# Jefter's Emacs configurations

Jefter Santiago

November 20, 2021

## Startup performance

Taken from `https://github.com/daviwil/emacs-from-scratch/blob/master/Emacs.org` [frame=lines,linenos=true]common-lisp ;; The default is 800 kilobytes. Measured in bytes. (setq gc-cons-threshold (* 50 1000 1000))

(defun efs/display-startup-time () (message "Emacs loaded in (format "(float-time (time-subtract after-init-time before-init-time))) gcs-done))

(add-hook 'emacs-startup-hook 'efs/display-startup-time)

## General config

### Personal configs

#### Info

[frame=lines,linenos=true]common-lisp (setq-default user-full-name "Jefter Santiago") (setq-default user-mail-address "jeftersantiago@protonmail.com") (load " /Dropbox/private-configs/private.el")

#### Custom file

Separated file to store generated config from emacs. Puts the outputs in a separated file.

Taken from:

- `https://www.reddit.com/r/emacs/comments/4x655n/packageselectedpackages_always_appear_after/`
- `https://www.reddit.com/r/emacs/comments/53zpv9/how_do_i_get_emacs_to_stop_adding_custom_fields/`

[frame=lines,linenos=true]common-lisp (setq custom-file " /.emacs.d/custom.el")

### Backup disabled

[frame=lines,linenos=true]common-lisp (setq-default backup-inhibited t) (setq-default create-lockfiles nil) (setq-default make-backup-files nil)

### Automatic package updates

Automatizating the `M-x auto-package-update-now` command. [frame=lines,linenos=true]common-lisp (use-package auto-package-update :ensure t :custom (auto-package-update-interval 7) (auto-package-update-prompt-before-update t) (auto-package-update-hide-results t) :config (auto-package-update-maybe))

### No auto-save

Do not create the `auto-save/` folder [frame=lines,linenos=true]common-lisp (use-package no-littering :ensure t )
;; no-littering doesn't set this by default so we must place ;; auto save files in the same path as it uses for sessions (setq auto-save-file-name-transforms '((".*" ,(no-littering-expand-var-file-name "auto-save/") t)))

## Appearance

### System default variables

[frame=lines,linenos=true]common-lisp (scroll-bar-mode 0) (tool-bar-mode 0) (menu-bar-mode 0) (fringe-mode 0) (font-lock-mode t)
(setq initial-scratch-message nil) (setq inhibit-startup-message t)
(setq-default indent-tabs-mode nil) (setq-default tab-width 4) (setq confirm-kill-processes nil) (setq-default truncate-lines t) (setq-default fill-column 80)
(setq-default cursor-type 'square) (defalias 'yes-or-no-p 'y-or-n-p) Emacs window covering entire screen. Solution found in: `https://github.com/d12frosted/homebrew-emacs-plus/issues/177` [frame=lines,linenos=true]common-lisp (setq frame-resize-pixelwise t)

### Rainbow-delimiters

Makes the parenthesis shine [frame=lines,linenos=true]common-lisp (use-package rainbow-delimiters :hook (prog-mode . rainbow-delimiters-mode) :ensure t) (add-hook 'emacs-lisp-mode-hook 'rainbow-delimiters-mode)

### Theme and transparency

#### Theme

Loading theme and setting modeline background color. [frame=lines,linenos=true]common-lisp (use-package doom-themes :init (load-theme 'doom-Iosvkem t))
; (use-package spacemacs-theme ; :defer t ; :init (load-theme 'spacemacs-light t)) ; (set-background-color "black")

#### Transparency

Enabling transparency [frame=lines,linenos=true]common-lisp ; (set-frame-parameter (selected-frame) 'alpha '(98 98)) ; (add-to-list 'default-frame-alist '(alpha 98 98))

### Font

Taken from here: `https://emacs.stackexchange.com/q/45895` [frame=lines,linenos=true]common-lisp (set-frame-font "Monospace-12:antialias=true") [frame=lines,linenos=true]common-lisp (use-package default-text-scale :ensure t :hook (after-init . default-text-scale-mode)) (set-language-environment "UTF-8") (global-prettify-symbols-mode t) (prefer-coding-system 'utf-8)
(global-set-key (kbd "C-x C-l") 'font-lock-mode)

### Modeline

Nice and simple. [frame=lines,linenos=true]common-lisp (use-package all-the-icons :ensure t) (use-package doom-modeline :init (doom-modeline-mode 1) :custom ((doom-modeline-height 25)) :ensure t)

### Line number

[frame=lines,linenos=true]common-lisp ;(global-display-line-numbers-mode) ;(setq display-line-numbers-type 'relative)

## Dashboard

[frame=lines,linenos=true]common-lisp (use-package dashboard :ensure t :init (progn (setq dashboard-show-shortcuts nil) (setq dashboard-center-content nil) (setq dashboard-banner-logo-title "EMACS") (setq dashboard-set-file-icons t) (setq dashboard-set-heading-icons t) (setq dashboard-startup-banner

" /.emacs.d/images/emacs-logo.png") ) :config (dashboard-setup-startup-hook))

# Dired

## Sidebar

[frame=lines,linenos=true]common-lisp ; (use-package dired-sidebar ; :bind (("C-x C-n" . dired-sidebar-toggle-sidebar)) ; :ensure t ; :commands (dired-sidebar-toggle-sidebar) ; :init)

## Icons

[frame=lines,linenos=true]common-lisp ; (use-package all-the-icons-dired :ensure t) ; (add-hook 'dired-mode-hook 'all-the-icons-dired-mode)

## Default applications to extensions

[frame=lines,linenos=true]common-lisp (use-package dired-open :ensure t :config (setq dired-open-extensions '(("doc" . "openoffice4") ("docx" . "openoffice4") ("xopp" . "xournalpp") ("gif" . "mirage") ("jpeg" ."mirage") ("jpg" . "mirage") ("png" . "mirage") ("mkv" . "mpv") ("avi" . "mpv") ("mov" . "mpv") ("mp3" . "mpv") ("mp4" . "mpv") ("pdf" . "xreader") ("webm" . "mpv"))))

## Hide dotfiles and extra information (aka ownership and such)

[frame=lines,linenos=true]common-lisp (use-package dired-hide-dotfiles :ensure t :config (dired-hide-dotfiles-mode) (define-key dired-mode-map "." 'dired-hide-dotfiles-mode))

(setq-default dired-listing-switches "-lhvA") (add-hook 'dired-mode-hook (lambda () (dired-hide-details-mode 1))) ;; Taken from here: https://emacs.stackexchange.com/questions/13080/reloading-directory-local-variables/1309613096 (defun my-reload-dir-locals-for-current-buffer () "reload dir locals for the current buffer" (interactivye) (let ((enable-local-variables :all)) (hack-dir-local-variables-non-file-buffer))) (defun my-reload-dir-locals-for-all-buffer-in-this-directory () "For every buffer with the same ‘default-directory‘ as the current buffer's, reload dir-locals." (interactive) (let ((dir default-directory)) (dolist (buffer (buffer-list)) (with-current-buffer buffer (when (equal default-directory dir)) (my-reload-dir-locals-for-current-buffer)))))

## Text navigation

### Inserting new line

Add a new line below the current line [frame=lines,linenos=true]common-lisp (defun insert-new-line-below () (interactive) (let ((oldpos (point))) (end-of-line) (newline-and-indent)))
(global-set-key (kbd "C-o") 'insert-new-line-below)

### Scrolling

[frame=lines,linenos=true]common-lisp (setq kill-buffer-query-functions (remq 'process-kill-buffer-query-function kill-buffer-query-functions)) ;; mouse scrolls very slowly (setq confirm-kill-processes nil) (setq scroll-step 1 scroll-conservatively 10000 mouse-wheel-scroll-amount '(1 ((shift) . 1)) mouse-wheel-progressive-speed nil mouse-wheel-follow-mouse 't)

## Evil Mode

Yes, i use vim too. [frame=lines,linenos=true]common-lisp (add-to-list 'load-path " /.emacs.d/evil") (require 'evil) (evil-mode 1)

## Smartparents

Creates pairs of parenthesis in a smart way [frame=lines,linenos=true]common-lisp (use-package smartparens :ensure t :config (sp-use-paredit-bindings) (add-hook 'prog-mode-hook 'smartparens-mode) (sp-pair "" nil :post-handlers '(("||[i]" "RET"))))

## Ace-window

[frame=lines,linenos=true]common-lisp (use-package ace-window :ensure t :init (progn (global-set-key [remap other-window] 'ace-window) (custom-set-faces '(aw-leading-char-face ((t (:inherit ace-jump-face-foreground :height 2.0)))))))

# Multi-term

[frame=lines,linenos=true]common-lisp (use-package multi-term :ensure t) (setq multi-term-program "/bin/bash") (global-set-key (kbd "C-x t") 'multi-term)

# Org-mode

## General config

### Variables

[frame=lines,linenos=true]common-lisp
(setq org-startup-folded t) (setq org-src-tab-acts-natively t) (setq org-src-window-setup 'current-window)
(setq visual-fill-column-width 100 visual-fill-column-center-text t)
(setq-default fill-column 79) (setq org-refile-use-outline-path t) (setq org-outline-path-complete-in-steps nil)
(setq-default org-image-actual-width 620)

### Org-bullets

[frame=lines,linenos=true]common-lisp (use-package org-bullets :hook (org-mode . org-bullets-mode) :custom (org-bullets-bullet-list '("" "" "" "" "" "" "" "")))) (setq org-ellipsis "")
(font-lock-add-keywords 'org-mode '(("[: *space* :]] *
(−
)"(0(*prog*1()(*compose* − *region*(*match* − *beginning*1)(*match* − *end*1)""))))))

### Center org buffers

[frame=lines,linenos=true]common-lisp ; (defun efs/org-mode-visual-fill () ; (visual-fill-column-mode 1))
; (use-package visual-fill-column ; :ensure t ; :hook (org-mode . efs/org-mode-visual-fill))

## Tasks management

[frame=lines,linenos=true]common-lisp (add-hook 'org-mode-hook 'auto-fill-mode) (setq org-todo-keywords '((sequence "TODO(t)" "NEXT(n)" "|" "DONE(d!)" "DROP(x!)")) org-log-into-drawer t)

(defun org-file-path (filename) ;; return the absolute address of an org file, give its relative name (concat (file-name-as-directory org-directory) filename))

(setq org-index-file (org-file-path "TODOs.org")) (setq org-archive-location (concat (org-file-path "DONE.org") "::* From

;; copy the content out of the archive.org file and yank in the inbox.org (setq org-agenda-files (list org-index-file)) ;; 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)) (setq org-log-done 'time)

### Org capture templates

this breaks org-roam [frame=lines,linenos=true]common-lisp

(add-to-list 'org-structure-template-alist '("g" . "$\frac{d?}{d}$"))

### Displaying inline images

The joy of programming = `https://joy.pm/post/2017-09-17-a_graphviz_primer/nn` [frame=lines,linenos=true]common-lisp (defun my/fix-inline-images () (when org-inline-image-overlays (org-redisplay-inline-images))) (add-hook 'org-babel-after-execute-hook 'my/fix-inline-images) (add-hook 'org-mode-hook 'org-toggle-inline-images)

### Code

[frame=lines,linenos=true]common-lisp (org-babel-do-load-languages 'org-babel-load-languages '((python . t))) (require 'color) (set-face-attribute 'org-block nil :background (color-darken-name (face-attribute 'default :background) 3))

### org-publishing

#### Compiling pdf

[frame=lines,linenos=true]common-lisp (setq org-latex-pdf-process (list "latexmk -pdflatex='lualatex -shell-escape -interaction nonstopmode' -pdf -f

#### Open pdfs in xreader

Makes UTF-8 symbols appears in buffer I use it for editing Latex [frame=lines,linenos=true]common-lisp (add-hook 'org-mode-hook (lambda () (org-toggle-pretty-entities))) ;;

Opening pdfs (add-to-list 'org-file-apps '("
.pdf" . "xreader

### Org publishing folder

[frame=lines,linenos=true]common-lisp ; (defvar org-export-output-directory-prefix " /Documents" "prefix of directory used for org-mode export")

; (defadvice org-export-output-file-name (before org-add-export-dir activate) ; "Modifies org-export to place exported files in a different directory" ; (when (not pub-dir) ; (setq pub-dir (concat org-export-output-directory-prefix (substring extension 1))) ; (when (not (file-directory-p pub-dir)) ; (make-directory pub-dir))))

## Key-bindings in org-mode

[frame=lines,linenos=true]common-lisp (global-set-key (kbd "C-c C-x C-s") 'hrs/mark-done-and-archive) (global-set-key (kbd "C-c i") 'org-toggle-inline-images) (global-set-key (kbd "C-x p") 'org-latex-export-to-pdf) (define-key global-map "-cc" 'org-capture)

## Org LaTeX

### Preview Latex fragments

- org-fragtog loading latex fragments `./images/latex-preview-example.gif`
  - TODO [ ] Still want a way to store all images generated in one place.

  [frame=lines,linenos=true]common-lisp ; load the latex fragments automatically (use-package org-fragtog :ensure t) (add-hook 'org-mode-hook 'org-fragtog-mode)

  ; this is the only way to really work (idk y) (setq org-latex-create-formula-image-program 'dvisvgm)

  ; adjusting the size (setq org-format-latex-options (plist-put org-format-latex-options :scale 2.0))

  ; (setq org-latex-caption-above nil)

**bibtex**

[frame=lines,linenos=true]common-lisp (setq org-latex-to-pdf-process (list "latexmk -pdf

**minted**

[frame=lines,linenos=true]common-lisp
(setq org-latex-listings 'minted) (setq org-latex-minted-options '(("frame" "") ("linenos=true")))

**Tikz**

[frame=lines,linenos=true]common-lisp ; (add-hook 'org-mode-hook ; (lambda () (texfrag-mode))
(add-to-list 'org-latex-packages-alist '("" "tikz" t)) (eval-after-load "preview" '(add-to-list 'preview-default-preamble "
PreviewEnvironmenttikzpicture" t))

### Org-ref

Org references in bibtex Found in: `https://github.com/berquist/dotfiles/blob/main/dotfiles/emacs.d/config.org` [frame=lines,linenos=true]common-lisp ; (use-package org-ref ; :disabled t ; :config ; (setq reftex-default-bibliography " /bibliography2/references.bib") ; (setq org-ref-default-bibliography " /bibliography2/references.bib") ; (setq org-ref-bibliography-notes " /bibliography2/notes.org") ; (setq org-ref-pdf-directory " /bibliography2/pdfs") ; (setq bibtex-completion-bibliography " /bibliography2/references.bib") ; (setq bibtex-completion-library-path " /bibliography2/pdfs") ; (setq bibtex-completion-notes-path " /bibliography2/notes"))

## Org-babel

### Loading org-babel

[frame=lines,linenos=true]common-lisp (org-babel-do-load-languages 'org-babel-load-languages '((emacs-lisp . t) (python . t) )) (setq org-confirm-babel-evaluate t)

### Structure templates

[frame=lines,linenos=true]common-lisp (require 'org-tempo) (add-to-list 'org-modules 'org-tempo t)

(with-eval-after-load 'org (add-to-list 'org-structure-template-alist '("el" . "src emacs-lisp")) (add-to-list 'org-structure-template-alist '("jl" . "src julia")) (add-to-list 'org-structure-template-alist '("sh" . "src shell")) (add-to-list 'org-structure-template-alist '("py" . "src python")))

## Org-roam

[frame=lines,linenos=true]common-lisp (use-package org-roam :ensure t :custom (org-roam-v2-ack t) (org-roam-directory (file-truename " /Dropbox/notes/")) (org-roam-completion-everywhere t) (org-roam-capture-templates '(("d" "Default notes" plain ":if-new (file+head "*slug.org*"" + *title* :title") :unnarrowed t) ("p" "Notes on physics" plain "+setupfile: /Dropbox/Templates/physics.org * :if-new (file+head "*slug.org*"" + *title* :title") :unnarrowed t) ("m" "Notes on mathematics" plain "+setupfile: /Dropbox/Templates/mathematics.org * :if-new (file+head "*slug.org*"" + *title* :title") :unnarrowed t) ("c" "Notes on computing" plain "+setupfile: /Dropbox/Templates/computing.org * :if-new (file+head "*slug.org*"" + *title* :title") :unnarrowed t))) :bind (("C-c n l" . org-roam-buffer-toggle) ("C-c n f" . org-roam-node-find) ("C-c n g" . org-roam-graph) ("C-c n i" . org-roam-node-insert) ("C-c n c" . org-roam-capture) ;; Dailies ("C-c n j" . org-roam-dailies-capture-today)) :config (org-roam-db-autosync-mode) ;; If using org-roam-protocol (require 'org-roam-protocol))

(setq org-roam-v2-ack t)

### or-roam-ui

[frame=lines,linenos=true]common-lisp (use-package websocket :ensure t) (use-package simple-httpd :ensure t) (add-to-list 'load-path " /.emacs.d/org-roam-ui") (load-library "org-roam-ui")

## Swiper

[frame=lines,linenos=true]common-lisp (use-package swiper :ensure t :config (progn (ivy-mode 1) (setq ivy-use-virtual-buffers t) (global-set-key "-s" 'swiper)))

## Try

[frame=lines,linenos=true]common-lisp (use-package try :ensure t :config (progn (global-set-key (kbd "C-x b") 'ivy-switch-buffer))) (setq ivy-use-virtual-buffers t) (setq ivy-display-style 'fancy)

## Which-key

[frame=lines,linenos=true]common-lisp (use-package which-key :ensure t :config (which-key-mode))

## Yasnippet

[frame=lines,linenos=true]common-lisp
(use-package yasnippet :ensure t :config (setq yas-snippet-dirs '(" /.emacs.d/snippets")) (yas-global-mode 1))

## Flycheck

[frame=lines,linenos=true]common-lisp (use-package flycheck :ensure t :init (global-flycheck-mode t))

## projectile

[frame=lines,linenos=true]common-lisp (use-package projectile :diminish projectile-mode :config (projectile-mode) :bind-keymap ("C-c p" . projectile-command-map) :init (when (file-directory-p " /Projects/") (setq projectile-project-search-path '(" /Projects/")))) (setq projectile-switch-projects-action 'projectile-dired)

## treemacs

[frame=lines,linenos=true]common-lisp (use-package treemacs :ensure t :defer t :init (with-eval-after-load 'winum (define-key winum-keymap (kbd "M-0") 'treemacs-select-window)) :config (progn (setq treemacs-collapse-dirs (if treemacs-python-executable 3 0) treemacs-deferred-git-apply-delay 0.5 treemacs-directory-name-transformer 'identity treemacs-display-in-side-window t treemacs-eldoc-display t treemacs-file-event-delay 5000 treemacs-file-extension-regex

treemacs-last-period-regex-value treemacs-file-follow-delay 0.2 treemacs-file-name-transformer 'identity treemacs-follow-after-init t treemacs-expand-after-init t treemacs-git-command-pipe "" treemacs-goto-tag-strategy 'refetch-index treemacs-indentation 2 treemacs-indentation-string " " treemacs-is-never-other-window nil treemacs-max-git-entries 5000 treemacs-missing-project-action 'ask treemacs-move-forward-on-expand nil treemacs-no-png-images nil treemacs-no-delete-other-windows t treemacs-project-follow-cleanup nil treemacs-persist-file (expand-file-name ".cache/treemacs-persist" user-emacs-directory) treemacs-position 'left treemacs-read-string-input 'from-child-frame treemacs-recenter-distance 0.1 treemacs-recenter-after-file-follow nil treemacs-recenter-after-tag-follow nil treemacs-recenter-after-project-jump 'always treemacs-recenter-after-project-expand 'on-distance treemacs-litter-directories '("/node$_{m}odules$""/.venv""/.cask
$show-cursorniltreemacs-show-hidden-filesttreemacs-silent-filewatchniltreemacs-silent-refreshniltreemacs-sorting'alphabetic-asctreemacs-select-when-already-in-treemacs'move-backtreemacs-space-between-root-nodesttreemacs-tag-follow-cleanupttreemacs-tag-follow-delay1.5treemacs-text-scaleniltreemacs-user-mode-line-formatniltreemacs-user-header-line-formatniltreemacs-wide-toggle-width70treemacs-width35treemacs-width-increment1treemacs-width-is-initially-lockedttreemacs-workspace-switch-cleanupnil$)

;; The default width and height of the icons is 22 pixels. If you are ;; using a Hi-DPI display, uncomment this to double the icon size. ;;(treemacs-resize-icons 44)

(treemacs-follow-mode t) (treemacs-filewatch-mode t) (treemacs-fringe-indicator-mode 'always)

(pcase (cons (not (null (executable-find "git"))) (not (null treemacs-python-executable))) ('(t . t) (treemacs-git-mode 'deferred)) ('(t . )(*treemacs-git-mode'simple*)))

(treemacs-hide-gitignored-files-mode nil)) :bind (:map global-map ("M-0" . treemacs-select-window) ("C-x n 1" . treemacs-delete-other-windows) ("C-x n t" . treemacs) ("C-x n B" . treemacs-bookmark) ("C-x n C-t" . treemacs-find-file) ("C-x n M-t" . treemacs-find-tag)))

(use-package treemacs-evil :after (treemacs evil) :ensure t)

(use-package treemacs-projectile :after (treemacs projectile) :ensure t)

(use-package treemacs-icons-dired :hook (dired-mode . treemacs-icons-dired-enable-once) :ensure t)

; (use-package treemacs-magit ; :ensure t)

(use-package treemacs-persp ;;treemacs-perspective if you use perspective.el vs. persp-mode :after (treemacs persp-mode) ;;or perspective vs. persp-mode :ensure t :config (treemacs-set-scope-type 'Perspectives))

# Latex

### setup

Loads `Auctex` and `lsp` for latex. [frame=lines,linenos=true]common-lisp (use-package auctex :hook ((latex-mode LaTeX-mode) . lsp) :config (add-to-list 'texmathp-tex-commands "dmath" 'env-on) (texmathp-compile) :init (setq-default TeX-master 'shared) ;; nil is the default; this remains here as a reminder that setting it to ;; true makes emacs hang on every save when enabled. (setq TeX-auto-save nil) (setq TeX-parse-self t))

(setq-default TeX-master nil) (use-package auctex-latexmk :config (setq auctex-latexmk-inherit-TeX-PDF-mode t) :init (auctex-latexmk-setup))

(add-hook 'LaTeX-mode-hook 'visual-line-mode) (add-hook 'LaTeX-mode-hook 'flyspell-mode) (add-hook 'LaTeX-mode-hook 'LaTeX-math-mode)

### compile shortcuts and open the file in my favorite pdf reader (`xreader`).

[frame=lines,linenos=true]common-lisp (setq TeX-view-program-selection '((output-pdf "PDF Viewer")))

(setq TeX-view-program-list '(("PDF Viewer" "xreader

(eval-after-load "tex" '(add-to-list 'TeX-command-list '("PdfLatex" "pdflatex -interaction=nonstopmode

# lsp-mode

[frame=lines,linenos=true]common-lisp (defun efs/lsp-mode-setup () (setq lsp-headerline-breadcrumb-segments '(path-up-to-project file symbols)) (lsp-headerline-breadcrumb-mode))

(use-package lsp-mode :ensure t :commands (lsp lsp-deferred) :hook (lsp-mode . efs/lsp-mode-setup) :init (setq lsp-keymap-prefix "C-c l") ;; Or 'C-l', 's-l' :config (lsp-enable-which-key-integration t))

(use-package lsp-ivy :ensure t :after lsp)

(use-package lsp-treemacs :ensure t :after lsp) (global-set-key (kbd "C-x C-n") 'lsp-treemacs-symbols)

(use-package lsp-mode :commands lsp :hook ((fortran-mode f90-mode sh-mode) . lsp) :config (setq lsp-auto-guess-root t) (setq lsp-enable-snippet nil) (setq lsp-file-watch-threshold 500000) (setq lsp-headerline-breadcrumb-enable nil) (setq lsp-modeline-diagnostics-enable nil) (setq lsp-prefer-flymake nil) (setq lsp-rust-clippy-preference "on"))

## Eglot

[frame=lines,linenos=true]common-lisp (use-package eglot :ensure t) (add-hook 'LaTeX-mode-hook 'eglot-ensure)

# C/C++

Taken from: `https://stackoverflow.com/a/3346308` [frame=lines,linenos=true]common-lisp ;; function decides whether .h file is C or C++ header, sets C++ by ;; default because there's more chance of there being a .h without a ;; .cc than a .h without a .c (ie. for C++ template files) (defun ejb/c-c++-header () "Sets either c-mode or c++-mode, whichever is appropriate for the header, based upon the associated source code file." (interactive) (let ((c-filename (concat (substring (buffer-file-name) 0 -1) "c"))) (if (file-exists-p c-filename) (c-mode) (c++-mode)))) (add-to-list 'auto-mode-alist '("
.h
'" . ejb/c-c++-header))

(defun ejb/c-c++-toggle () "Toggles a buffer between c-mode and c++-mode." (interactive) (cond ((string= major-mode "c-mode") (c++-mode)) ((string= major-mode "c++-mode") (c-mode)))) [frame=lines,linenos=true]common-lisp (setq c-basic-offset 4) (setq c-default-style '((java-mode . "java") (awk-mode . "awk") (other . "kr"))) (setq c-doc-comment-style '((c-mode . javadoc) (java-mode . javadoc) (pike-mode . autodoc)))

(defconst my-cc-style '("cc-mode" (c-offsets-alist . ((innamespace . [0])))))

(c-add-style "my-cc-mode" my-cc-style) [frame=lines,linenos=true]common-lisp (use-package ccls :ensure t :after lsp-mode :hook ((c-mode c++-mode) . lsp))

(use-package clang-format :ensure t :bind (("C-M-<tab>" . clang-format-region)))

(use-package astyle :ensure t :when (executable-find "astyle"))

# Julia

### julia mode

[frame=lines,linenos=true]common-lisp (use-package julia-mode :ensure t) ;; Snail requires vterm (use-package vterm :ensure t :config (setq vterm-always-compile-module t))

(use-package julia-snail :hook (julia-mode . julia-snail-mode))

### lsp-julia

[frame=lines,linenos=true]common-lisp (use-package lsp-julia :hook (julia-mode . (lambda () (require 'lsp-julia) (lsp))) :config (setq lsp-julia-default-environment " /.julia/environments/v1.6"))

## Python

### lsp-jedi

[frame=lines,linenos=true]common-lisp (use-package python-mode :ensure t :hook (python-mode . lsp-deferred))

### jedi-server for auto-completetion

[frame=lines,linenos=true]common-lisp (use-package jedi :ensure t :init (add-hook 'python-mode-hook 'jedi:setup) (add-hook 'python-mode-hook 'jedi:ac-setup))

## Gnuplot

## Auto-completation

[frame=lines,linenos=true]common-lisp (use-package auto-complete :ensure t :init (progn (ac-config-default) (global-auto-complete-mode t) ))

## Company

Taken from `https://cestlaz.github.io/posts/using-emacs-45-company/`
[frame=lines,linenos=true]common-lisp (use-package company :ensure t :config (setq company-idle-delay 0) (setq company-minimum-prefix-length 1) :init (global-company-mode t))

(use-package company-box :ensure t :hook (global-company-mode . company-box))

(use-package company-irony :ensure t :config (add-to-list 'company-backends '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-auto-setup-compile-options))

(use-package irony-eldoc :ensure t :config (add-hook 'irony-mode-hook 'irony-eldoc))

# External

## Elcord

Showing emacs as discord status. [frame=lines,linenos=true]common-lisp (use-package elcord :ensure t :config (setq elcord-use-major-mode-as-main-icon t) (setq elcord-display-buffer-detail 'nil) (setq elcord-refresh-rate 2) :init)

(global-set-key (kbd "C-c d") 'elcord-mode)