Emacs Config - Jakob Klemm

Jakob Klemm

August 14, 2020

Setup

`use-package`

Verify the `use-package` installation and ensures the packages.

```
(require 'use-package-ensure)
(setq use-package-always-ensure t)
```

Update and compile

Update and compile all packages.

```
(use-package auto-compile
:config (auto-compile-on-load-mode))
(setq load-prefer-newer t)
```

Defaults

Default settings cloned from Harry R. Schwartz. Functions:

- Ensuring that files end with newlines,
- Always enabling syntax highlighting,
- Increasing the garbage collection threshold,
- Defaulting line-length to 80 characters,
- Creating parent directories after saving a deeply nested file.

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

Ressources

Add `resources` to the path

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

Interface

Scrollbar

Disable the scrollbar in the entire window and minibuffers.

```
;; Menu bar
(tool-bar-mode 0)
(menu-bar-mode 0)
(scroll-bar-mode -1)
;; Minibuffer
(set-window-scroll-bars (minibuffer-window) nil nil)
```

Improvements

Symbols

Use prettify symbols -> fancy lambdas

```
(global-prettify-symbols-mode t)
```

Bottom line

Use `moody` for a better bottom line.

```
 (use-package moody
 :config
 (setq x-underline-at-descent-line t
moody-mode-line-height 30)
(moody-replace-mode-line-buffer-identification)
(moody-replace-vc-mode))
```

Minor modes

Hide all minor modes with `minions`.

```
(use-package minions
:config
(setq minions-mode-line-lighter ""
minions-mode-line-delimiters '("" . ""))
(minions-mode 1))
```

Scrolling

Don't skip to center of page when at bottom / top, *normal* smooth scrolling.

```
(setq scroll-conservatively 100)
```

Fullscreen:

```
(add-to-list 'default-frame-alist '(fullscreen . maximized))
```

Current line

Hightlight the current line.

```
(global-hl-line-mode)
```

Line numbers

```
(global-display-line-numbers-mode)
```

Theme

Load the elixify theme from Astonj with some modifications.

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

(load-theme ‘elixify t)
```

Dashboard

Setup the dashboard with come modifications and configs. “Every time I see this package I think to myself ”People exit Emacs¿“”

Dependancies

Page-break-lines

```
(turn-on-page-break-lines-mode)
```

Icons

```
(require 'all-the-icons)
```

Setup

Setup the dashboard.

```
(require 'dashboard)
(dashboard-setup-startup-hook)
;; Or if you use use-package
(use-package dashboard
:ensure t
:config
(dashboard-setup-startup-hook))
```

Config

Options and configuration for dashboard following the readme.

```
;; Set the banner
(setq dashboard-startup-banner 2)
;; Content is not centered by default. To center, set
(setq dashboard-center-content t)
;; Icons
(setq dashboard-set-heading-icons t)
(setq dashboard-set-file-icons t)
;; Navigator
(setq dashboard-set-navigator t)
;; Init info
(setq dashboard-set-init-info t)
;; Message
(setq dashboard-footer-messages '("Every time I see this package I think to myself \"P
```

Font

Use Fira Code as default font.

```
(set-face-attribute
'default nil
:font "Fira Code"
:weight 'normal
:width 'normal
)
```

New window

Directly switch to new window after opening. (Credit: hrs)

```
(defun hrs/split-window-below-and-switch ()
"Split the window horizontally, then switch to the new pane."
(interactive)
(split-window-below)
(balance-windows)
(other-window 1))

(defun hrs/split-window-right-and-switch ()
"Split the window vertically, then switch to the new pane."
(interactive)
(split-window-right)
(balance-windows)
(other-window 1))

;; Keys
(global-set-key (kbd "C-x 2") 'hrs/split-window-below-and-switch)
(global-set-key (kbd "C-x 3") 'hrs/split-window-right-and-switch)
```

Beacon

Beacon for highlighting the cursor when switching buffers.

```
(use-package beacon
:custom
(beacon-color "#c678dd")
:hook (after-init . beacon-mode))
```

Title

Set the window title to the current file.

```
(setq-default frame-title-format
'(:eval
(format "%s@%s: %s %s"
(or (file-remote-p default-directory 'user)
user-real-login-name)
(or (file-remote-p default-directory 'host)
system-name)
(buffer-name)
(cond
(buffer-file-truename
(concat "(" buffer-file-truename ")"))
(dired-directory
(concat "{" dired-directory "}"))
(t
"[no file]")))))
```

Tabs

List buffers like tabs.

```
(require 'centaur-tabs)
(centaur-tabs-mode t)
(global-set-key (kbd "C-x x")  'centaur-tabs-backward)
(global-set-key (kbd "C-x c") 'centaur-tabs-forward)
(global-set-key (kbd "C-x y") 'centaur-tabs-toggle-groups)

(setq centaur-tabs-style "bar")
(setq centaur-tabs-set-icons t)
(setq centaur-tabs-height 32)
(setq centaur-tabs-set-bar 'left)

(setq centaur-tabs-set-icons t)
(setq centaur-tabs-plain-icons t)
```

Exclude certain buffers from centaur.

```
(defun centaur-tabs-hide-tab (x)
(let ((name (format "%s" x)))
```

```
(or
(string-prefix-p "*dashboard*" name)
(string-prefix-p "*scratch*" name)
(string-prefix-p "*Messages*" name)
(string-prefix-p "*Completions*" name)
(string-prefix-p "*Org PDF LaTeX Output*" name)
(and (string-prefix-p "magit" name)
(not (file-name-extension name)))
)))
```

Tab Grouping

```
;; Show groups
(setq centaur-tabs--buffer-show-groups t)
(centaur-tabs-group-by-projectile-project)

(defun centaur-tabs-buffer-groups ()
"‘centaur-tabs-buffer-groups’ control buffers’ group rules.
Group centaur-tabs with mode if buffer is derived from ‘eshell-mode’ ‘emacs-lisp-mode’
All buffer name start with * will group to \"Emacs\".
Other buffer group by ‘centaur-tabs-get-group-name’ with project name."
(list
(cond
((or (string-equal "*" (substring (buffer-name) 0 1))
(memq major-mode ’(magit-process-mode
 magit-status-mode
 magit-diff-mode
 magit-log-mode
 magit-file-mode
 magit-blob-mode
 magit-blame-mode
 )))
 "Emacs")
 ((derived-mode-p ’prog-mode)
 "Editing")
 ((derived-mode-p ’dired-mode)
 "Dired")
 ((memq major-mode ’(helpful-mode
 help-mode))
 "Help")
 ((memq major-mode ’(org-mode
```

```
 org-agenda-clockreport-mode
 org-src-mode
 org-agenda-mode
 org-beamer-mode
 org-indent-mode
 org-bullets-mode
 org-cdlatex-mode
 org-agenda-log-mode
 diary-mode))
 "OrgMode")
 (t
 (centaur-tabs-get-group-name (current-buffer))))))
```

Projects

Management

Projectile for project management.

```
(projectile-mode +1)
(define-key projectile-mode-map (kbd "C-c p") 'projectile-command-map)
```

Completion

```
 (use-package company
 :diminish company-mode
:ensure t
:init (add-hook 'after-init-hook 'global-company-mode)
:config
(setq company-idle-delay              0.1
company-minimum-prefix-length   2
company-show-numbers            t
company-tooltip-limit           20
company-dabbrev-downcase        nil
company-backends                '((company-gtags))
)
:bind ("s-;" . company-complete-common)
)


 (use-package company-box
```

```
:hook (company-mode . company-box-mode))

(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))

(global-set-key (kbd "C-ö") 'company-complete)

(use-package lsp-mode
:commands lsp
:ensure t
:diminish lsp-mode
:hook
(elixir-mode . lsp)
:init
(add-to-list 'exec-path "~/.emacs.d/elixir-ls"))
```

General

Indentation

Show tabs as 2 wide.

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

CamelCase

Treat camel casing (the best and only right variable naming system) as multiple words.

```
(use-package subword
:config (global-subword-mode 1))
```

UTF-8

Treat every file as UTF-8 by default.

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

Wrap

Auto wrap paragraphs. Or use `M-q`.

```
(add-hook 'text-mode-hook 'auto-fill-mode)
(add-hook 'gfm-mode-hook 'auto-fill-mode)
(add-hook 'org-mode-hook 'auto-fill-mode)
```

Spacing

Cycle spacing options.

```
(global-set-key (kbd "M-SPC") 'cycle-spacing)
```

Modes

Other *cool* default modes.

```
(show-paren-mode 1)
(column-number-mode 1)
(size-indication-mode 1)
(transient-mark-mode 1)
(delete-selection-mode 1)
```

Kill current

Kill the current buffer instead of asking.

```
(defun kill-current-buffer ()
(interactive)
(kill-buffer (current-buffer)))

;; Keybind
(global-set-key (kbd "C-x k") 'kill-current-buffer)
```

Save

Save the location within a file.

```
(save-place-mode t)
```

Which key

Helpful with long keybinds.

```
(use-package which-key
:config (which-key-mode))
```

Jump

Jump to function definitions. (Works with elixir)

```
(use-package dumb-jump
:bind (("M-g o" . dumb-jump-go-other-window)
("M-g j" . dumb-jump-go)
("M-g b" . dumb-jump-back)
("M-g i" . dumb-jump-go-prompt)
("M-g x" . dumb-jump-go-prefer-external)
("M-g z" . dumb-jump-go-prefer-external-other-window))
:config (setq dumb-jump-selector 'ivy)
:ensure)
```

google-this

Automaticly google something.

```
(google-this-mode 1)
(global-set-key (kbd "C-c t") 'google-this)
```

Terminal

Bind `C-x t` to `eshell`.

```
(global-set-key (kbd "C-x t") 'eshell)
```

Ido

Globaly enable IDO mode

```
(setq ido-enable-flex-matching t)
(setq ido-everywhere t)
(ido-mode 1)
(setq ido-use-filename-at-point 'guess)
(setq ido-create-new-buffer 'always)
(setq ido-file-extensions-order '(".ex" ".exs" ".org" ".md" ".txt" ".py" ".emacs" ".xm
```

Git

Magit keybinds.

```
(global-set-key (kbd "C-x g") 'magit-status)
(global-set-key (kbd "C-x p") 'magit-init)
```

Programming

Elixir

Elixir major mode with synatx highlighting etc.

```
(unless (package-installed-p 'elixir-mode)
(package-install 'elixir-mode))

;; Start smartpares for "do" "end" with elixir mode
(add-hook 'elixir-mode-hook #'smartparens-mode)
```

Commands:
Use

$$M - xelixir - format$$

to format the document following mix styleguide.

Webmode

Web mode and enable rainbow mode for hex colors.

```
(use-package web-mode)
(add-hook 'web-mode-hook
(lambda ()
(rainbow-mode)
(rspec-mode)
(setq web-mode-markup-indent-offset 2)))
```

Golang

Golang major mode.

```
(use-package go-mode)
(use-package go-errcheck)
```

JavaScript

JavaScript major mode.

```
(use-package coffee-mode)
```

Rust

Rust major mode.

```
(use-package rust-mode)
```

Scala

Scala major mode.

```
(use-package scala-mode
:interpreter
("scala" . scala-mode))
(use-package sbt-mode)
```

Markdown

Github markdown.

```
(use-package markdown-mode
:commands gfm-mode
:mode (("\\.md$" . gfm-mode))
:config
(setq markdown-command "pandoc --standalone --mathjax --from=markdown")
(custom-set-faces
'(markdown-code-face ((t nil)))))
```

Org-mode

Bullets

Use org-bulltes whenever possible.

```
(use-package org-bullets
:init
(add-hook 'org-mode-hook 'org-bullets-mode))
```

Folded

Instead of "..." show a downward pointing arrow at the end of title. TODO Change symbol or something.

```
(setq org-ellipsis "")
```

Codeblock

Higlight the entire code block when editing.

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

Todos

(copied from hrs/config)

Location

Org document storage location for archive and other documents.

```
(setq org-directory "~/documents/org")

(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 "~/emacs/main.org"))
(setq org-archive-location
(concat (org-file-path "archive.org") "::* From %s"))
```

Archive

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

```
(defun hrs/mark-done-and-archive ()
"Mark the state of an org-mode item as DONE and archive it."
(interactive)
(org-todo 'done)
(org-archive-subtree))
;; Shortcut to archive
(define-key org-mode-map (kbd "C-c C-x C-s") 'hrs/mark-done-and-archive)
```

Time

Record the time that a todo was archived.

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

Check

Ensure that a task can't be marked as done if it contains unfinished subtasks or checklist items. This is handy for organizing "blocking" tasks hierarchically.

```
(setq org-enforce-todo-dependencies t)
(setq org-enforce-todo-checkbox-dependencies t)
```

Stats

Add new states to the todo cycle to extend the basic TODO and DONE states that org mode normally provides.

```
(setq org-todo-keywords
'((sequence "TODO" "SEARCH" "PROGRESS" "BLOCKED" "|" "DONE" "PAL")))
```

Export

Allow export to markdown and beamer (for presentations).

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

Code

Allow `babel` to evaluate Emacs lisp, Ruby, dot, or Gnuplot code.

```
(use-package gnuplot)
(org-babel-do-load-languages
'org-babel-load-languages
'((emacs-lisp . t)
(ruby . t)
(dot . t)
(gnuplot . t)
(python . t)
(go . t)
```

```
(sql . t)
(elixir . t)
))
```

Don't ask before evaluating code blocks.

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

`htmlize`

Use `htmlize` to ensure that exported code blocks use syntax highlighting.

```
(use-package htmlize)
```

Translate quoates into correct style.

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

HTML

Disable footer.

```
(setq org-html-postamble nil)
```

Tex

Parse

Parse file after loading it.

```
(setq TeX-parse-self t)
```

PDF-Latex

```
(setq TeX-PDF-mode t)
```

Math mode.

```
(add-hook 'LaTeX-mode-hook
(lambda ()
(LaTeX-math-mode)
(setq TeX-master t)))
```

Templates

```
(setq org-reveal-note-key-char nil)
;; Easy templates <s
;; TODO: insert (require 'org-tempo)
```