Emacs configuration file

Lars Tveito

December 23, 2013

Contents

1 About

This is a Emacs configuration file written in `org-mode`. There are a few reasons why I wanted to do this. My `.emacs.d/` was a mess, and needed a proper clean-up. Also I like keeping all my configurations in a single file, using `org-mode` I can keep this file *organized*. I aim to briefly explain all my configurations.

2 Configurations

2.1 Meta

Emacs can only load `.el`-files. We can use `C-c C-v t` to run `org-babel-tangle`, which extracts the code blocks from the current file into a source-specific file (in this case a `.el`-file).

To avoid doing this each time a change is made we can add a function to the `after-save-hook` ensuring to always tangle and byte-compile the `org`-document after changes.

```
(defun init-hook ()
  "If the current buffer is 'init.org' the code-blocks are
tangled, and the tangled file is compiled."
  (when (equal (buffer-file-name)
               (concat user-emacs-directory "init.org"))
    (org-babel-tangle)
    (byte-compile-file (concat user-emacs-directory "init.el"))))

(add-hook 'after-save-hook 'init-hook)
```

2.2 Package

Managing extensions for Emacs is simplified using `package` which is built in to Emacs 24 and newer. To load downloaded packages we need to initialize `package`.

```
(package-initialize)
```

Packages can be fetched from different mirrors, melpa is the largest archive and is well maintained.

```
(add-to-list 'package-archives
             '("MELPA" . "http://melpa.milkbox.net/packages/") t)
```

Some packages I find useful are installed if missing.

```
(let ((packages
       '(ac-geiser         ; Auto-complete backend for geiser
         ac-slime          ; An auto-complete source using slime completions
         ace-jump-mode     ; quick cursor location minor mode
         auto-compile      ; automatically compile Emacs Lisp libraries
         auto-complete     ; auto completion
         elscreen          ; window session manager
         expand-region     ; Increase selected region by semantic units
         flx-ido           ; flx integration for ido
         ido-vertical-mode ; Makes ido-mode display vertically.
         geiser            ; GNU Emacs and Scheme talk to each other
         haskell-mode      ; A Haskell editing mode
         jedi              ; Python auto-completion for Emacs
         magit             ; control Git from Emacs
         markdown-mode     ; Emacs Major mode for Markdown-formatted files.
         monokai-theme     ; A fruity color theme for Emacs.
         move-text         ; Move current line or region with M-up or M-down
         multiple-cursors  ; Multiple cursors for Emacs.
         org               ; Outline-based notes management and organizer
         paredit           ; minor mode for editing parentheses
         pretty-lambdada   ; the word ‘lambda’ as the Greek letter.
         ;; slime          ; Superior Lisp Interaction Mode for Emacs
         smex              ; M-x interface with Ido-style fuzzy matching.
         )))
  ;; 'remove-if' is a part of the cl-library, so we require this feature.
  (require 'cl)
  ;; Filter out installed packages and install the remaining.
  (mapc 'package-install (remove-if 'package-installed-p packages)))
```

2.3 Require

Some features are not loaded by default to minimize initialization time, so they have to be required (or loaded, if you will).

```
(dolist (feature
         '(auto-compile             ; auto-compile .el files
           auto-complete-config     ; a configuration for auto-complete-mode
           jedi                     ; auto-completion for python
           pretty-lambdada          ; show 'lambda' as the greek letter.
           ox-latex                 ; the latex-exporter (from org)
           recentf                  ; recently opened files
           tex-mode                 ; TeX, LaTeX, and SliTeX mode commands
           ))
  (require feature))
```

2.4 Sane defaults

These are what *I* consider to be saner defaults.

We can set variables to whatever value we'd like using `setq`.

```
(setq initial-scratch-message nil     ; Clean scratch buffer.
      inhibit-startup-message t       ; No splash screen please.
      default-input-method "TeX"      ; Use TeX when toggeling input method.
      doc-view-continuous t           ; At page edge goto next/previous.
      echo-keystrokes 0.1             ; Show keystrokes asap.
      )
```

Some variables are buffer-local, so changing them using `setq` will only change them in a single buffer. Using `setq-default` we change the buffer-local variable's default value.

```
(setq-default fill-column 76                  ; Maximum line width.
              indent-tabs-mode nil            ; Use spaces instead of tabs.
              split-width-threshold 100       ; Split verticly by default.
              auto-fill-function 'do-auto-fill ; Auto-fill-mode everywhere.
              )
```

Answering *yes* and *no* to each question from Emacs can be tedious, a single *y* or *n* will suffice.

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

To avoid file system clutter we put all auto saved files in a single directory.

```
(defvar emacs-autosave-directory
  (concat user-emacs-directory "autosaves/")
  "This variable dictates where to put auto saves. It is set to a
  directory called autosaves located wherever your .emacs.d/ is
  located.")

;; Sets all files to be backed up and auto saved in a single directory.
(setq backup-directory-alist
      `((".*" . ,emacs-autosave-directory))
      auto-save-file-name-transforms
      `((".*" ,emacs-autosave-directory t)))
```

Set `utf-8` as preferred coding system.

```
(set-language-environment "UTF-8")
```

By default the `narrow-to-region` command is disabled and issues a warning, because it might confuse new users. I find it useful sometimes, and don't want to be warned.

```
(put 'narrow-to-region 'disabled nil)
```

Call `auto-complete` default configuration, which enables `auto-complete` globally.

```
(ac-config-default)
```

Automaticly revert `doc-view`-buffers when the file changes on disk.

```
(add-hook 'doc-view-mode-hook 'auto-revert-mode)
```

2.5 Modes

There are some modes that are enabled by default that I don't find particularly useful. We create a list of these modes, and disable all of these.

```
(dolist (mode
         '(tool-bar-mode                ; No toolbars, more room for text.
           scroll-bar-mode              ; No scroll bars either.
           blink-cursor-mode            ; The blinking cursor gets old.
```

```
          ))
  (funcall mode 0))
```

Let's apply the same technique for enabling modes that are disabled by default.

```
(dolist (mode
         '(abbrev-mode                ; E.g. sopl -> System.out.println.
           auto-compile-on-load-mode  ; Compile .el files on load ...
           auto-compile-on-save-mode  ; ... and save.
           column-number-mode         ; Show column number in mode line.
           delete-selection-mode      ; Replace selected text.
           recentf-mode               ; Recently opened files.
           show-paren-mode            ; Highlight matching parentheses.
           ))
  (funcall mode 1))
```

This makes `.md`-files open in `markdown-mode`.

```
(add-to-list 'auto-mode-alist '("\\.md\\'" . markdown-mode))
```

2.6 Visual

Change the color-theme to `monokai` (downloaded using `package`).

```
(load-theme 'monokai t)
```

Use the Inconsolata font if it's installed on the system.

```
(when (member "Inconsolata" (font-family-list))
  (set-face-attribute 'default nil :font "Inconsolata-13"))
```

2.7 Ido

Interactive do (or `ido-mode`) changes the way you switch buffers and open files/directories. Instead of writing complete file paths and buffer names you can write a part of it and select one from a list of possibilities. Using `ido-vertical-mode` changes the way possibilities are displayed, and `flx-ido-mode` enables fuzzy matching.

```
(dolist (mode
         '(ido-mode                   ; Interactivly do.
```

```
          ido-everywhere             ; Use Ido for all buffer/file reading.
          ido-vertical-mode          ; Makes ido-mode display vertically.
          flx-ido-mode               ; Toggle flx ido mode.
          ))
  (funcall mode 1))
```

We can set the order of file selections in `ido`. I prioritize source files along with `org`- and `tex`-files.

```
(setq ido-file-extensions-order
      '(".el" ".scm" ".lisp" ".java" ".c" ".h" ".org" ".tex"))
```

Sometimes when using `ido-switch-buffer` the `*Messages*` buffer get in the way, so we set it to be ignored (it can be accessed using `C-h e`, so there is really no need for it in the buffer list).

```
(add-to-list 'ido-ignore-buffers "*Messages*")
```

To make `M-x` behave more like `ido-mode` we can use the `smex` package. It needs to be initialized, and we can replace the binding to the standard `execute-extended-command` with `smex`.

```
(smex-initialize)
(global-set-key (kbd "M-x") 'smex)
```

2.8 Mail

```
(setq user-full-name         "Lars Tveito"
      user-mail-address      "larstvei@ifi.uio.no"
      smtpmail-smtp-server   "smtp.uio.no"
      smtpmail-smtp-service   587)
```

2.9 Calendar

Define a function to display week numbers in `calender-mode`. The snippet is from EmacsWiki.

```
(defun calendar-show-week (arg)
  "Displaying week number in calendar-mode."
  (interactive "P")
  (copy-face font-lock-constant-face 'calendar-iso-week-face)
  (set-face-attribute
```

```
  'calendar-iso-week-face nil :height 0.7)
 (setq calendar-intermonth-text
       (and arg
            '(propertize
              (format
               "%2d"
               (car (calendar-iso-from-absolute
                     (calendar-absolute-from-gregorian
                      (list month day year)))))
              'font-lock-face 'calendar-iso-week-face))))
```

Evaluate the `toggle-calendar-show-week` function.

```
(calendar-show-week t)
```

Set Monday as the first day of the week, and set my location.

```
(setq calendar-week-start-day 1
      calendar-latitude 60.0
      calendar-longitude 10.7
      calendar-location-name "Oslo, Norway")
```

2.10 Flyspell

Flyspell offers on-the-fly spell checking. We can enable flyspell for all text-modes with this snippet.

```
(add-hook 'text-mode-hook 'turn-on-flyspell)
```

To use flyspell for programming there is `flyspell-prog-mode`, that only enables spell checking for comments and strings. We can enable it for all programming modes using the `prog-mode-hook`. Flyspell interferes with auto-complete mode, but there is a workaround provided by auto complete.

```
(add-hook 'prog-mode-hook 'flyspell-prog-mode)
(ac-flyspell-workaround)
```

2.11 Org

I use `org-agenda` for appointments and such.

```
(setq org-agenda-start-on-weekday nil             ; Show agenda from today.
      org-agenda-files '("~/Dropbox/life.org")    ; A list of agenda files.
      org-agenda-default-appointment-duration 120 ; 2 hours appointments.
      )
```

When editing org-files with source-blocks, we want the source blocks to be themed as they would in their native mode.

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

2.12 Interactive functions

To search recent files useing `ido-mode` we add this snippet from EmacsWiki.

```
(defun recentf-ido-find-file ()
  "Find a recent file using Ido."
  (interactive)
  (let ((f (ido-completing-read "Choose recent file: " recentf-list nil t)))
    (when f
      (find-file f))))
```

`just-one-space` removes all whitespace around a point - giving it a negative argument it removes newlines as well. We wrap a interactive function around it to be able to bind it to a key.

```
(defun remove-whitespace-inbetween ()
  "Removes whitespace before and after the point."
  (interactive)
  (just-one-space -1))
```

This interactive function switches you to a `shell`, and if triggered in the shell it switches back to the previous buffer.

```
(defun switch-to-shell ()
  "Jumps to eshell or back."
  (interactive)
  (if (string= (buffer-name) "*shell*")
      (switch-to-prev-buffer)
    (shell)))
```

To duplicate either selected text or a line we define this interactive function.

```
(defun duplicate-thing ()
  "Ethier duplicates the line or the region"
  (interactive)
  (save-excursion
    (let ((start (if (region-active-p) (region-beginning) (point-at-bol)))
          (end   (if (region-active-p) (region-end) (point-at-eol))))
      (goto-char end)
      (unless (region-active-p)
        (newline))
      (insert (buffer-substring start end)))))
```

To tidy up a buffer we define this function borrowed from simenheg.

```
(defun tidy ()
  "Ident, untabify and unwhitespacify current buffer, or region if active."
  (interactive)
  (let ((beg (if (region-active-p) (region-beginning) (point-min)))
        (end (if (region-active-p) (region-end) (point-max))))
    (indent-region beg end)
    (whitespace-cleanup)
    (untabify beg (if (< end (point-max)) end (point-max)))))
```

2.13 Key bindings

Bindings for expand-region.

```
(global-set-key (kbd "C-'")  'er/expand-region)
(global-set-key (kbd "C-;")  'er/contract-region)
```

Bindings for multiple-cursors.

```
(global-set-key (kbd "C-c e")  'mc/edit-lines)
(global-set-key (kbd "C-c a")  'mc/mark-all-like-this)
(global-set-key (kbd "C-c n")  'mc/mark-next-like-this)
```

Bindings for ace-jump-mode.

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

Bind some native Emacs functions.

```
(global-set-key (kbd "C-c t")    'org-agenda-list)
(global-set-key (kbd "C-x k")    'kill-this-buffer)
```

```
(global-set-key (kbd "C-x C-r")  'recentf-ido-find-file)
```

Bind the functions defined above.

```
(global-set-key (kbd "C-c j")    'remove-whitespace-inbetween)
(global-set-key (kbd "C-x t")    'switch-to-shell)
(global-set-key (kbd "C-c d")    'duplicate-thing)
(global-set-key (kbd "<C-tab>")  'tidy)
```

Bindings for `move-text`.

```
(global-set-key (kbd "<M-S-up>")    'move-text-up)
(global-set-key (kbd "<M-S-down>")  'move-text-down)
```

2.14 Advice

An advice can be given to a function to make it behave differently. This advice makes `eval-last-sexp` (bound to `C-x C-e`) replace the sexp with the value.

```
(defadvice eval-last-sexp (around replace-sexp (arg) activate)
  "Replace sexp when called with a prefix argument."
  (if arg
      (let ((pos (point)))
        ad-do-it
        (goto-char pos)
        (backward-kill-sexp)
        (forward-sexp))
    ad-do-it))
```

3 Language mode specific

3.1 Lisp

`Pretty-lambda` provides a customizable variable `pretty-lambda-auto-modes` that is a list of common lisp modes. Here we can add some extra lisp-modes. We run the `pretty-lambda-for-modes` function to activate `pretty-lambda-mode` in lisp modes.

```
(dolist (mode '(slime-repl-mode inferior-lisp-mode inferior-scheme-mode))
  (add-to-list 'pretty-lambda-auto-modes mode))
```

```
(pretty-lambda-for-modes)
```

I use `Paredit` when editing lisp code, we enable this for all lisp-modes in the `pretty-lambda-auto-modes` list.

```
(dolist (mode pretty-lambda-auto-modes)
  ;; add paredit-mode to all mode-hooks
  (add-hook (intern (concat (symbol-name mode) "-hook")) 'paredit-mode))
```

3.1.1 Emacs Lisp

In `emacs-lisp-mode` we can enable `eldoc-mode` to display information about a function or a variable in the echo area.

```
(add-hook 'emacs-lisp-mode-hook 'turn-on-eldoc-mode)
(add-hook 'lisp-interaction-mode-hook 'turn-on-eldoc-mode)
```

3.1.2 Common lisp

I use Slime along with `lisp-mode` to edit Common Lisp code. Slime provides code evaluation and other great features, a must have for a Common Lisp developer. Quicklisp is a library manager for Common Lisp, and you can install Slime following the instructions from the site along with this snippet.

```
(when (file-exists-p "~/quicklisp/slime-helper.elc")
  (load (expand-file-name "~/quicklisp/slime-helper.elc")))
```

We can specify what Common Lisp program Slime should use (I use SBCL).

```
(setq inferior-lisp-program "sbcl")
```

To improve auto completion for Common Lisp editing we can use `ac-slime` which uses slime completions as a source.

```
(add-hook 'slime-mode-hook 'set-up-slime-ac)
(add-hook 'slime-repl-mode-hook 'set-up-slime-ac)

(eval-after-load "auto-complete"
  '(add-to-list 'ac-modes 'slime-repl-mode))
```

3.1.3 Scheme

Geiser provides features similar to Slime for Scheme editing. Everything works pretty much out of the box, the only thing we need to add is the auto completion.

```
(add-hook 'geiser-mode-hook 'ac-geiser-setup)
(add-hook 'geiser-repl-mode-hook 'ac-geiser-setup)
(eval-after-load "auto-complete"
  '(add-to-list 'ac-modes 'geiser-repl-mode))
```

3.2 Java and C

The `c-mode-common-hook` is a general hook that work on all C-like languages (C, C++, Java, etc...). I like being able to quickly compile using `C-c C-c` (instead of `M-x compile`), a habit from `latex-mode`.

```
(defun c-setup ()
  (local-set-key (kbd "C-c C-c") 'compile))

(add-hook 'c-mode-common-hook 'c-setup)
```

Some statements in Java appear often, and become tedious to write out. We can use abbrevs to speed this up.

```
(define-abbrev-table 'java-mode-abbrev-table
  '(("psv" "public static void main(String[] args) {" nil 0)
    ("sopl" "System.out.println" nil 0)
    ("sop" "System.out.printf" nil 0)))
```

To be able to use the abbrev table defined above, `abbrev-mode` must be activated.

```
(defun java-setup ()
  (abbrev-mode t)
  (setq-local compile-command (concat "javac " (buffer-name))))

(add-hook 'java-mode-hook 'java-setup)
```

3.3 Assembler

When writing assembler code I use # for comments. By defining `comment-start` we can add comments using `M-;` like in other programming modes. Also in assembler should one be able to compile using `C-c C-c`.

```
(defun asm-setup ()
  (setq comment-start "#")
  (local-set-key (kbd "C-c C-c") 'compile))

(add-hook 'asm-mode-hook 'asm-setup)
```

3.4 LaTeX

`.tex`-files should be associated with `latex-mode` instead of `tex-mode`.

```
(add-to-list 'auto-mode-alist '("\\.tex\\'" . latex-mode))
```

I like using the Minted package for source blocks in LaTeX. To make org use this we add the following snippet.

```
(add-to-list 'org-latex-packages-alist '("" "minted"))
(setq org-latex-listings 'minted)
```

Because Minted uses Pygments (an external process), we must add the `-shell-escape` option to the `org-latex-pdf-process` commands.

```
(setq org-latex-pdf-process
      (mapcar
       (lambda (str)
         (concat "pdflatex -shell-escape "
                 (substring str (string-match "-" str))))
       org-latex-pdf-process))
```

3.5 Python

Jedi offers very nice auto completion for `python-mode`. Mind that it is dependent on some python programs as well, so make sure you follow the instructions from the site.

```
(setq jedi:server-command
      (cons "python3" (cdr jedi:server-command))
```

```
      python-shell-interpreter "python3")
(add-hook 'python-mode-hook 'jedi:setup)
(setq jedi:complete-on-dot t)
(add-hook 'python-mode-hook 'jedi:ac-setup)
```

3.6 Haskell

`haskell-doc-mode` is similar to `eldoc`, it displays documentation in the echo area. Haskell has several indentation modes - I prefer using `haskell-indent`.

```
(add-hook 'haskell-mode-hook 'turn-on-haskell-doc-mode)
(add-hook 'haskell-mode-hook 'turn-on-haskell-indent)
```