Contents

1 Why Emacs?

I started using Emacs in 2010 and my relationship with it has been on and off. I mostly use Jetbrains IntelliJ IDEA for my work. But the idea of Emacs - everything inside is awesome. You can configure IDEA to an extent but after that it just gives up. Checking emails, managing schedule, and taking notes - few things that are lacking in IDEA. Emacs on the other hand is an Operating System in itself. You can configure it the way you want and for your comfort and there is a huge community to help, guide and support you.

2 About

This is my personal Emacs configuration. Over past few years, I had been using Emacs configuration by others. I used to modify few things here and there in my old configuration but not before spending several hours and

without understanding what is going on. So, after reading `r/emacs` discussions, I decided that I will get rid of all the Emacs configurations and start it at the very beginning. I have not used Emacs for any coding purposes. I code in Java and IntelliJ IDEA is the best IDE for that. Therefore, I decided that in the beginning, I do not need many configurations . Since the discussion on the `r/emacs` was all about Literate Programming and how `org-mode` facilitates that, I decided to start with setting up org-mode first

3 Configurations

3.1 About this file

3.1.1 Org File Tweaks

There are a few tweaks included in this org file that make it a little easier to work with.

1. Automatically Tangle First there is a property defined on the file:

   ```
   header-args :tangle yes
   ```

   This tells emacs to automatically tangle (include) all code blocks in this file when generating the code for the config, unless the code block explicitly includes `:tangle no` as the above code block does.

2. Visibility Settings Next we have a property that defines the visibility for org to show it's direct children on startup. This way a clean outline of all sub headings under Configuration is shown each time this file is opened in org-mode.

   ```
   :PROPERTIES:
   :VISIBILITY: children
   :END:
   ```

3. Table of Contents Finally, there is a Table of Contents heading that includes the tag: `:TOC_5_gh:`. This tells an org-mode package `toc-org` to generate a table of contents under this heading that has a max depth of 3 and is created using Github-style hrefs. This table of contents is updated everytime the file is saved and makes for a functional table of contents that works property directly on github.

4. Org Babel

   This file is loaded using *org-babel*. Create *init.el* file in *.emacs.d* folder and put following code

```
(org-babel-load-file "~/.emacs.d/systemhalted.org")
```

3.2 Global Settings

3.2.1 Load Customizations from folder

I put some scripts in *customizations* folder. Lets load it.

```
(add-to-list 'load-path "~/.emacs.d/customizations")
```

3.2.2 Setup Proxy

I use Emacs on my work laptop and there is a firewall that I need to pass through. Proxy settings allow me to do that. Save the below code under `customizations` and call it `setup-proxy.el`

```
;; HttpProxy
(setq url-proxy-services
      '(("no_proxy" . "^\\(localhost\\|10.*\\)")
("http" . "proxy:port")
("https" . "proxy:prt")))

(setq url-http-proxy-basic-auth-storage
      (list (list "proxy:port"
  (cons "Input your LDAP UID !"
(base64-encode-string "UserName:Password")))))
```

3.2.3 Package Configuration

Define package repositories

```
(require 'package)
(setq-default
 load-prefer-newer t
 package-enable-at-startup nil)
(add-to-list 'package-archives
     '("gnu" . "https://elpa.gnu.org/packages/") t)
```

```
(add-to-list 'package-archives
     '("marmalade" . "http://marmalade-repo.org/packages/") t)
(add-to-list 'package-archives
     '("tromey" . "http://tromey.com/elpa/") t)
(add-to-list 'package-archives
     '("org" . "http://orgmode.org/elpa/") t)
(add-to-list 'package-archives
     '("melpa" . "http://melpa.org/packages/") t)
(add-to-list 'package-archives
     '("melpa-stable" . "http://stable.melpa.org/packages/") t)
(package-initialize)
```

3.2.4 Install and Configure use-package

We are going to use `use-package` for most of our setup. It is easy to use it. In case you need a tutorial on few of its keywords, read this.

```
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package t))
(setq-default use-package-always-defer t
      use-package-always-ensure t)
```

3.3 UI Customization

Some of these settings were copied from Sergei Nosov's configurations.

3.3.1 Lean and mean

Emacs doesn't need a lot of UI elements - it should be lean and mean. Well, and clean.

1. Disable startup/splash screen

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

2. Setup initial major mode to Org-mode

   ```
   (setq-default initial-major-mode (quote org-mode))
   ```

3. Remove scratch message

```
(setq-default initial-scratch-message nil)
```

4. Disable Unnecessary Interface

```
(menu-bar-mode -1)
(tool-bar-mode -1)
(unless (and (display-graphic-p) (eq system-type 'darwin))
  (push '(menu-bar-lines . 0) default-frame-alist))
(push '(tool-bar-lines . 0) default-frame-alist)
(push '(vertical-scroll-bars) default-frame-alist)
```

5. Reduce the delay echoing the keystrokes When you press C-x, for example, and hesitate with a next character, C-x will be displayed in the echo-area after some time. But I don't see any reason why you should wait for it.

```
(setq echo-keystrokes 0.00111)
```

6. Join following line

```
(define-key global-map (kbd "C-c j")
  (defun systemhalted/join-following-line (arg)
    "Joins the following line or the whole selected region"
    (interactive "P")
    (if (use-region-p)
(let ((fill-column (point-max)))
  (fill-region (region-beginning) (region-end)))
      (join-line -1))))
```

7. Full Screen

```
(toggle-frame-fullscreen)
(add-to-list 'default-frame-alist '(fullscreen . fullboth))
;;(add-hook 'window-setup-hook 'toggle-frame-maximized t).
```

3.3.2 PDF

```
(setq doc-view-continuous t)
```

3.3.3 Theme

```
(use-package ample-theme
  :init (progn (load-theme 'ample t t)
       (load-theme 'ample-flat t t)
       (load-theme 'ample-light t t)
       (enable-theme 'ample-light))
  :defer t
  :ensure t)

(use-package spacemacs-common
  :ensure spacemacs-theme
  :config (load-theme 'spacemacs-dark t))

(use-package leuven-theme
  :config (load-theme 'leuven t))

(add-to-list 'load-path "~/.emacs.d/elegant-emacs")
(require 'elegance)
(require 'sanity)
```

3.3.4 Modeline Time and Battery

```
(display-time-mode 1)
(display-battery-mode 1)
```

3.3.5 Spaceline

```
(use-package spaceline :ensure t
  :config
  (use-package spaceline-config
    :config
    (spaceline-toggle-minor-modes-off)
    (spaceline-toggle-buffer-encoding-off)
    (spaceline-toggle-buffer-encoding-abbrev-off)
    (setq powerline-default-separator 'rounded)
    (setq spaceline-highlight-face-func 'spaceline-highlight-face-evil-state)
    (spaceline-define-segment line-column
      "The current line and column numbers."
      "l:%l c:%2c")
    (spaceline-define-segment time
```

```
    "The current time."
    (format-time-string "%H:%M"))
  (spaceline-define-segment date
    "The current date."
    (format-time-string "%h %d"))
  (spaceline-toggle-time-on)
  (spaceline-emacs-theme 'date 'time)))
```

3.3.6 Cursor Position

```
(setq line-number-mode t)
(setq column-number-mode t)
```

3.4 Text Manipulation

3.4.1 Move text

Most of the time, I need to move a the text up an down a bit. There is a *transpose-line* command that maps to *C-x C-t*, which is cumbersome and most of the time it messes-up with my flow. So, here we will map it to *M-n* and *M-p* following the convention of movement keys. Note: If you need to move the text to some pretty distant place, then, of course, it's easier to kill and yank it.

```
(eval-after-load "move-text-autoloads"
  '(progn
     (if (require 'move-text nil t)
 (progn
   (define-key global-map (kbd "M-n") 'move-text-down)
   (define-key global-map (kbd "M-p") 'move-text-up))
       (message "WARNING: move-text not found"))))
```

3.4.2 Duplicate the current line

Equivalent of Ctrl+d (Command+d on Mac) in IntelliJ IDEA Source: `https://www.emacswiki.org/emacs/CopyingWholeLines#toc12`

```
(define-key global-map (kbd "C-c k")
  (defun systemhalted/duplicate-line-or-region (&optional n)
    "Duplicate current line, or region if active.
    With argument N, make N copies.
```

```
    With negative N, comment out original line and use the absolute value."
    (interactive "*p")
    (let ((use-region (use-region-p)))
      (save-excursion
(let ((text (if use-region        ;Get region if active, otherwise line
(buffer-substring (region-beginning) (region-end))
      (prog1 (thing-at-point 'line)
(end-of-line)
(if (< 0 (forward-line 1)) ;Go to beginning of next line, or make a new one
    (newline))))))
  (dotimes (i (abs (or n 1)))     ;Insert N times, or once if not specified
    (insert text))))
      (if use-region nil                  ;Only if we're working with a line (not a reg
(let ((pos (- (point) (line-beginning-position)))) ;Save column
  (if (> 0 n)                             ;Comment out original with negative arg
      (comment-region (line-beginning-position) (line-end-position)))
  (forward-line 1)
  (forward-char pos))))))
```

3.4.3 String manipulations

Emacs 24.4 came with a subr-x library with routines for string manipulations, like string-trim, string-join and etc. It's better to always have these at hand.

```
(require 'subr-x nil t)
```

3.5 Shortcuts, Longcuts, Miscellaneous Configs

3.5.1 Clipboard. Copy from terminal emacs to the X clipboard.

```
(use-package xclip
  :ensure t
  :config
  (xclip-mode 1))
```

3.5.2 Simplify Yes/No Prompts

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

3.5.3 But make it hard to accidentally exit

```
(setq-default confirm-kill-emacs (quote y-or-n-p))
```

3.5.4 Make finding file easy

```
(global-set-key (kbd "C-x f")    'find-file)
```

3.5.5 UTF-8 Coding System

Use UTF-8 as much as possible

```
(set-language-environment 'utf-8)
(setq locale-coding-system 'utf-8)

;; set the default encoding system
(prefer-coding-system 'utf-8)
(setq default-file-name-coding-system 'utf-8)
(set-default-coding-systems 'utf-8)
(set-terminal-coding-system 'utf-8)
(set-keyboard-coding-system 'utf-8)

;; Treat clipboard input as UTF-8 string first; compound text next, etc.
(setq x-select-request-type '(UTF8_STRING COMPOUND_TEXT TEXT STRING))
```

3.5.6 Shut up the bell

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

3.5.7 Disabled Commands

Change nil to t to disable the command. Note: currently not using it. But this is the way to do it

```
(put 'upcase-region 'disabled nil)
```

3.5.8 Always kill the buffer

```
(defun kill-current-buffer ()
  "Kills the current buffer."
  (interactive)
  (kill-buffer (current-buffer)))
(global-set-key (kbd "C-x k") 'kill-current-buffer)
```

3.5.9 Visit systemhalted.org

```
(defun config-visit ()
  (interactive)
  (find-file "~/.emacs.d/systemhalted.org"))
(global-set-key (kbd "C-c e") 'config-visit)
```

3.5.10 Reload Config

```
(defun config-reload ()
  "Reloads ~/.emacs.d/systemhalted.org at runtime"
  (interactive)
  (org-babel-load-file (expand-file-name "~/.emacs.d/systemhalted.org")))
(global-set-key (kbd "C-c r") 'config-reload)
```

3.5.11 Electric

```
(setq electric-pair-pairs '(
    (?\{ . ?\})
    (?\( . ?\))
    (?\[ . ?\])
    (?\" . ?\")
    ))

(electric-pair-mode t)
```

3.5.12 Show Parens

```
(show-paren-mode 1)
```

3.5.13 Exile the backup files

Backup files are insanely irritating if you expect clean ls output and don't want to filter out irrelevant junk. The right thing is to exile them to a dedicated directory:

Reference: Somehwhere on Reddit (find the post and link here)

```
(setq backup-by-copying t
      backup-directory-alist `(("." . ,(concat user-emacs-directory "backups")))
      tramp-backup-directory-alist backup-directory-alist
      delete-old-versions t
      kept-new-versions 3
```

```
      kept-old-versions 2
      version-control t
      vc-cvs-stay-local nil)
```

3.5.14 PDFLatex

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

3.6 Packages

3.6.1 Which Key?

```
(use-package which-key
  :init
  (which-key-mode)
  :config
  (which-key-setup-side-window-bottom)
  (setq which-key-sort-order 'which-key-key-order-alpha
which-key-side-window-max-width 0.33
which-key-idle-delay 0.05)
  :diminish which-key-mode)
```

3.6.2 Helm

```
(use-package helm
  :ensure t
  :bind
  ("C-x C-f" . 'helm-find-files)
  ("C-x C-b" . 'helm-buffers-list)
  ("M-x" . 'helm-M-x)
  :config
  (defun systemhalted/helm-hide-minibuffer ()
    (when (with-helm-buffer helm-echo-input-in-header-line)
      (let ((ov (make-overlay (point-min) (point-max) nil nil t)))
(overlay-put ov 'window (selected-window))
(overlay-put ov 'face
     (let ((bg-color (face-background 'default nil)))
       `(:background ,bg-color :foreground ,bg-color)))
(setq-local cursor-type nil))))
```

```
  (add-hook 'helm-minibuffer-set-up-hook 'systemhalted/helm-hide-minibuffer)
  (setq helm-autoresize-max-height 0
helm-autoresize-min-height 40
helm-M-x-fuzzy-match t
helm-buffers-fuzzy-matching t
helm-recentf-fuzzy-match t
helm-semantic-fuzzy-match t
helm-imenu-fuzzy-match t
helm-split-window-in-side-p nil
helm-move-to-line-cycle-in-source nil
helm-ff-search-library-in-sexp t
helm-scroll-amount 8
helm-echo-input-in-header-line t)
  :init
  (helm-mode 1))

(require 'helm-config)
(helm-autoresize-mode 1)
(define-key helm-find-files-map (kbd "C-b") 'helm-find-files-up-one-level)
(define-key helm-find-files-map (kbd "C-f") 'helm-execute-persistent-action)
```

3.6.3 SMEX

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

3.6.4 iBuffer

Before iPhone, there was iBuffer

```
(global-set-key (kbd "C-x b") 'ibuffer)
(setq ibuffer-expert t)
```

3.6.5 Ivy

```
(use-package ivy
  :demand t)
```

3.6.6 Magit

The magical git client. Let's load magit only when one of the several entry pont functions we invoke regularly outside of magit is called.

```
(use-package magit
  :commands (magit-status magit-blame magit-log-buffer-file magit-log-all))
```

3.6.7 Projectile

Projectile is a quick and easy project management package that "just works". We're going to install it and make sure it's loaded immediately.

```
(use-package projectile
  :ensure t
  :bind-keymap
  ("C-c p" . projectile-command-map)
  :config
  (projectile-mode +1))
```

3.6.8 Org

1. Org Agenda and Todo setup Let's include a newer version of org-mode than the one that is built in. We're going to manually remove the org directories from the load path, to ensure the version we want is prioritized instead.

   ```
   (use-package org
     :ensure org-plus-contrib
     :pin org
     :defer t
     :config (setq org-log-done 'time
   org-log-done 'note
   org-agenda-files (list "~/org/inbox.org"
          "~/org/gtd.org"
          "~/org/tickler.org"
          "~/org/references.org")
   org-capture-templates '(("t" "Todo [inbox]" entry
    (file+headline "~/org/inbox.org" "Tasks")
   ```

```
 "* TODO %i%?")
("T" "Tickler" entry
 (file+headline "~/org/tickler.org" "Tickler")
 "* %i%? \n %U"))
org-todo-keywords '((sequence "TODO(t)" "START(s)" "WAIT(w)" "|" "DONE(d)" "CANCEI
  :init
  (define-key global-map (kbd "C-c l") 'org-store-link)
  (define-key global-map (kbd "C-c a") 'org-agenda)
  (define-key global-map (kbd "C-c c") 'org-capture)
  )

(setq org-refile-targets '((org-agenda-files :maxlevel . 4)
   ("~/org/someday.org" :maxlevel . 1)
   ("~/org/archive.org" :maxlevel . 4)
   ))
```

2. Code editing in same window

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

3. Org Bullets Makes it all look a bit nicer, I hate looking at asterisks.

```
(use-package org-bullets
  :hook
  (( org-mode ) . org-bullets-mode))
```

4. Org Tempo

```
(require 'org-tempo)
```

5. Toc-org Let's install and load the `toc-org` package after org mode is loaded. This is the package that automatically generates an up to date table of contents for us.

```
(use-package toc-org
  :after org
  :init (add-hook 'org-mode-hook #'toc-org-enable))
```

6. Org-sidebar When I write, I need a map of the document or the table of content on the side. Org-sidebar helps with that:

```
(use-package org-sidebar
  :custom (org-sidebar-tree-side 'left))
```

7. Git Auto commit for Org files

   ```
   (use-package git-auto-commit-mode)
   ```

8. HTMLIZE Org-mode supports HTML export natively but syntax highlighting is added through htmlize.el.

   ```
   (use-package htmlize
     :config
     (setq org-src-fontify-natively t))
   ```

3.6.9 Nov Mode

I prefer reading EPUB books on Emacs. Nov Mode allows me do that

```
(use-package nov
  :demand t)

(add-to-list 'auto-mode-alist '("\\.epub\\'" . nov-mode))

;; set unzip
(setq nov-unzip-program "/usr/bin/unzip") ;;nov needs to know the location of unzip pa
```

3.6.10 define-word

Word and their meanings and what better way to have this information at point.

```
(use-package define-word
  :defer t
  :ensure t
  :init (global-set-key (kbd "C-c d") 'define-word-at-point)
  (global-set-key (kbd "C-c D") 'define-word))
```

3.6.11 Treemacs

```
(use-package treemacs
  :init
  (add-hook 'treemacs-mode-hook
    (lambda () (treemacs-resize-icons 15))))
```

3.6.12 IDLE-HIGHLIGHT

```
(use-package idle-highlight)
```

3.6.13 Font-lock

```
(require 'font-lock)
```

3.7 Custom Functions

1. SpeedTest

   ```
   (load "setup-speedtest.el")
   ```

3.8 Programming

3.8.1 yasnippet

```
(use-package yasnippet
  :ensure t
  :config
  (use-package yasnippet-snippets
    :ensure t)
  (yas-reload-all))
```

3.8.2 flycheck

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

3.8.3 company mode

I set the delay for company mode to kick in to half a second, I also make sure that it starts doing its magic after typing in only 2 characters.

```
(use-package company
  :ensure t
  :config
  (setq company-idle-delay 0)
  (setq company-minimum-prefix-length 3))

(with-eval-after-load 'company
  (define-key company-active-map (kbd "M-n") nil)
  (define-key company-active-map (kbd "M-p") nil)
  (define-key company-active-map (kbd "C-n") #'company-select-next)
  (define-key company-active-map (kbd "C-p") #'company-select-previous)
  (define-key company-active-map (kbd "SPC") #'company-abort))
```

3.8.4 Languages

1. C/C++

   ```
   (add-hook 'c++-mode-hook 'yas-minor-mode)
   (add-hook 'c-mode-hook 'yas-minor-mode)

   (use-package flycheck-clang-analyzer
     :ensure t
     :config
     (with-eval-after-load 'flycheck
       (require 'flycheck-clang-analyzer)
       (flycheck-clang-analyzer-setup)))

   (with-eval-after-load 'company
     (add-hook 'c++-mode-hook 'company-mode)
     (add-hook 'c-mode-hook 'company-mode))

   (use-package company-c-headers
     :ensure t)

   (use-package company-irony
     :ensure t
     :config
     (setq company-backends '((company-c-headers
       company-dabbrev-code
       company-irony))))
   ```

```
(use-package irony
  :ensure t
  :config
  (add-hook 'c++-mode-hook 'irony-mode)
  (add-hook 'c-mode-hook 'irony-mode)
  (add-hook 'irony-mode-hook 'irony-cdb-autosetup-compile-options))
```

2. Haskell

```
(use-package haskell-mode
  :defer t
  :init
  (progn
    (add-hook 'haskell-mode-hook #'haskell-indentation-mode)
    (add-hook 'haskell-mode-hook #'turn-on-haskell-doc-mode)
    (add-hook 'haskell-mode-hook #'subword-mode))
  :config
  (progn
    (let ((my-cabal-path (expand-file-name "~/.cabal/bin")))
      (setenv "PATH" (concat my-cabal-path ":" (getenv "PATH")))
      (add-to-list 'exec-path my-cabal-path))
    (custom-set-variables '(haskell-tags-on-save t))

    (custom-set-variables
     '(haskell-process-suggest-remove-import-lines t)
     '(haskell-process-auto-import-loaded-modules t)
     '(haskell-process-log t))
    (define-key haskell-mode-map (kbd "C-c C-l")
      'haskell-process-load-or-reload)
    (define-key haskell-mode-map (kbd "C-c C-z")


      (eval-after-load 'haskell-cabal
'(progn
   (define-key haskell-cabal-mode-map (kbd "C-c C-z")
     'haskell-interactive-switch)
   (define-key haskell-cabal-mode-map (kbd "C-c C-k")
     'haskell-interactive-mode-clear)
   (define-key haskell-cabal-mode-map (kbd "C-c C-c")
```

```
  'haskell-process-cabal-build)
(define-key haskell-cabal-mode-map (kbd "C-c c")
  'haskell-process-cabal)))

 (custom-set-variables '(haskell-process-type 'cabal-repl))

 (autoload 'ghc-init "ghc" nil t)
 (autoload 'ghc-debug "ghc" nil t)
 (add-hook 'haskell-mode-hook (lambda () (ghc-init)))))
```

3. Common Lisp/Slime

   Slime stands for Superior Lisp Interaction Mode for Emacs. For a quick intro, read this.

```
(use-package slime
  :ensure t
  :config
  (setq inferior-lisp-program "/usr/local/bin/sbcl"))

(slime-setup '(slime-fancy))
```

4. Easy-to-add emacs-lisp template Hitting tab after an "<el" in an org-mode file will create a template for elisp insertion.

```
(add-to-list 'org-structure-template-alist
     '("le" .  "#+BEGIN_SRC emacs-lisp\n \n#+END_SRC"))
```

3.9 Post Initialization

1. Garbage Collection Let's lower our GC thresholds back down to a sane level.

```
(setq gc-cons-threshold 16777216
      gc-cons-percentage 0.1)
```