe-mode '(8 . 0))
                      (linum-mode 1))))

  :bind (("A-C-k"   . linum-mode)
         ("s-C-k"   . linum-mode)
         ("A-C-M-k" . linum-fringe-toggle)
         ("s-C-M-k" . linum-fringe-toggle)))
```

relative

```
(use-package linum-relative
  :ensure t
  :config
  (defun linum-new-mode ()
    "If line numbers aren't displayed, then display them.
     Otherwise, toggle between absolute and relative numbers."
    (interactive)
    (if linum-mode
        (linum-relative-toggle)
      (linum-mode 1)))

  :bind ("A-k" . linum-new-mode)
  ("s-k" . linum-new-mode))   ;; For Linux
```

6. Turn off mouse FIXME - why here

```
(mouse-wheel-mode -1)
(global-set-key [wheel-up] 'ignore)
(global-set-key [wheel-down] 'ignore)
(global-set-key [double-wheel-up] 'ignore)
(global-set-key [double-wheel-down] 'ignore)
(global-set-key [triple-wheel-up] 'ignore)
(global-set-key [triple-wheel-down] 'ignore)
```

Org-mode

1. General setup

load org mode

```
(use-package org
  :init
  (setq org-use-speed-commands t
        org-return-follows-link t
        org-completion-use-ido t
        org-outline-path-complete-in-steps nil))
```

Directory, inbox ..

```
(setq org-directory "~/Dropbox")
(defun org-file-path (filename)
  "Return the absolute address of an org file, given its relative name."
  (concat (file-name-as-directory org-directory) filename))
(setq org-index-file (org-file-path "/capture/inbox.org"))
(setq org-archive-location
      (concat (org-file-path "archive.org") "::* From %s"))
```

Multiple files for agenda source:

```
;;   (setq org-agenda-files (list org-index-file))
(setq org-agenda-files '("~/Dropbox/capture"
                         "~/Dropbox/work"
                         "~/Dropbox/planning"
                         "~/Dropbox/life"))
```

Refile targets / create new targets if necessary

```
;;(setq org-refile-targets '((org-agenda-files :maxlevel . 3)))
 (setq org-refile-targets '(("~/Dropbox/work/work-todo.org" :maxlevel .
    2)
                            ("~/Dropbox/life/life-todo.org" :maxlevel .
                               2)
                            ("~/Dropbox/org/gtd/tickler.org" :maxlevel .
                                2)))
  (setq org-refile-use-outline-path 'file)
  (setq org-refile-allow-creating-parent-nodes 'confirm)
(setq org-refile-allow-creating-parent-nodes 'confirm)
```

Drawers

```
(setq org-log-into-drawer t)

;; Add the REPORT drawer
(setq org-drawers '("PROPERTIES" "CLOCK" "LOGBOOK" "REPORT"))
```

Hitting C-c C-x C-s will mark a todo as done and move it to an appropriate place in the archive.

```
(defun tl/mark-done-and-archive ()
  "Mark the state of an org-mode item as DONE and archive it."
  (interactive)
  (org-todo 'done)
  (org-archive-subtree))

;;    (define-key org-mode-map (kbd "C-c C-x C-s") 'tl/
   mark-done-and-archive)
```

Record the time that a todo was archived.

```
(setq org-log-done 'time)
```

2. Capture

Capture templates..

```
(setq org-capture-templates
      (quote (("t" "todo" entry (file+headline org-index-file "Inbox")
               "* TODO %?\nSCHEDULED: %(org-insert-time-stamp (
                  org-read-date nil t \"+0d\"))\n%a\n")
              ("n" "note" entry (file+headline org-index-file "Inbox")
               "* %?\n\n  %i\n\n  See: %a" :empty-lines 1)
              ("r" "respond" entry (file+headline org-index-file "Inbox")
               "* TODO Respond to %:from on %:subject\nSCHEDULED: %(
                  org-insert-time-stamp (org-read-date nil t \"+0d\"))\n%
                  a\n")
              ("m" "Mail" entry (file+headline org-index-file "Inbox")
               "* TODO %?\n%a   %:from %:fromname %:fromaddress" :prepend
                   t :jump-to-captured t)
              ("p" "Daily Plan" plain (file+datetree "~/Dropbox/planning/
                 daily-plan.org")
               "+ [ ] The 3 most important tasks [/]
                 - [ ]
                 - [ ]
                 - [ ]
              + [ ] Other tasks that are in the system [/]
                 - [ ]
              + [ ] ToDos which are not tracked by my system [/]
                 - [ ] " :immediate-finish t)
              )))
```

3. Taking Meeting Notes

directly from https://github.com/howardabrams/dot-files/blob/master/emacs-org.org) I've notice that while I really like taking notes in a meeting, I don't always like the multiple windows I have opened, so I created this function that I can easily call to eliminate distractions during a meeting.

```
(defun meeting-notes ()
  "Call this after creating an org-mode heading for where the notes for
     the meeting
should be. After calling this function, call 'meeting-done' to reset the
   environment."
  (interactive)
  (outline-mark-subtree)                                ;; Select org-mode
     section
  (narrow-to-region (region-beginning) (region-end))  ;; Only show that
     region
  (deactivate-mark)
```

```
  (delete-other-windows)                                ;; Get rid of other
      windows
  (text-scale-set 3)                                    ;; Text is now
     readable by others
  (fringe-mode 0)
  (message "When finished taking your notes, run meeting-done."))
```

Of course, I need an ‘undo’ feature when the meeting is over…

```
(defun meeting-done ()
  "Attempt to 'undo' the effects of taking meeting notes."
  (interactive)
  (widen)                                           ;; Opposite of
     narrow-to-region
  (text-scale-set 0)                                ;; Reset the font size
     increase
  (fringe-mode 1)
  (winner-undo))                                    ;; Put the windows back
     in place
```

End.

4. Coding

Allow babel to evaluate C …

```
(org-babel-do-load-languages
 'org-babel-load-languages
 '((C . t)
   (R . t)
   (emacs-lisp . t)
   (sh . t)
   (makefile . t)
   (latex .t)
   ))
```

Don’t ask before evaluating code blocks.

```
(setq org-confirm-babel-evaluate nil)
```

smart brackets in export

```
(setq org-export-with-smart-quotes t)
```

Done.

5. Export

Export packages…

```
(require 'ox-latex)
(require 'ox-beamer)
```

Htmlize required for reveal…

```
(use-package htmlize
  :ensure t)
```

6. Flyspell

Enable spell-checking in Org-mode.

```
(add-hook 'org-mode-hook 'flyspell-mode)
```

7. Color and display

Use syntax highlighting in source blocks while editing.

```
(setq org-src-fontify-natively t)
```

Make TAB act as if it were issued in a buffer of the language's major mode.

```
(setq org-src-tab-acts-natively t)
```

When editing a code snippet, use the current window rather than popping open a new one (which shows the same information).

```
(setq org-src-window-setup 'current-window)
```

8. Image preview

Inline images support:

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

(add-to-list 'org-latex-packages-alist
             '("" "tikz" t))

(eval-after-load "preview"
  '(add-to-list 'preview-default-preamble "\\PreviewEnvironment{
     tikzpicture}" t))
(setq org-latex-create-formula-image-program 'imagemagick)


(setq org-confirm-babel-evaluate nil)
(add-hook 'org-babel-after-execute-hook 'org-display-inline-images)
(add-hook 'org-mode-hook 'org-display-inline-images)
```

9. Keybindings

Standard bindings

```
(define-key global-map "\C-cl" 'org-store-link)
(define-key global-map "\C-ca" 'org-agenda)
(define-key global-map "\C-cc" 'org-capture)
```

Quickly open index file

```
(defun open-index-file ()
  "Open the master org TODO list."
  (interactive)
  (find-file org-index-file)
  (flycheck-mode -1)
  (end-of-buffer))

(global-set-key (kbd "C-c i") 'open-index-file)
```

undef a key

```
(add-hook 'org-mode-hook
          '(lambda ()
             ;; Undefine C-c [ and C-c ] since this breaks my
             ;; org-agenda files when directories are include It
             ;; expands the files in the directories individually
             (org-defkey org-mode-map "\C-c[" 'undefined))
          'append)
```

10. Org-ref

```
(use-package org-ref
  :ensure t)
```

Define format for bibtex entries

```
;; variables that control bibtex key format for auto-generation
;; I want firstauthor-year-title-words
;; this usually makes a legitimate filename to store pdfs under.
(setq bibtex-autokey-year-length 4
      bibtex-autokey-name-year-separator "-"
      bibtex-autokey-year-title-separator "-"
      bibtex-autokey-titleword-separator "-"
      bibtex-autokey-titlewords 2
      bibtex-autokey-titlewords-stretch 1
      bibtex-autokey-titleword-length 5)
```

Where are the refs?

```
(setq reftex-default-bibliography '("~/Dropbox/work/bibliography/
   references.bib"))

;; see org-ref for use of these variables
(setq org-ref-bibliography-notes "~/Dropbox/work/bibliography/notes.org"
      org-ref-default-bibliography '("~/Dropbox/work/bibliography/
         references.bib")
      org-ref-pdf-directory "~/Dropbox/work/bibliography/bibtex-pdfs/")
```

11. Latex templates

Latex templates

```
(setq org-latex-listings t)

;;(setq org-latex-to-pdf-process '("xelatex %f && bibtex %f && xelatex %f
    && xelatex %f"))
(defun sk-latexmk-cmd (backend)
  "When exporting from .org with latex, automatically run latex,
     pdflatex, or xelatex as appropriate, using latexmk."
  (when (org-export-derived-backend-p backend 'latex)
    (let ((texcmd)))
    ;; default command: xelatex
    (setq texcmd "jobname=$(basename %f | sed 's/\.tex//');latexmk -
       xelatex -shell-escape -quiet %f && mkdir -p latex.d && mv ${
       jobname}.* latex.d/. && mv latex.d/${jobname}.{org,pdf,fdb_latexmk
       ,aux} .")
    ;; pdflatex -> .pdf
    (if (string-match "LATEX_CMD: pdflatex" (buffer-string))
        (setq texcmd "latexmk -pdflatex='pdflatex -interaction
           nonstopmode' -pdf -bibtex %f"))
    ;; xelatex -> .pdf
    (if (string-match "LATEX_CMD: xelatex" (buffer-string))
        (setq texcmd "latexmk -pdflatex='xelatex -interaction nonstopmode
           ' -pdf -bibtex -f  %f"))
    ;; LaTeX compilation command
    (setq org-latex-pdf-process (list texcmd))))

(org-add-hook 'org-export-before-processing-hook 'sk-latexmk-cmd)

(unless (boundp 'org-latex-classes)
  (setq org-latex-classes nil))
```

12. NHMRC project grant

```
(add-to-list 'org-latex-classes
             '("NHMRC_project_grant"
               "\\documentclass[12pt,table,names]{article}
\\usepackage{\\string~\"/.emacs.d/latex_templates/NHMRC_grant\"}
[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}")))
```

13. ARC Discovery Grant

```
(add-to-list 'org-latex-classes
             '("ARC_discovery_grant"
               "\\documentclass[12pt]{article}
\\usepackage{\\string~\"/.emacs.d/latex_templates/ARC_discovery\"}
[NO-DEFAULT-PACKAGES]
[NO-PACKAGES]"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")))
```

14. Internal report

```
(add-to-list 'org-latex-classes
             '("report"
               "\\documentclass[12pt]{article}
\\usepackage{\\string~\"/.emacs.d/latex_templates/report\"}
[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}")))
```

15. Simple presentation

```
(add-to-list 'org-latex-classes
             '("simplepresentation"
               "\\documentclass[aspectratio=169,18pt,t]{beamer}
\\usepackage{\\string~\"/.emacs.d/latex_templates/simple\"}
[NO-DEFAULT-PACKAGES]
[NO-PACKAGES]"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\begin{frame}[fragile]\\frametitle{%s}"
                "\\end{frame}"
                "\\begin{frame}[fragile]\\frametitle{%s}"
                "\\end{frame}")))
```

end.

Writing

1. Flyspell config

Installing aspell on linux:

```
apt install aspell aspell-en
```

on mac:

```
brew install aspell
```

Note in the config below I assume aspell is installed in `/usr/bin/` !.

```
(use-package flyspell
  :ensure t
  :diminish flyspell-mode
  :init
  (add-hook 'prog-mode-hook 'flyspell-prog-mode)

  (dolist (hook '(text-mode-hook org-mode-hook))
    (add-hook hook (lambda () (flyspell-mode 1))))

  (dolist (hook '(change-log-mode-hook log-edit-mode-hook
     org-agenda-mode-hook))
    (add-hook hook (lambda () (flyspell-mode -1))))

  :config
  (setq ispell-program-name "/usr/bin/aspell"
        ispell-local-dictionary "en_US"
        ispell-dictionary "american" ; better for aspell
        ispell-extra-args '("--sug-mode=ultra" "--lang=en_US")
        ispell-list-command "--list"
        ispell-local-dictionary-alist '(("en_US" "[[:alpha:]]" "[^[:alpha
           :]]" "‘’[']"
                                         t ; Many other characters
                                         ("-d" "en_US") nil utf-8))))
```

There is more stuff in Howard Abram's config but I'll leave this for now..

2. Writegood mode

This does not work - there is a wring gpg signature in melpa…

```
(when (file-exists-p "/home/user/programs/writegood-mode")
(message "Loading writegood-mode")
(add-to-list 'load-path "/home/user/programs/writegood-mode")
(require 'writegood-mode)
    (add-hook 'text-mode-hook 'writegood-mode)
     (add-hook 'org-mode-hook 'writegood-mode)
)
```

this mode will improve various aspects of writing.
end.

3. LangTool

I added the Emacs-langtool code from:
https://github.com/mhayashi1120/Emacs-langtool
To my /elisp/ directory.
To install langtool install maven package, java 8 then:

```
cd ~/programs
git clone https://github.com/languagetool-org/languagetool.git
./build.sh languagetool-standalone package
```

This does not work!
I now simply download the pre-compiles zip package…
To load:

```
(require 'langtool)
(setq langtool-language-tool-jar "/home/user/programs/langtool/
   LanguageTool-4.0/languagetool-commandline.jar")
```

Programming

General programming settings..

1. General

I like shallow indentation, but tabs are displayed as 8 characters by default. This reduces that.

```
(setq-default tab-width 2)
```

Treating terms in CamelCase symbols as separate words makes editing a little easier for me, so I like to use subword-mode everywhere.

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

Compilation output goes to the **compilation** buffer. I rarely have that window selected, so the compilation output disappears past the bottom of the window. This automatically scrolls the compilation window so I can always see the output.

```
 ;;(setq compilation-scroll-output t)
 (setq compile-command "make")
 (setq compilation-scroll-output 'first-error)
 (setq compilation-always-kill t)
 (setq compilation-disable-input t)
(add-hook 'compilation-mode-hook 'visual-line-mode)
```

2. Auto Complete

```
(use-package auto-complete
  :ensure t
  :commands auto-complete-mode
  :init
  (progn
    (auto-complete-mode t))
  :config
  (progn
    (use-package auto-complete-config)

    (ac-set-trigger-key "TAB")
    (ac-config-default)

    (setq ac-delay 0.02)
    (setq ac-use-menu-map t)
    (setq ac-menu-height 50)
    (setq ac-use-quick-help nil)
    (setq ac-comphist-file  "~/.emacs.d/ac-comphist.dat")
    (setq ac-ignore-case nil)
    (setq ac-dwim  t)
    (setq ac-fuzzy-enable t)

    (use-package ac-dabbrev
      :ensure t
      :config
      (progn
        (add-to-list 'ac-sources 'ac-source-dabbrev)))

    (setq ac-modes '(emacs-lisp-mode
                     lisp-mode
                     lisp-interaction-mode
                     c-mode
                     cc-mode
                     c++-mode
                     makefile-mode
                     sh-mode))))


(define-key ac-complete-mode-map "\C-n" 'ac-next)
(define-key ac-complete-mode-map "\C-p" 'ac-previous)
```

Headers

```
(use-package auto-complete-c-headers
  :ensure t)
(require 'auto-complete-config)
(ac-config-default)
                                        ; start yasnippet with emacs
                                        ; let's define a function which
                                           initializes
                                           auto-complete-c-headers and
                                           gets called for c/c++ hooks
(defun my:ac-c-header-init ()
  (require 'auto-complete-c-headers)
  (add-to-list 'ac-sources 'ac-source-c-headers)
  (add-to-list 'achead:include-directories '"/usr/include")
  )
(add-hook 'c-mode-hook 'my:ac-c-header-init)
(semantic-mode 1)
                                        ; let's define a function which
                                           adds semantic as a suggestion
                                           backend to auto complete
                                        ; and hook this function to
                                           c-mode-common-hook
(defun my:add-semantic-to-autocomplete()
  (add-to-list 'ac-sources 'ac-source-semantic)
  )
(add-hook 'c-mode-common-hook 'my:add-semantic-to-autocomplete)
                                        ; turn on automatic reparsing of
                                           open buffers in semantic
(global-semantic-idle-scheduler-mode 1)

(global-semanticdb-minor-mode 1)
```

```
(setq-local imenu-create-index-function #'ggtags-build-imenu-index)
```

3. Yasnippet

```
(global-set-key (kbd "M-/") 'hippie-expand)

(setq hippie-expand-try-functions-list
      '(try-expand-dabbrev
        try-expand-dabbrev-all-buffers
        try-expand-dabbrev-from-kill
        try-complete-file-name-partially
        try-complete-file-name
        try-expand-all-abbrevs
        try-expand-list
```

```
        try-expand-line))
```

Yasnippet

```
(use-package yasnippet
  :ensure t
  :init
  (yas-global-mode 1)
  :config
  (add-to-list 'yas-snippet-dirs (tl/emacs-subdirectory "snippets")))
```

4. Comments

```
(use-package smart-comment
  :ensure t
  :bind ("M-;" . smart-comment))
```

5. Tags:

```
(use-package etags
  :init (setq tags-revert-without-query 1))
```

```
(use-package ctags-update
  :ensure t
  :config
  (add-hook 'prog-mode-hook  'turn-on-ctags-auto-update-mode)
  :diminish ctags-auto-update-mode)
```

6. C

ggtags

```
(use-package ggtags
  :ensure t
  :commands (
             ggtags-mode
             ggtags-find-reference
             ggtags-idutils-query
             pop-tag-mark
             ) ; end :commands
  :init
  (add-hook 'c-mode-common-hook
            (lambda ()
              (when (derived-mode-p 'c-mode)
                (ggtags-mode 1))))
  :config

                                        ; This must be set to the
                                          location of gtags (global)
```

```
  ;;(setq ggtags-executable-directory "~/global-6.5.6/bin/")
                                        ; Allow very large database files
  (setq ggtags-oversize-limit 104857600)
  (setq ggtags-sort-by-nearness t)
  (setq ggtags-use-idutils t)
  (setq ggtags-use-project-gtagsconf nil)

  :bind (
         ("M-," . pop-tag-mark)
         ;; ("M-/" . ggtags-find-reference)
         ("M-]" . ggtags-idutils-query)

         :map ggtags-navigation-map
                                        ;Ergo
         ("M-u" . ggtags-navigation-previous-file)
         ("M-o" . ggtags-navigation-next-file)
         ("M-l" . ggtags-navigation-visible-mode)
         ("M-j" . ggtags-navigation-visible-mode)
         ("M-k" . next-error)
         ("M-i" . previous-error)
         ) ; end :bind
  )
```

Smartparens

```
(use-package smartparens
  :ensure t
  :init (add-hook 'C-mode-hook 'smartparens-mode))
```

Autocomplete

(use-package auto-complete :ensure t) (use-package auto-complete-c-headers :ensure t) ; do default config for auto-complete (require ‘auto-complete-config) (ac-config-default) ; start yasnippet with emacs ; let’s define a function which initializes auto-complete-c-headers and gets called for c/c++ hooks (defun my:ac-c-header-init () (require ‘auto-complete-c-headers) (add-to-list ‘ac-sources ‘ac-source-c-headers) (add-to-list ‘achead:include-directories ””/usr/include”)) (add-hook ‘c-mode-hook ‘my:ac-c-header-init)

; turn on Semantic (semantic-mode 1) ; let’s define a function which adds semantic as a suggestion backend to auto complete ; and hook this function to c-mode-common-hook (defun my:add-semantic-to-autocomplete() (add-to-list ‘ac-sources ‘ac-source-semantic)) (add-hook ‘c-mode-common-hook ‘my:add-semantic-to-autocomplete) ; turn on automatic reparsing of open buffers in semantic (global-semantic-idle-scheduler-mode 1)

(global-semanticdb-minor-mode 1)

SET BSD indent style

```
(add-hook 'c-mode-hook
```

```
          '(lambda()
             (c-set-style "bsd")
             ))
```

Don't ask with make command to run…

```
(setq compilation-read-command nil)
```

7. ESS (emacs speaks statistics…).

```
(use-package ess
  :ensure t
  :init (require 'ess-site))
```

Email

```
(require 'starttls)
(setq starttls-use-gnutls t)

(require 'smtpmail)
(setq send-mail-function  'smtpmail-send-it
      message-send-mail-function    'smtpmail-send-it
      starttls-use-gnutls t
      smtpmail-starttls-credentials  '(("smtp.office365.com" 587 nil nil)
         )
      smtpmail-auth-credentials (expand-file-name "~/.authinfo.gpg")
      smtpmail-smtp-server  "smtp.office365.com"
      smtpmail-stream-type  'starttls
      smtpmail-smtp-service 587)
```

1. Mu4e

```
     (cond
      ((string-equal system-type "windows-nt") ; Microsoft Windows
       (progn
         (message "Microsoft Windows")))
      ((string-equal system-type "darwin") ; Mac OS X
       (progn
         (add-to-list 'load-path "/usr/local/Cellar/mu/HEAD-0d9d8d2_1/
            share/emacs/site-lisp/mu/mu4e")
         (setq mu4e-mu-binary "/usr/local/bin/mu")
         ))
      ((string-equal system-type "gnu/linux") ; linux
       (progn
         (add-to-list 'load-path "~/programs/mu/mu4e")
         (setq mu4e-mu-binary "~/programs/mu/mu/mu")
```

```
    )))

;;  (add-to-list 'load-path "~/programs/mu/mu4e")

;;          (add-to-list 'load-path "/usr/local/share/emacs/site-lisp
   /mu/mu4e")

;; the modules
;;(if (file-exists-p mu4e-mu-binary)
;;    (message "Loading Mu4e...")


  (if (not (require 'mu4e nil t))
  (message "`mu4e' not found")

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


  (message "Loading Mu4e...")

  (setq mu4e-maildir (expand-file-name "~/Maildir"))

  (setq mu4e-sent-folder "/office365/sent")
  (setq mu4e-drafts-folder "/drafts")
  (setq mu4e-refile-folder "/office365/Archive")   ;; saved messages
  (setq mu4e-trash-folder "/office365/trash")


  (setq message-kill-buffer-on-exit t)
  (setq mu4e-change-filenames-when-moving t)
  (setq mu4e-confirm-quit nil)
  (setq mail-user-agent 'mu4e-user-agent)

  (setq mu4e-sent-messages-behavior 'sent)

  (setq mu4e-view-show-addresses t)

  (setq mu4e-attachment-dir "~/Downloads")


  (define-key mu4e-headers-mode-map (kbd "C-c c") '
     org-mu4e-store-and-capture)
```

```
       (define-key mu4e-view-mode-map (kbd "C-c c") '
          org-mu4e-store-and-capture)

       (setq mu4e-get-mail-command "~/programs/offlineimap/offlineimap.py
          ")

       (setq mu4e-compose-signature
                           "Associate Professor Timo Lassmann
Feilman Fellow
Academic Head of Computational Biology, Telethon Kids Institute
Adjunct Associate Professor, Center for Child Health Research
University of Western Australia

Telethon Kids Institute
100 Roberts Road, Subiaco, Western Australia, 6008
PO Box 855, West Perth, Western Australia, 6872

https://scholar.google.com.au/citations?user=7fZs_tEAAAAJ&hl=en

Visiting Scientist, RIKEN Yokohama Institute, Japan
Division of Genomic Technology,
RIKEN Center for Life Science Technologies,
Yokohama Institute,1-7-22 Suehiro-cho,
Tsurumi-ku, Yokohama, 230-0045 JAPAN")
)
```

Spell check

```
(add-hook 'mu4e-compose-mode-hook
          'flyspell-mode)
(add-hook 'message-mode-hook 'turn-on-orgtbl)
(add-hook 'message-mode-hook 'turn-on-orgstruct++)
```

TRAMP

```
(tramp-set-completion-function "ssh"
                               '((tramp-parse-sconfig "~/.ssh/config")))

(setq tramp-default-method "ssh")
```

Autoinsert templates

Again from Howards Abrams:

```
(use-package autoinsert
  :ensure t
```

```
  :init
  (setq auto-insert-directory (tl/emacs-subdirectory "templates/"))
  ;; Don't want to be prompted before insertion:
  (setq auto-insert-query nil)

  (add-hook 'find-file-hook 'auto-insert)
  (auto-insert-mode 1))
```

Use yes snippet for templates.

```
(defun tl/autoinsert-yas-expand()
  "Replace text in yasnippet template."
  (yas-expand-snippet (buffer-string) (point-min) (point-max)))
```

Set templates

```
(use-package autoinsert
 :config
  (define-auto-insert "\\.org$" ["default-orgmode.org" tl/
     autoinsert-yas-expand]))
```

Errors

Sometimes there is a problem with org “org-header function is void..
In this case delete the *elc files in the org dir:

```
cd .emacs.d/elpa/org-ZXXX/


rm -rf *elc
```

End.

End

```
    ;;(if (window-system)
    ;;(require 'init-client)
   ;; (if (daemonp)
     ;;   (add-hook 'after-make-frame-functions
     ;;             (lambda (frame)
    ;;               (tl/change-theme 'gotham 'org-src-color-blocks-dark
       )))
    ;;  (tl/change-theme 'gotham 'org-src-color-blocks-dark))
    ;;(require 'init-client)
    ;;((message "nada")))



(if (daemonp)
```

```
(add-hook 'after-make-frame-functions
          '(lambda (f)
             (with-selected-frame f
               (when (window-system f) (require 'init-client) ))))
(require 'init-client) )
```

```
(require 'init-local nil t)
```

```
(provide 'config-main)
```

All good.

```
;; == irony-mode ==
(use-package irony
  :ensure t
  :defer t
  :init
  (add-hook 'c++-mode-hook 'irony-mode)
  (add-hook 'c-mode-hook 'irony-mode)
  (add-hook 'objc-mode-hook 'irony-mode)
  :config
  ;; replace the `completion-at-point' and `complete-symbol' bindings in
  ;; irony-mode's buffers by irony-mode's function
  (defun my-irony-mode-hook ()
    (define-key irony-mode-map [remap completion-at-point]
      'irony-completion-at-point-async)
    (define-key irony-mode-map [remap complete-symbol]
      'irony-completion-at-point-async))
  (add-hook 'irony-mode-hook 'my-irony-mode-hook)
  (add-hook 'irony-mode-hook 'irony-cdb-autosetup-compile-options)
  )

;; == company-mode ==
(use-package company
  :ensure t
  :defer t
  :init (add-hook 'after-init-hook 'global-company-mode)
  :config
  (use-package company-irony :ensure t :defer t)
  (setq company-idle-delay              nil
  company-minimum-prefix-length   2
  company-show-numbers            t
  company-tooltip-limit           20
```

```
company-dabbrev-downcase        nil
company-backends                '((company-irony company-gtags))
)
:bind ("C-;" . company-complete-common)
)
```

This itt