Emacs Configuration

Florian Feldmann

September 22, 2019

Contents

1 Configure `use-package`

I use `use-package` to install and configure my packages. My `init.el` includes the initial setup for `package.el` and ensures that `use-package` is installed, since I wanna do that right away.

This makes sure that `use-package` will install the package if it's not already available. It also means that I should be able to open Emacs for the first time on a fresh Debian box and have my whole environment automatically installed. I'm not *totally* sure about that, but we're gettin' close.

Always compile packages, and use the newest version available.

2 Use sensible-defaults.el

Use sensible-defaults.el for some basic settings.

3 Set personal information

1. Who am I?

4 Add `resources` to `load-path`

5 Utility functions

Define a big ol' bunch of handy utility functions.

6 UI preferences

1. Tweak window chrome

   There's a tiny scroll bar that appears in the minibuffer window. This disables that:

   The default frame title isn't useful. This binds it to the name of the current project:

2. Use fancy lambdas

   Why not?

3. Load up a theme

   I'm currently using the "solarized-dark" theme. I've got a scenic wallpaper, so just a hint of transparency looks lovely and isn't distracting or hard to read.

   If this code is being evaluated by `emacs --daemon`, ensure that each subsequent frame is themed appropriately.

4. Use `moody` for a beautiful modeline

   This gives me a truly lovely ribbon-based modeline.

5. Disable visual bell

   `sensible-defaults` replaces the audible bell with a visual one, but I really don't even want that (and my Emacs/Mac pair renders it poorly). This disables the bell altogether.

6. Scroll conservatively

   When point goes outside the window, Emacs usually recenters the buffer point. I'm not crazy about that. This changes scrolling behavior to only scroll as far as point goes.

7. Highlight the current line

   `global-hl-line-mode` softly highlights the background color of the line containing point. It makes it a bit easier to find point, and it's useful when pairing or presenting code.

8. Highlight uncommitted changes

   Use the `diff-hl` package to highlight changed-and-uncommitted lines when programming.

7 Project management

I use a few packages in virtually every programming or writing environment to manage the project, handle auto-completion, search for terms, and deal with version control. That's all in here.

1. `ag`

   Set up `ag` for displaying search results.

2. `company`

   Use `company-mode` everywhere.

   Use `M-/` for completion.

3. `flycheck`

8 Programming environments

I like shallow indentation, but tabs are displayed as 8 characters by default. This reduces that.

Treating terms in CamelCase symbols as separate words makes editing a little easier for me, so I like to use `subword-mode` everywhere.

Compilation output goes to the `*compilation*` buffer. I rarely have that window selected, so the compilation output disappears past the bottom of the window. This automatically scrolls the compilation window so I can always see the output.

1. CSS, Sass, and Less

   Indent by 2 spaces.

   Don't compile the current SCSS file every time I save.

   Install Less.

2. Golang

   Install `go-mode` and related packages:

   Define my `$GOPATH` and tell Emacs where to find the Go binaries.

Run `goimports` on every file when saving, which formats the file and automatically updates the list of imports. This requires that the `goimports` binary be installed.

When I open a Go file,

- Start up `company-mode` with the Go backend. This requires that the `gocode` binary is installed,
- Redefine the default `compile` command to something Go-specific, and
- Enable `flycheck`.

Config for company-go

3. JavaScript and CoffeeScript

   Install `coffee-mode` from editing CoffeeScript code.

   Indent everything by 2 spaces.

4. Lisps

   I like to use `paredit` in Lisp modes to balance parentheses (and more!).

   `rainbow-delimiters` is convenient for coloring matching parentheses.

   All the lisps have some shared features, so we want to do the same things for all of them. That includes using `paredit`, `rainbow-delimiters`, and highlighting the whole expression when point is on a parenthesis.

   If I'm writing in Emacs lisp I'd like to use `eldoc-mode` to display documentation.

   I also like using `flycheck-package` to ensure that my Elisp packages are correctly formatted.

9 Publishing and task management with Org-mode

1. Display preferences

   I like to see an outline of pretty bullets instead of a list of asterisks.

   I like seeing a little downward-pointing arrow instead of the usual ellipsis (. . .) that org displays when there's stuff under a header.

   Use syntax highlighting in source blocks while editing.

   Make TAB act as if it were issued in a buffer of the language's major mode.

   When editing a code snippet, use the current window rather than popping open a new one (which shows the same information).

2. Task and org-capture management

   Record the time that a todo was archived.

3. Exporting

   Allow export to markdown and beamer (for presentations).

   Translate regular ol' straight quotes to typographically-correct curly quotes when exporting.

   (a) Exporting to HTML

       Don't include a footer with my contact and publishing information at the bottom of every exported HTML document.

   (b) Exporting to PDF

       enable latex

       I want to produce PDFs with syntax highlighting in the code. The best way to do that seems to be with the `minted` package, but that package shells out to `pygments` to do the actual work. `pdflatex` usually disallows shell commands; this enables that.

       Include the `minted` package in all of my LaTeX exports.

4. TeX configuration

   I rarely write LaTeX directly any more, but I often export through it with org-mode, so I'm keeping them together.

   Automatically parse the file after loading it.

   Always use `pdflatex` when compiling LaTeX documents. I don't really have any use for DVIs.

   Open compiled PDFs in `okular` instead of in the editor.

5. Wrap paragraphs automatically

   `AutoFillMode` automatically wraps paragraphs, kinda like hitting `M-q`. I wrap a lot of paragraphs, so this automatically wraps 'em when I'm writing text, Markdown, or Org.

10 Editing settings

1. Look for executables in `/usr/local/bin`

2. Save my location within a file

   Using `save-place-mode` saves the location of point for every file I visit. If I close the file or close the editor, then later re-open it, point will be at the last place I visited.

3. Switch and rebalance windows when splitting

   When splitting a window, I invariably want to switch to the new window. This makes that automatic.