Sublime

Saturday, June 20, 2015 5:18 PM

==
6/20/2016
If a Sublime plug-in ever uses the mouse for something that you don't like, you can edit its Default.sublime-mousemap file to change it. There's probably a way to do this in a more "Sublime" way, e.g. by editing a user copy of this file, but I probably won't be using Sublime for long enough to care about that way.

    Example path: %appdata%\Sublime Text 2\Packages\PlainTasks\Default.sublime-mousemap

==
8/26/2015
Sublime accepts a filename as an argument, and if the filename doesn't exist, it will create it on saving.

This means that instead of something like this:

```
touch test.txt
s test.txt     <-- where 's' is my Sublime shortcut
```

Just do the second step

```
s test.txt
```

Then save the file when you're done.

==
8/18/2015
Writing Plug-ins (for ST2)

You can make a new plug-in from the Tools menu. I will be defined like "class MyCoolPluginCommand(sublime_plugin.TextCommand)". To invoke it, you can launch the console with ctrl+` and type view.run_command('my_cool_plugin')

    This is because Sublime takes TheNameOfYourCommand, drops the "Command", and makes it "the_name_of_your".

    This means you can add a keyboard shortcut by putting a line like this into your settings file:

```
{ "keys": ["keypad_plus"], "command": "my_cool_plugin" },
```

The View class contains all of your selected regions and any text. To see everything that you have selected, you can do this:

```
regions = self.view.sel()
for region in regions:
    if not region.empty():
        print self.view.substr(region) # this prints to the console, ctrl+`
```

Regions are not modifiable. I don't know if this is best, but when I want to update a region, I first save all of the regions to another list, then clear the original list, then add the modified regions back in:

```
# Copy regions first
regions = []
for r in self.view.sel():
```

```
            regions.append(r)

        offset = 0
        for region in regions:
            old_offset = offset

            # Replace some text. "offset" is not necessarily equal to len(new_proto) in the case that tab characters were expanded into spaces.
            offset += self.view.insert(edit, region.a + offset, new_proto)

            new_start_index = region.a + len(class_name) + len('.prototype.') + old_offset
            self.view.sel().add(sublime.Region(new_start_index, new_start_index + 3))
```

A Region's start and end are "a" and "b" or "begin()" and "end()", e.g. new sublime.Region(5,10).a == 5

==
8/17/2015
Extensions that I like using:

- **DocBlockr**: formats comments and automatically uses the jsdoc style.
- **Evaluate**: useful for highlighting mathematical expressions like "2+5" and then evaluating. This lets you easily multiply a whole bunch of rows by 2, for example. I think it's just a Python "eval", so you can also do something like "max(4,5)".
- **FileDiffs**: lets you diff selections and tabs of arbitrary content.
- **Git**: tons of useful git commands right in Sublime (add file, diff all, log, blame, etc.).
    - First, init it by providing the parent of a ".git" folder.
    - Then, you need to be working on a file that is a descendant of that parent folder.
    - Helpful tips:
        - Log All: lets you see the log across all commits, not just for the file that you're working on.
- **GitGutter**: puts a symbol in the gutter when a line has been modified from the git source
- **JSFormat**: because "reindent lines" never works the way I want. Note that you have to put "format_selection": true in the USER settings, not the default settings, otherwise it will always format the whole file.
- ReverseR: this is a plug-in that I made and haven't put on GitHub. It changes "true" to "false", "Yes" to "No", "BEFORE" to "AFTER", etc.
- All sorts of syntax highlighting: PowerShell, JavaScript Next (for ES6 at the time of writing (8/17/2015)), various assembly languages, etc.

To find more extensions, search here: https://packagecontrol.io/

==
8/5/2015
I only *just* figured out how DocBlockr figures out where to divide comments.

```
    // These will
    //
    // be combined
    //
    // when you press alt+Q because there is a space after each "//".
```

vs.

```
        // These
        //
        // won't be combined.
```

===
========

6/20/2015
In all of my time using Sublime, I've never had performance problems until I installed GitGutter. After that, moving lines up and down started acting VERY slow, but only on Windows. I changed two settings and the problems were fixed:

```
  // Live mode evaluates changes every time file is modified,
  // Set false to disable evaluation after each input
  "live_mode": false,

  // When set to true GitGutter runs asynchronously in a seperate thread
  // This may cause a small delay between a modification and the icon change
  // but can increase performance greatly if needed.
  "non_blocking": true,
```

===
========