Put-Vim-on-Autopilot

I'm always looking for ways make development easier, and making computers do the work is my favorite way of getting to the fun stuff: development. Though the following won't do your job for you, here are some tips for getting vim to do some of the more tedious stuff for you.

Syntax highlighting and automatic indentation

Vim will color and indent the code you type for over 500 different filetypes out of the box! Even though vim does so much on its own, it's good to know how to install support for a new filetype in case the need arises. First, make sure you have syntax highlighting enabled in your ~/.vimrc:

```
syntax on
```

And filetype-specific indentation and plugins enabled:

```
filetype plugin indent on
```

For most filetypes, this is enough, but for newer ones, you might need to add the syntax files and indentation files yourself (at least until someone adds them to vim). For instance, if you wanted vim to highlight and automatically indent slim files, you could add the vim-slim plugin to your ~/.vim/ directory (by extracting the plugin's files to the corresponding folders in ~/.vim/; or by using a plugin manager plugin, like unbundle, pathogen, or vundle).

No-configuration plugins

There are plenty of vim plugins that enhance vim in some way without you having to do anything beyond installing them. Here are a few:

- syntastic: automatic syntax checking
  - Automatically checks the syntax of files when you save them.

  - It depends on external syntax checkers, so make sure you have any desired syntax checkers/compilers/linters installed before using it (see the FAQ for more info).

- pasta: enhances the default paste command
  - Makes the default paste command indent the text you're pasting
  - This is similar to the builtin ‘]p‘ command (docs), but adds some useful differences.

- endwise: completes block statements in vimscript, ruby, and more.
  - When you type 'do' in ruby (and other languages) and hit enter, this plugin adds an 'end' on the next line.

- surround: Easy handling of surrounding quotes, brackets, tags, etc
  - Allows you to change, delete, and add surrounding quotes, xml tags, etc.

- rsi aka "Readline style insertion"
  - More readline-like keybindings for vim

- repeat: Repeat some plugin actions
  - Allows plugins to repeat their actions using the builtin . command.
  - Commands from plugins like vim-surround can't always be repeated with the builtin . command. This plugin provides an interface for those plugins to add support for the repeat command.
    * Note: Additional plugins aren't supported by default, some plugins that allow their commands to be repeated with repeat are described in the README.

- powerline and airline: enhanced status bar for vim
  - Enhances the default status bar in vim.
  - Note: to show the statusbar at all times, add set laststatus=2 to your ~/.vimrc; by default it will be shown when you have more than one window open.

- niji: Highlights matching parenthesis and braces
  - When you're editing code with many matching parens/braces it can be hard to read which parens/braces match up. While indentation helps a lot with this, sometimes it's useful to have a few extra colors.
- rails and rake plugins: makes editing ruby/rails projects easier
  - In addition to adding commands like :Econtroller that ease moving around in projects, these plugins enhance the builtin gf command to work better in those projects.
- paredit: Structured editing parenthesies/quotes/brackets
  - This is a port of the great Emacs ParEdit plugin
  - Helps you keep your parens/etc matched when editing so you can focus on other things (like what you're building).
- stopsign: insert debugging statements (shameless self-promotion)
  - When you type 'dbg', this plugin replaces it with a line of code to stop your program
  - This is a little more useful than the standard abbreviation command because it's not filetype dependant (it will expand to different code depending on the filetype)

Filetype specific settings

You might want some options set only on certian filetypes. By using autocommands and local options you can achieve this easily. For example here's a snippet from my dotfiles that spell checks markdown files, git commit messages, and cucumber files as they're edited:

```
autocmd FileType markdown,gitcommit,cucumber setlocal spell " spellcheck these files by
```

Automatic text formatting

For filetypes and comments where you know you're going to want some automatic formatting (for instance markdown or orgmode files) you can change the compound option formatoptions. All available options (described each by a single character) are described in the documentation.

Conclusion

Though your editor still isn't an automatic code-producing robot, it should feel a little more automated than before.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.