Emacs Config

will@wmedrano.dev

May 28, 2025

Contents

1 Packages

1.1 Melpa

Add Melpa to the package manager. Melpa contains many popular Emacs packages.

```
(require 'package)
(add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t)
```

Counting the default package archives, the following package archives are available.

gnu	`https://elpa.gnu.org/packages/`
nongnu	`https://elpa.nongnu.org/nongnu/`
melpa	`https://melpa.org/packages/`

For this Emacs setup, the following packages are required:

```
(setq-default package-selected-packages
              '(doom-themes
                doom-modeline
                ivy
                counsel
                company
                rust-mode
                htmlize
                magit))
```

Package archives must be manually refreshed or fetched with `M-x package-refresh-contents`. All packages added to the `package-selected-packages` variable can be installed with `M-x package-install-selected-packages`. Packages can also be installed on a one-off basis interactively with `M-x package-install`.

In some cases, package installation may fail with "... not found". This likely means that the package archives point to an old (and non-existant) version of the package. The package definitions can be updated by running `M-x package-refresh-contents`. Packages may also be upgraded all at once with `M-x package-upgrade-all`.

1.2 Initialize

Initialize the package archive. This makes all previously installed packages available.

```
(package-initialize)
```

2 Startup

Disable the default Emacs startup screen. Instead, this displays just the opened file or the `*Scratch*` buffer if no file has been opened.

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

3 File System

3.1 Auto-Revert

Auto revert mode automatically reloads file-based buffer's when their files have been updated.

```
(global-auto-revert-mode t)
```

3.2 Backups & Autosaves

Backup and autosaves may litter the filesystem so we disable them. This is ok as my disk is reliable, I save often, and use version control.

```
(setq-default auto-save-interval 0
              create-lockfiles   nil
              make-backup-files  nil)
```

4 Performance

Increase the amount of memory allowed before garbage collection kicks in. Emacs runs garbage collection when it is over the limit. The limit is equal to the amount of memory that is `actively-used` plus `used-memory * gc-cons-percentage`. Once the limit is reached, Emacs runs garbage collection and recomputes the amount of memory that is `actively-used`.

```
(setq-default gc-cons-percentage 1.0)
```

Run the garbage collector on idle, every 3 seconds. This reduces the chances of a GC run when interacting with Emacs.

```
(run-with-idle-timer 3 t #'garbage-collect)
```

5 Appearance & Feel

5.1 Remove Clutter

Remove the menu bar and tool bar.

```
(menu-bar-mode -1)
(tool-bar-mode -1)
```

Disable the scroll bar. The functionality is ok sometimes, but it clashes with the theming.

```
(scroll-bar-mode -1)
```

5.2 Lines

Scroll conservatively values above 100 cause Emacs to scroll the minimum number of lines required to get the cursor in position. The default value of 0 causes Emacs to recenter the window.

```
(setq-default scroll-conservatively 101)
```

Display line numbers for text buffers. This can be toggled in an individual buffer with `M-x display-line-numbers-mode`.

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

Highlight the currently selected line. This can be toggled in an individual buffer with `M-x hl-line-mode`.

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

5.3 Color Scheme

Use the `doom-dracula` theme from the Doom Themes package.

```
(load-theme 'doom-dracula t)
```

5.4 Modeline

Use Doom Modeline to display a nicer modeline. Mainly, it:

- Uses more icons.
- Displays a minimal amount of information while still keeping important information such as:
  - Syntax errors
  - Version control information

```
(doom-modeline-mode t)
```

6 Editor Completions

Editor completions refers to auto complete done within the editor context, as opposed to code. Editor completion is used to complete prompts for things such as selecting a file, buffer, or command.

6.1 Ivy

Editor completions are displayed using the Ivy package. This provides a huge improvement over the default built-in Emacs completion.

```
(ivy-mode t)
```

6.2 Counsel

Counsel provides functions that wrap ivy completion with some extra features. For example, `counsel-M-x` is an `M-x` replacement that also displays a keybinding if there is an active keybinding for the particular function.

```
(counsel-mode t)
```

Enabling `counsel-mode` makes the `counsel-mode-map` keymap active. This keymap defines several rebinds.

However, it does not provide a rebind for `counsel-switch-buffer`. We make this our default (interactive) switch buffer command as it allows previewing the contents of a buffer before switching.

```
(define-key counsel-mode-map (kbd "C-x b") #'counsel-switch-buffer)
```

7 Version Control

7.1 Git/Magit

Magit provides an Emacs interface for Git. This involves things such as viewing diffs, staging, comitting, branching, and many other things. The Magit documentation provides a lot (too much) information on how to use Magit.

For a quickstart, try running `M-x magit` to bring up the magit status buffer and pressing `?` to see all the commands.

8 LSP

8.1 Background

The LanguageServerProtocol defines a way for a language server to communicate programming language specific information for a project to an IDE(Emacs). The protocol defines things such as syntax checking, autocomplete, and code formatting.

8.2 Updating Eglot Package

Eglot is included in Emacs. However, Eglot can be upgraded to the latest version with `M-x eglot-upgrade-eglot`.

8.3 Obtaining Language Servers

Eglot is configured to run the most popular language servers by default. However, they must still be installed on the system. Some popular language servers include `rust-analyzer` for Rust.

8.4 Enabling Eglot

Eglot can be manually enabled on a buffer with `M-x eglot`. To enable it automatically, you may call `eglot-ensure` on the buffer automatically through hooks.

```
(add-hook 'rust-mode-hook #'eglot-ensure)
```

9 Formatting

9.1 Tabs

Emacs uses a combination of tabs and spaces when auto-indenting. This pleases neither the spaces nor tabs crowds. Tabs are disabled to prevent the mixed use, though opinionated languages will still find a way to use their correct default. For example, Go will still use tabs when indenting.

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

Use a default tab width of 4 spaces.

```
(setq-default tab-width 4)
```

9.2 Line Width

Set a target line width of 80. Contents of a "paragraph" may be made to follow the target line width through `M-x fill-paragraph` (default keybind `M-q`) or a highlighted region with `M-x fill-region`.

```
(setq-default fill-column 80)
```

Some languages have a different target line length.

```
(defun fill-column-100 ()
  (setq-local fill-column 100))

(add-hook 'rust-mode-hook #'fill-column-100)
```

9.3 Delete Trailing Whitespace

Trailing whitespace is usually unintended. These are whitespace characters hanging at the end of the sentence or newlines/whitespace at the end of the file. Trailing whitespace can be automatically deleted before save.

```
(add-hook 'before-save-hook #'delete-trailing-whitespace)
```

9.4 Language Specific Autoformat

Eglot provides 2 functions for formatting.

- `eglot-format` - Formats the selected region.
- `eglot-format-buffer` - Format the current buffer.

However, running these functions interactively is not needed as we can automatically run `eglot-format-buffer` before save.

```
(defun eglot-maybe-format-buffer ()
  (when (eglot-managed-p) (eglot-format-buffer)))

(add-hook 'before-save-hook #'eglot-maybe-format-buffer)
```

10 Code Auto-Complete

Code auto-complete is handled by the Company package. Company is an autocompletion frontend that comes with many backends. Company comes with a lot of built-in backends and usually selects the best choice among them for auto-complete suggestions. One of the more useful backends is the Eglot backend which is automatically used if the buffer has Eglot mode enabled.

Company mode is usually fine to enable globally. If the buffer doesn't have a suitable backend, then it does nothing.

```
(global-company-mode t)
```

11 Syntax Checking

11.1 On The Fly Syntax Checking

Syntax errors are surfaced by the built-in Flymake package. The Flymake package provides the frontend and several backends.

The most common backend for Flymake is Eglot. Additionally, Eglot automatically enables Flymake on Eglot managed buffers so there is no setup on that front.

11.2 Custom Compile

`M-x compile` and `M-x project-compile` can be used to run a command. `compile` runs in the current directory while `project-compile` runs the command in the projec's root directory, where a project is often defined as a version controlled (like Git) repo.

The outputs of the command will be displayed in a buffer named "**compilation**". By default, pressing `n` and `p` can be used to go through all detected errors in the buffer. For other keybindings, run `M-x describe-keymap` to check `compilation-mode-map`.

11.2.1 Error Detection

`M-x compile` has some built in mechanisms to detect errors. However, some packages, like `rust-mode`, add new patterns. For these patterns to be added, the package has to be loaded. Packages are often lazily loaded or can be manually loaded with something like `M-: (require 'rust-mode)`.

12 Languages

12.1 Rust

Rust is not built into Emacs so we install the Rust Mode package.

Beyond that, there is not much to Rust as most of its functionality comes throught Eglot + the `rust-analyzer` LSP.

12.2 Org Mode

Enable syntax highlighting for exported material html. Note that this will use the currently active theme. This requires the htmlize package.

13 Elisp Concepts

13.1 Interactive Commands

Interactive functions that can be run "interactively". Here, interactively maens that they can be run through `M-x`. Interactive functions are defined by adding (interactive) in their function definition.

```
(defun my-function ()
  "Do a thing."
  (message "Hello World"))

(defun my-interactive-function ()
  "Do a thing."
  (interactive)
  (message "Hello World"))
```

Interactive functions also have a sophisticated mechanism of querying the user for standard options and passing them as flags. See the Emacs Documentation for "Using interactive" for more details.