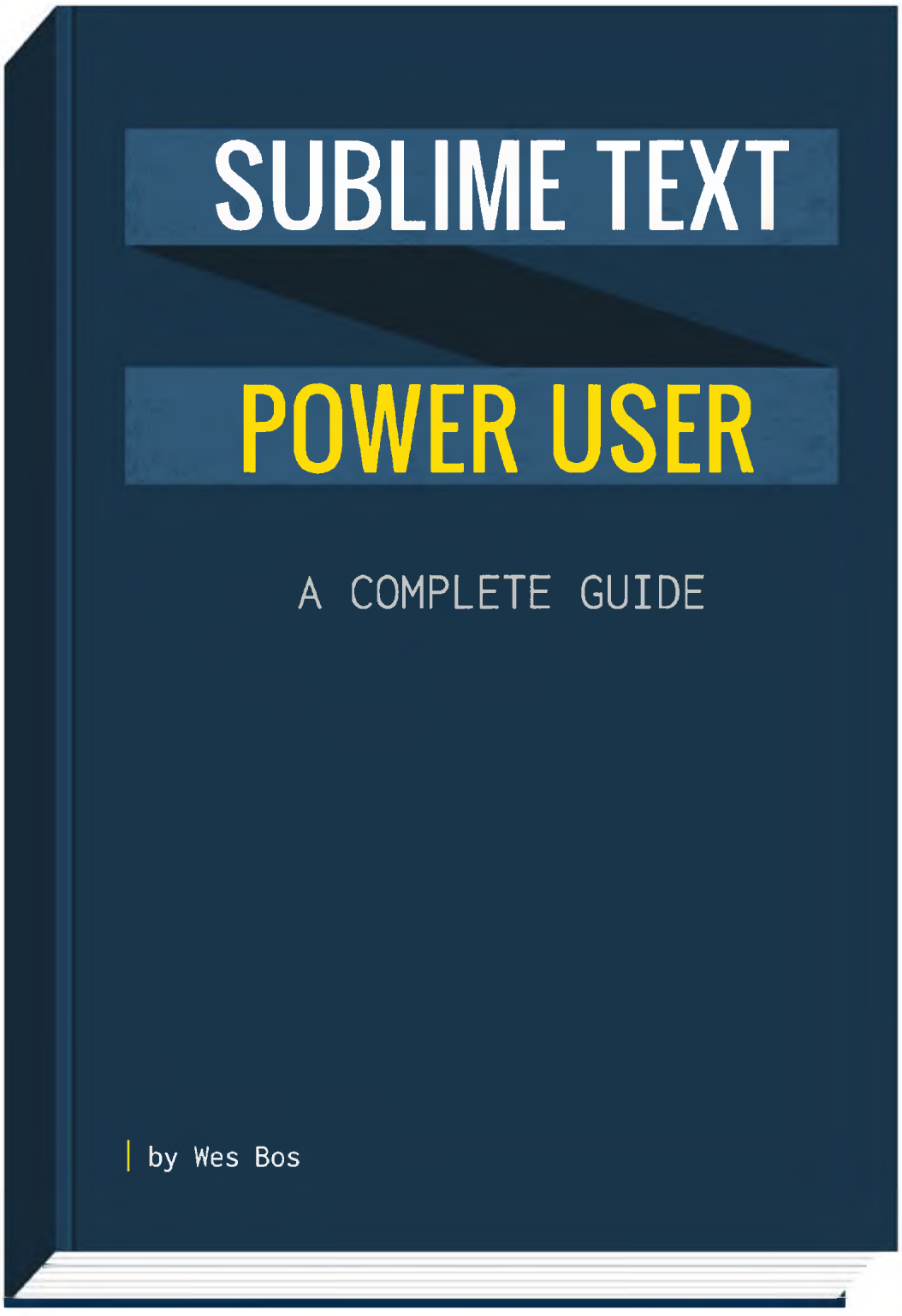


Version 1.2 - Last updated October 14, 2014

        - ANONYMOUS PRO
        - Dejavu Sans Mono
        - Envy Code R
        - Hermit
    - Tweaking Fonts
5. Sidebar
6. Minimap

05. Code Completions and Intelligence

1. Code Hinting / Auto Complete
    - Where Auto Complete Fails
    - Settings
2. SublimeCodeIntel
3. Installing on ST3
    - 1. Use the development branch
    - 2. Clear your CodeIntel cache
4. 3. Fix the language-specific config
5. 4. Be patient

06. Terminal and Command Line Integration

1. OSX
2. Windows
3. Linux
4. Using subl from the command line
    - Arguments
5. Terminal Package

07. Maximizing Screen Real Estate with Multiple Panes and Origami

- Panes Exercise
- Origami
    - Create
    - Destroy
    - Focus
    - Move
    - Focus
    - Resize

2. Moving Between Tabs
    - OSX
    - Windows and Linux

08. Working with Multiple Carets and Selection

1. Replacing Words
    - Quick Find Next / Quick Skip Next
2. Modifying Multiple Lines at Once
3. Another Multi-caret Example

09. Themes and Color Schemes

1. Color Schemes
    - Color Scheme Selector Package
2. Themes

3. Finding Themes
    - Handy Tools

10. Snippets

1. Creating Snippets
    - Content
    - Tab Trigger
    - Snippet Scope
    - Description
    - Saving
2. Finding Snippets

11. Efficient Searching, Finding and Replacing

1. Searching Inside of a Document
2. Search Options
    - Regex Search
    - Case Sensitive
    - Whole Word
    - Show Context
    - In Selection
    - Wrap
    - Highlight Matches
    - Use Buffer
3. Search & Replace Inside Projects and Folders
    - Combining Filters
4. Incremental Find
5. Other Searching Tips

12. Moving Selecting, Expanding and Wrapping

1. Moving Lines and Code Blocks
2. Line Bubbling / Swapping
3. Reindenting Code Blocks
4. Joining
5. Duplicating
6. Deleting
    - Deleting Words
    - Deleting Letters
7. Inserting a line before
8. Wrapping with tags
9. Jump to BOL or EOL
10. Moving to ends and starts of lines and files.
11. Selecting, Jumping & Expanding
    - Jump by Word
    - Select & Expand word by word
    - Select & Expand to certain words
    - Jump by line
    - Select & Expand to Line
    - Select & Expand to Tag
    - Select & Expand to Brackets
    - Select & Expand to Indent

- Select & Expand to Quotes
- Selection and beyond!

13. Code Folding

1. Practice Code
2. Folding Selected Text
   - Block level code folding
3. Fold Multiple blocks at once
4. Folding with arrows
5. Folding element attributes
6. Maintaining Folding State

14. Projects

1. .sublime-project file makeup
   - 1. Folder Settings
   - 2. Settings Overwrite
   - 3. Build Systems
2. Creating and updating projects

15. Mastering Keyboard Shortcuts

1. Negating carpel tunnel
2. Reducing mistakes
3. Becoming a more efficient coder
4. The process of becoming a keyboard shortcut master
5. Referencing Shortcuts
6. What the heck are the ^ ⇧ ⌘ ⌥ ? + SUPER Keys!?
   - ^
   - ⇧
   - ⌘ / Super / ❖
   - ⌥
   - ?
7. Creating Custom Keyboard Shortcuts
   - Default Shortcuts
   - Your Custom Keymap File
     - keys
     - command
     - args
     - context
8. Dealing with Keyboard Shortcut Conflicts

16. Macros

1. Recording a macro
2. Adding a keyboard shortcut
3. Editing Macros

17. Running, Testing and Deploying with Build Systems

1. Creating a build File
   - Selectors
   - Variables
   - Capturing Errors

- Path Issues
- Cross Platform

2. Build Resources

18. Bookmarks

19. Working with Git

1. Sublime + Git Tutorial
   - Gittin' Ready
   - Gittin' Goin'
   - Adds and Commits
   - Diffing
   - Gitting everything else
2. Additional Git Packages
   - Sidebar Git
3. GitGutter
   - Sublimerge
     - Comparing and merging two files
     - Comparing Git Revisions
     - Git, SVN and Mercurial Integration

20. Mastering Emmet

1. Emmet and HTML
   - Elements
   - Classes and IDs
   - Attributes
   - Text
   - Multiple Elements and $ placeholder
   - Nesting Elements
   - Emmet Filters
     - Closing Element Comments
     - Escaping HTML
     - Pipe to HAML or Jade
     - Expand in a single Line
   - Even More
2. Emmet and CSS
   - Numbers and Units
   - Colors
   - More CSS
3. Other Emmet Hot Tips
   - Wrap with Emmet Snippet
   - Encoding / Decoding Data URI
   - Increment/Decrement
   - Lorem Ipsum
   - Matching Pair
4. Other Emmet Treats

21. Workflow & Code Quality

1. Live reload
   - Installing

        - Live reload on mobile devices
    2. Sublime Server
        - Installation and Usage
    3. Live Linting with SublimeLinter
        - Linting your code
            - JavaScript Linting Example
            - CSS Linting Example
        - Linting Settings
    4. Working with FTP / SFTP
        - SFTP Package
            - Remote only server
            - Mapping local to remote
            - SFTP → Filezilla
        - Transmit Doc Send
    5. Tricky trick: Renaming and moving files
    6. Bower Integration
        - Bower Caveats

22. Vim Mode

1. Making Sublime Text act like Vim
2. Sublime Text 2
3. Sublime Text 3
4. Using Vintage Mode
    - OSX 10.7+
5. What's not included

23. Language Specific Tweaks

1. CSS
2. LESS, SASS and Stylus
    - Syntax Highlighting
    - Helpful Tools
3. Coffeescript
4. Templating: HAML, Slim, EJS, Jade
5. JavaScript
6. jQuery
7. Node.js
8. PHP
9. Wordpress
10. Python
11. Ruby

24. Must have Add-on Packages

1. Emmet
2. Autofilename
3. HTML + CSS + JSON Prettifyer
4. Sidebar enhancements
    - Open with...
    - More Features
5. JSHint Gutter
6. Alignment

7. Bracket Highlighter
8. Writing Markdown with Sublime Text
    - Syntax Highlighter
    - Compiling
    - Table of Contents
9. Maintaining State on a file
10. Expand to quotes
11. TODO

25. ## Tip + Tricks Grab Bag

1. Converting Case
2. Code Comments
3. Sort, Reverse, Unique and Shuffle
4. Distraction Free / Fullscreen Mode

26. ## fin

1. Updates
2. Have a question?

Sublime Text Power User

1.1 About The Author

Wes Bos is an independent full stack web developer, author, educator and speaker. He works with startups and large companies alike consulting on HTML5, CSS3, JavaScript, Node.js and WordPress projects. Wes is also a lead instructor at Ladies Learning Code and the Toronto based HackerYou where he leads part-time courses and bootcamps on everything from beginner HTML to advanced JavaScript.

We lives in downtown Hamilton, Ontario where he works with his wife Kait Bos from their 140 year old home under the company BosType.

1.2 Reviewers

This book would not have been possible without the hard work of the reviewers listed below. I want to say a big thanks to everyone who gave this book a once-over suggesting edits and correcting mistakes along the way.

- Ricardo Vazquez - vazquez.io / @iamrvazquez
- Alyne Francis - alynefrancis.com / @alynejf
- Darcy Clarke - darcyclarke.me / @Darcy
- Simon W. Bloom - simonwbloom.com / @SimonWBloom

1.3 Introduction

Welcome and thank you for purchasing Sublime Text Power User!

As a developer, your editor is one of the most powerful tools that you have to increase productivity and develop higher quality code. Learning to master your editor is no different than learning to master your programming of choice. By reading this book and learning to take advantage of every feature in Sublime Text, you are committing to becoming a better and more well rounded developer.

Over the past 3.5 years, I've been a full-time Sublime Text user with a constant hunger for getting to know the editor better. This is the book I wish I would have had.

Whether you are new to programming or a seasoned vet, get ready for a torrent of everything from neat little tricks, to foundational workflow ideas.

1.4 Mac, PC, Linux

Most of the features and functionalities of Sublime Text are available on Mac, PC and Linux. I have written this book from the perspective of a Mac user, but provide insight for Windows and Linux users when the operating systems are inconsistent.

Jump Around!

This book is written in small easy to digest sections and can be read from start to finish, but doesn't necessarily need to be. With the exception of the first few chapters, feel free to jump around to the parts that you need to reference.

Getting Started

2.1 Version 2 or 3?

The current stable release of Sublime Text is version **2.0.2**. That said, version **3** was released in early 2013, to much acclaim, and is the preferred build at this time. Although many add-ons and packages were initially incompatible with version 3 many projects have moved to support it.

If you are currently using version 2, it is highly recommended that you migrate to Sublime Text 3. A migration guide can be found at http://wesbos.com/migrating-to-sublime-text-3/.

2.2 Installing Sublime Text

This is the obvious first step for getting started with Sublime Text. There are three versions of Sublime Text available at http://www.sublimetext.com/3dev - **Stable**, **Dev** and **Nightly**. It's recommended that you run the **Dev** version which delivers semi-frequent updates. These releases are also fairly stable and give you access to some of the new and exciting features that may not make it into the Stable version for a number of months. If you want to live on the edge, using the **Nightly** version may be more your style. Since Sublime Text is so dang fast, it takes just seconds to upgrade to the latest build.

2.3 Installing Package Control

Like most editors, Sublime Text supports addons/plugins/extensions called *packages* that extend the native functionality of the editor. We will dive into the powerful package ecosystem more thoroughly in later Chapters of this book. For now, to get our environment setup, we need to install something called **Package Control**.

Created by the very talented Will Bond, this package manager allows you to easily add, edit and delete your Sublime Text packages right from the editor - no fiddling with downloads, updates or versions. There is a one time install to get the Package Manager running.

1. Head on over to https://sublime.wbond.net/installation and copy the install code on the page

> **Note:** Make sure to check the version of Sublime Text you're running. You can find this out by clicking on "Help > Documentation" and the appropriate version 2 or 3 support guide will be shown.

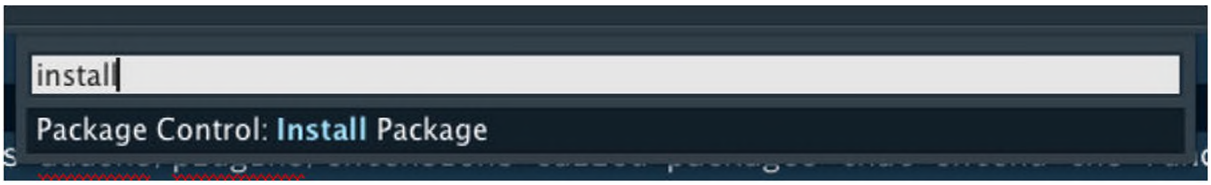
1. Open the console by hitting the key combination `Ctrl` + `\`
2. Paste the previously copied text into the prompt followed by `enter`
3. Restart your editor

This process will install the Sublime Package Manager and allows us to install third-party packages.

Installing a package

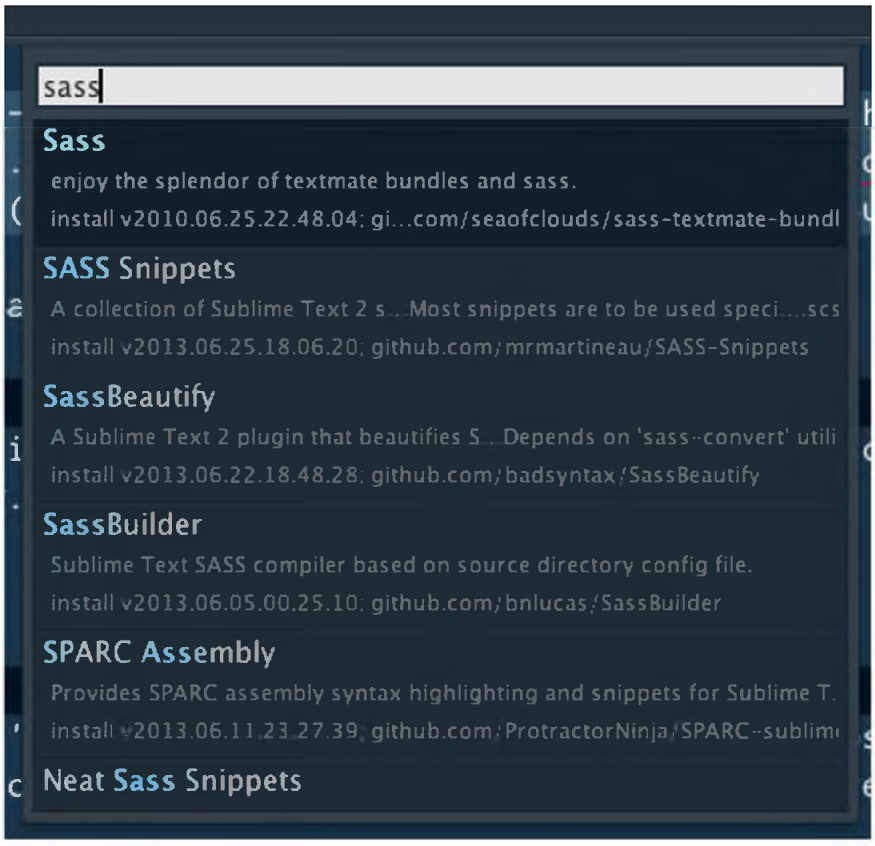
Installing a package with Package Control couldn't be easier. All packages are hosted either on Bitbucket or GitHub. Fortunately for us, package control will interface with these websites so we don't ever have to leave the editor.

Once you have package control installed, via the instructions above, open up the command palette with `⌘` + `Shift` + `p` or by clicking `Tools` → `Command Palette` and type `install`. You should be able to select the item that says `Package Control: Install Package`. If you don't see it, you need to go back to the previous steps and make sure you have installed package control properly.



If this is your first time installing a package or if you haven't installed one in a while, you may have to wait a few seconds while package control fetches the latest list of plugins. You can see that it's working in the lower left hand status bar:

Once it has finished fetching the list, you will be presented with a list of every package available. Simply search for the one you are looking for by typing it's name or scrolling down. You'll be required to do this often in this book; so getting comfortable with this process is a good idea.



Installing Packages Manually

Sometimes you may find a package that isn't in package control and you will need to install it manually; follow these steps in those cases:

1. Install Git on your computer
2. Grab the desired package's URL from it's repo page
3. Open up your computer's terminal or command line and navigate to the packages directory. You can do this by typing `cd` followed by one of the following paths:
    - **Mac**: `~/Library/Application Support/Sublime Text 2/Packages`
    - **Windows**: `%APPDATA%\Sublime Text 2\Packages`
    - **Linux**: `~/.Sublime Text 2/Packages`
4. Clone the package using: `git clone [your-git-url]` .

Your package should now be installed and ready to use!

> **Note:** Package URLs should be located at the bottom right hand side of a GitHub repo or the top right of a Bitbucket repo
>
> **Note:** If you are using Sublime Text 3, change the "2" in the packages directory paths above to "3"
>
> **Cloning Example:** For the Vintageous package, my command would be: `git clone https://github.com/guillermooo/Vintageous.git`

Adding a Repository

As of Sublime Text 3, we now have an easier way to do manual package installations. Using the **command palette** type in "Package Repo". Once there, look for **Package Control: Add repository** and select it. It will then prompt you for the GitHub or Bitbucket URL. Paste in your git URL and Sublime will attempt to install the package for you.

2.4 Onward

That is all there is to set up Sublime Text for now. In the coming Chapters, we will review every personalization possible to make the editor a perfect fit for you.

Getting Comfortable With The Command Palette

In the last Chapter, we briefly touched on the **command palette** as a way to install a package. Let's take a look at how we can do almost anything with this handy tool.

Although most commands can be accessed through Sublime Text's **menu system**, it's always faster, and more savvy, to use keyboard shortcuts or the **commands menu**.

Let's look at some examples that you'd encounter fairly often when developing.

> A **command palette** is often a new concept for users. I myself took time before finally familiarzing and utilizing it to its full extent. Don't make that same mistake. The command palette is one of the most powerful tools in Sublime Text; Helping speed up your workflow significantly.

3.1 Goto Anything

Goto Anything is a very powerful ability to have at your disposal. It works by bringing up a palette that allows you to navigate to any file, line or symbol within your current projects or open files. As you type, Sublime gives you a live preview to help narrow down your search.

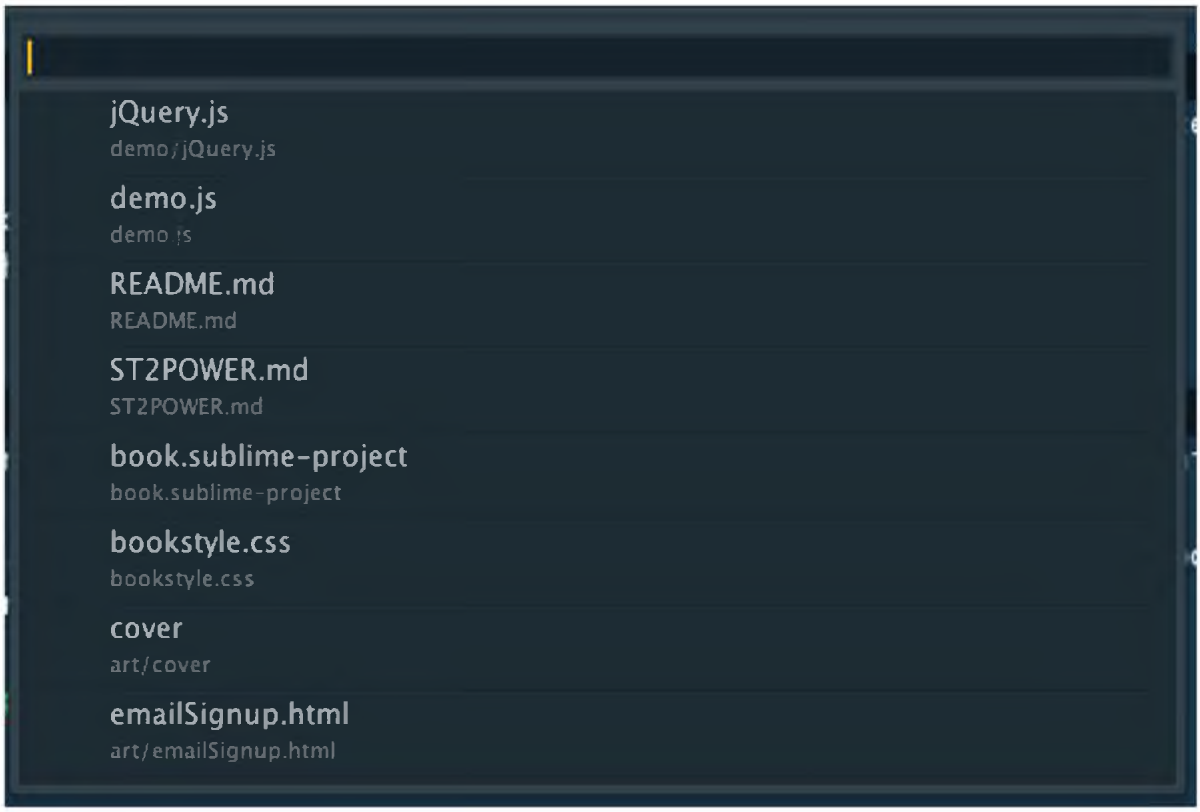
To bring up the **Goto Anything** palette, press `⌘` + `P`. There are also a few more specific shortcuts in the paragraphs below. You can always close this box by pressing `ESC` or the open keystrokes again.

> Windows and Linux users should uyse `Ctrl` in place of `⌘`

Files

When you press `⌘` + `P` you will be presented with a list of files, folders or projects you currently have open. The most recently used files will be at the top of this list. From here you can start to narrow down your search by typing the name of the file, folder or project you're looking for. Using the arrow keys, we can move up and down these items.

You'll notice that Sublime Text also provides a quick preview of files found using this search. Clicking or hitting `enter` on a file in the list of results will open it for you.



Using this shortcut is much quicker then clicking around your sidebar and rooting for the file you want. If you know you are looking for file named `style.scss`, you don't need to spend time drilling down folders, when you can simply pop open **Goto** and start typing.

If you have multiple matches for the same file name, no worries; Sublime helps you out by showing each file's path directly below its name.

Line Numbers

Goto Anything also allows you to jump to a specific line number within a file. When you bring up the palette, start your query with `:` followed by the line number you wish to go to.

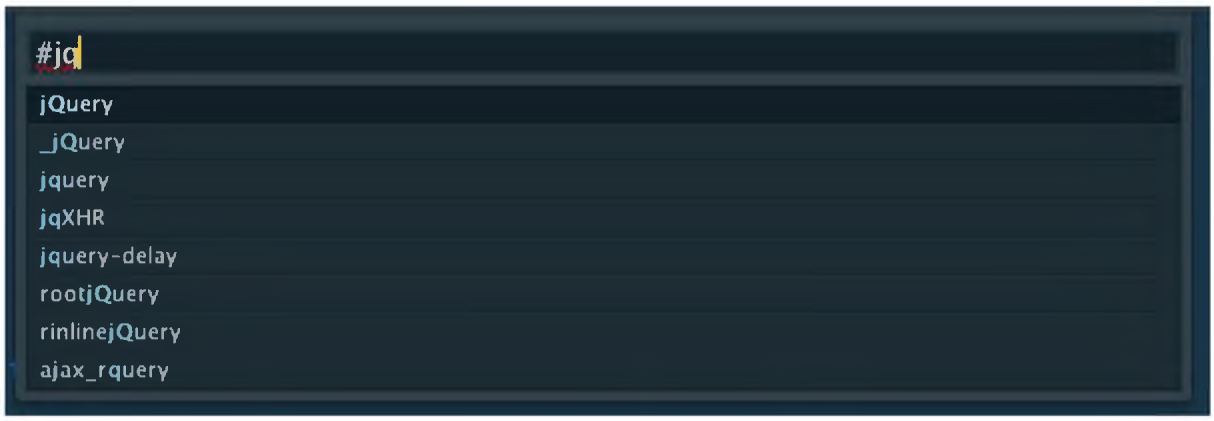
> **Example:** If I'm working on a file that gives me an error on line **235**, I just need to type `:235` into the Goto Anything palette. Better yet, hitting `Ctrl` + `G` will both bring up the Goto Anything palette and pre-populate it with `:`.

Fuzzy Search

If you've never heard of a **fuzzy search** before, you can think of it is a way of searching through documents for an approximate string of text rather than an exact match. I'll leave the complex, algorithmic interpretation out for now; Just know that you don't need to type an entire word or phrase to find a match.

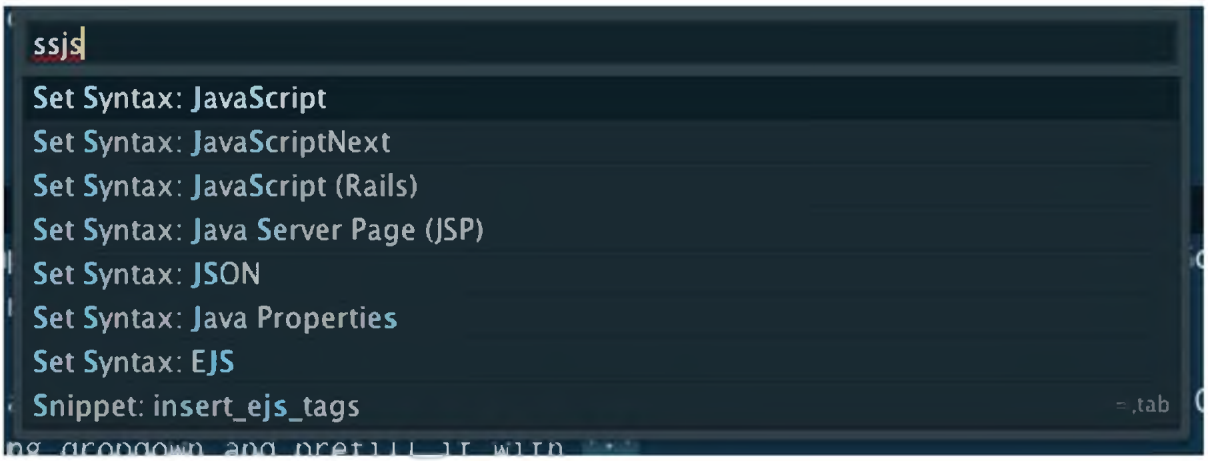
To do a fuzzy search, it's the same idea as with line numbers. The only difference is that you use the pound symbol `#` instead of the colon `:`.

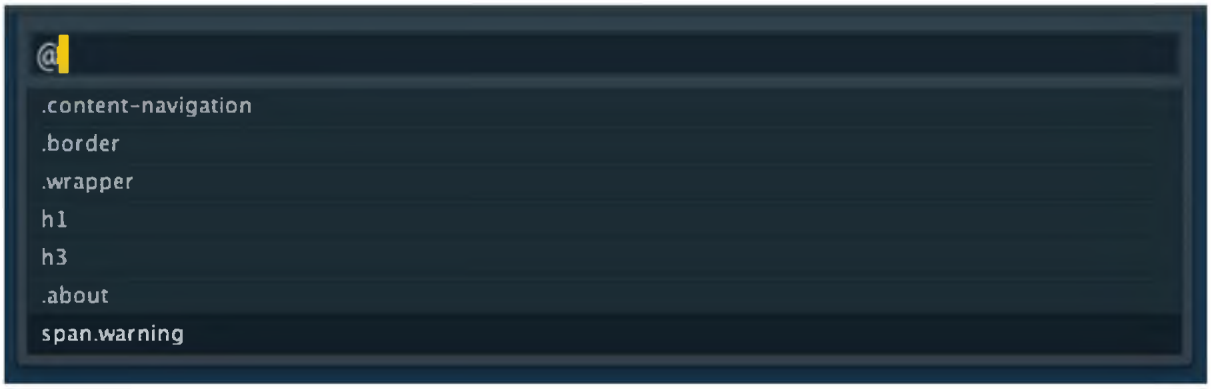
> **Example:** Lets say I had a copy of the jQuery source open and I wanted to find all instances of "jQuery" I'd simply open the Goto Anything palette and type `#jq`; I'll now be able to quickly see and page through the matches from my fuzzy search.



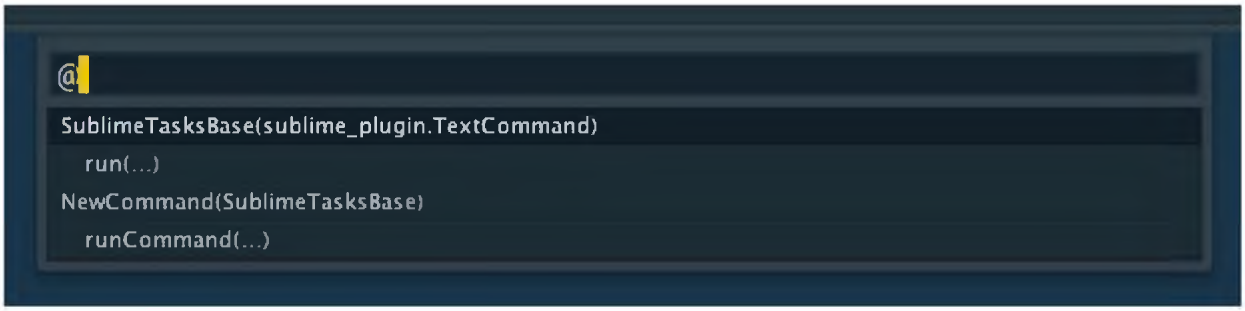
Developers often use this technique to do a quick search of their files to get to a certain line or section of code. It is an easier way to quickly scan a document for symbols or key combinations.

> **Note:** Fuzzy search also works in the command palette for command names. A good example of this would be to quickly change the syntax highlighting of an open document. To do this, hit `⌘ + P` and then type `ssjs`. You can see that the fuzzy search finds and highlights all the commands with the letters `ssjs` in them. You can read more about this in the **command palette section**.





CSS

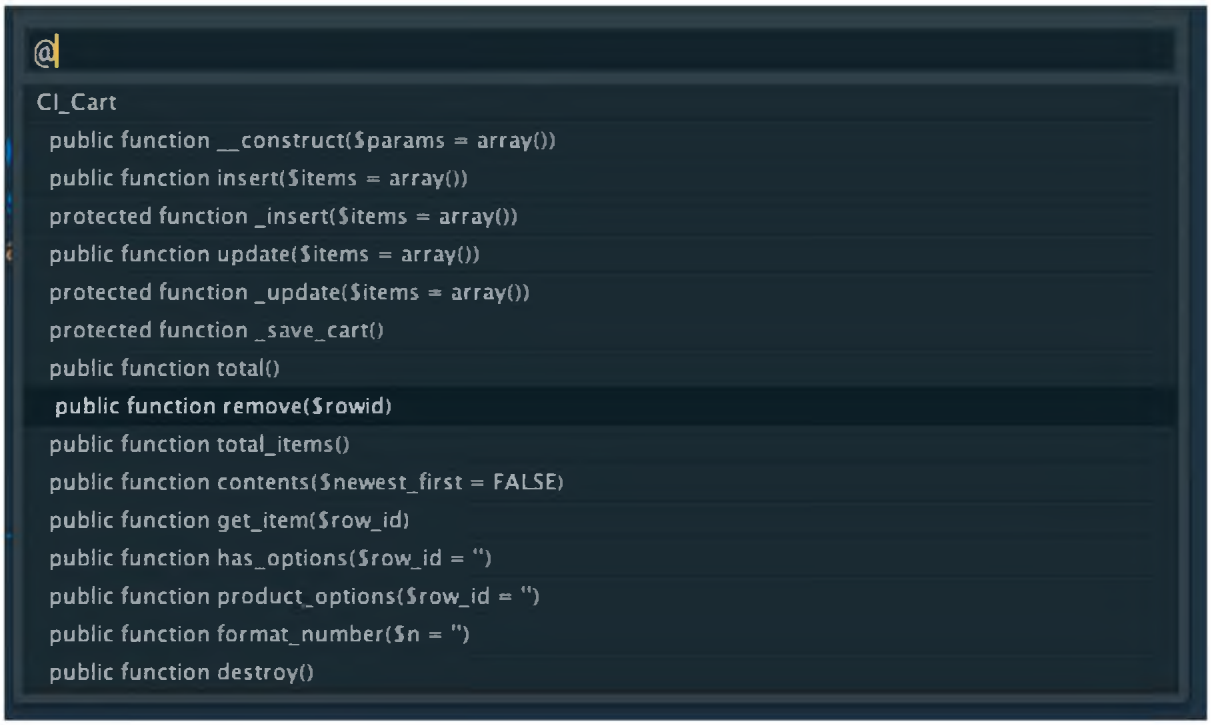


Python

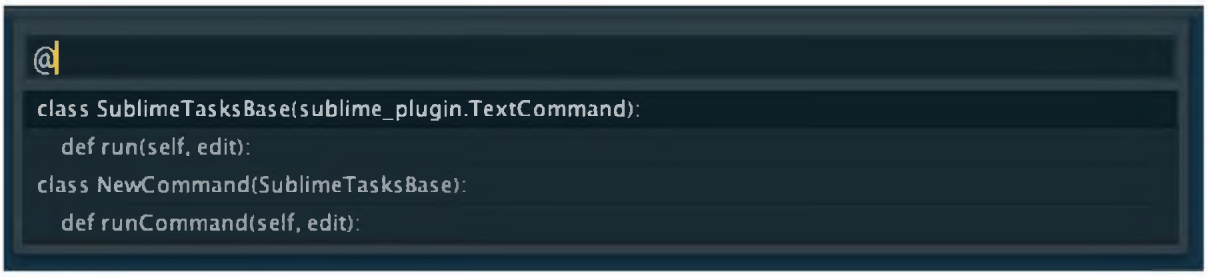
Code & Text Blocks

You're probably familiar with the pain associated in searching for a specific function, class, or chunk of text in a file. Sublime lets you quickly navigate through these blocks of text and code by opening the Goto Anything palette (`⌘` + `P`) and typing `@` (or just `⌘` + `R`). You're able to use the previously noted fuzzy search to filter this list for specific functions, classes or blocks of text.

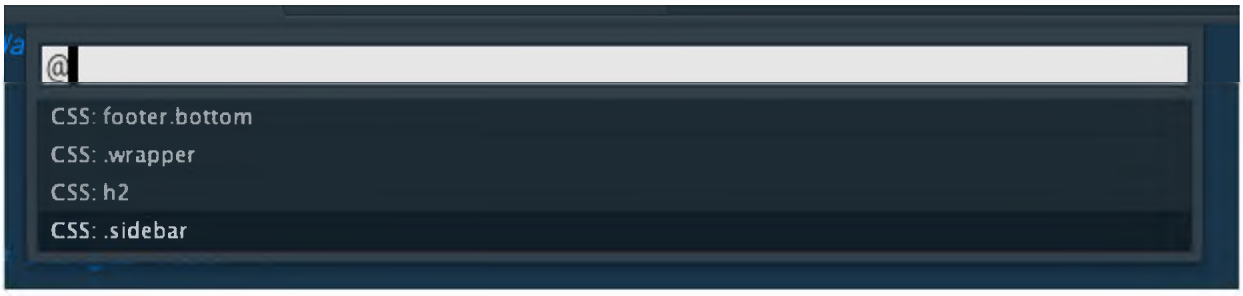
Example viewing a PHP file:



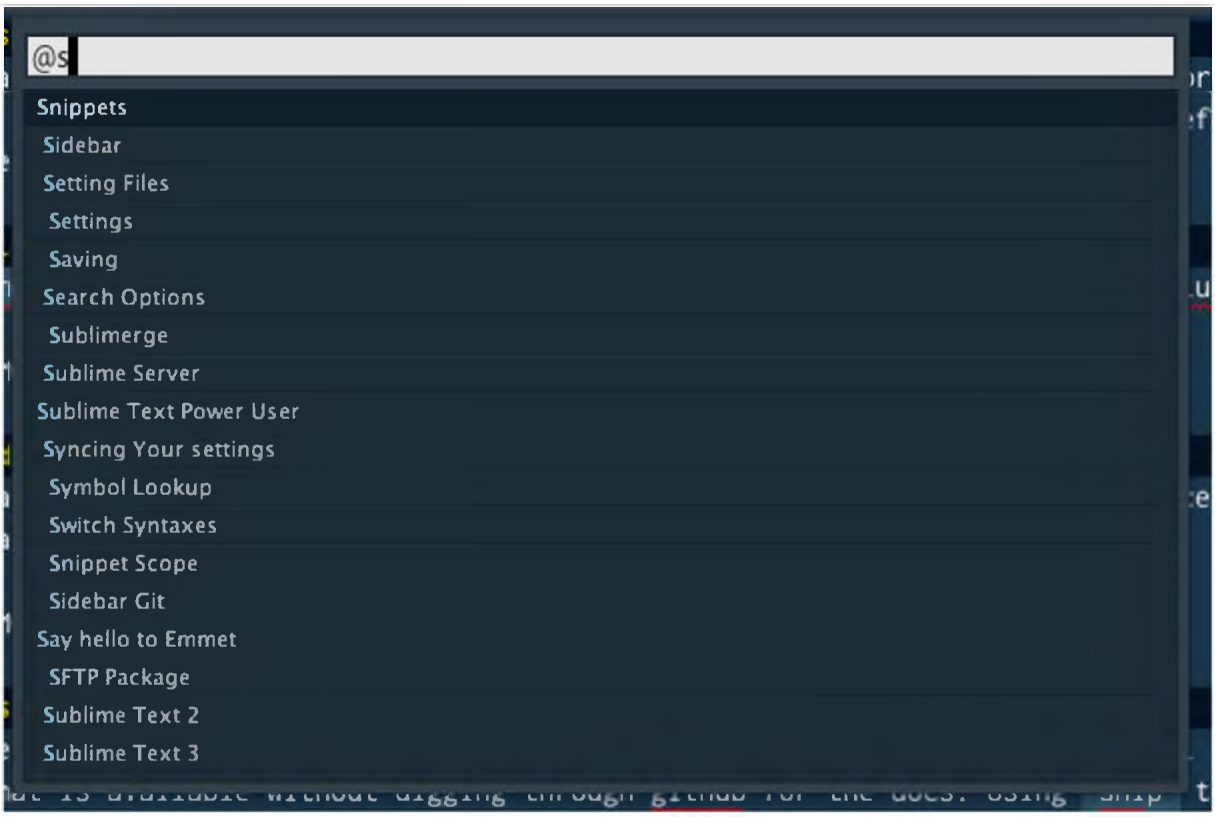
Example viewing a Python file:



Example viewing a CSS file:

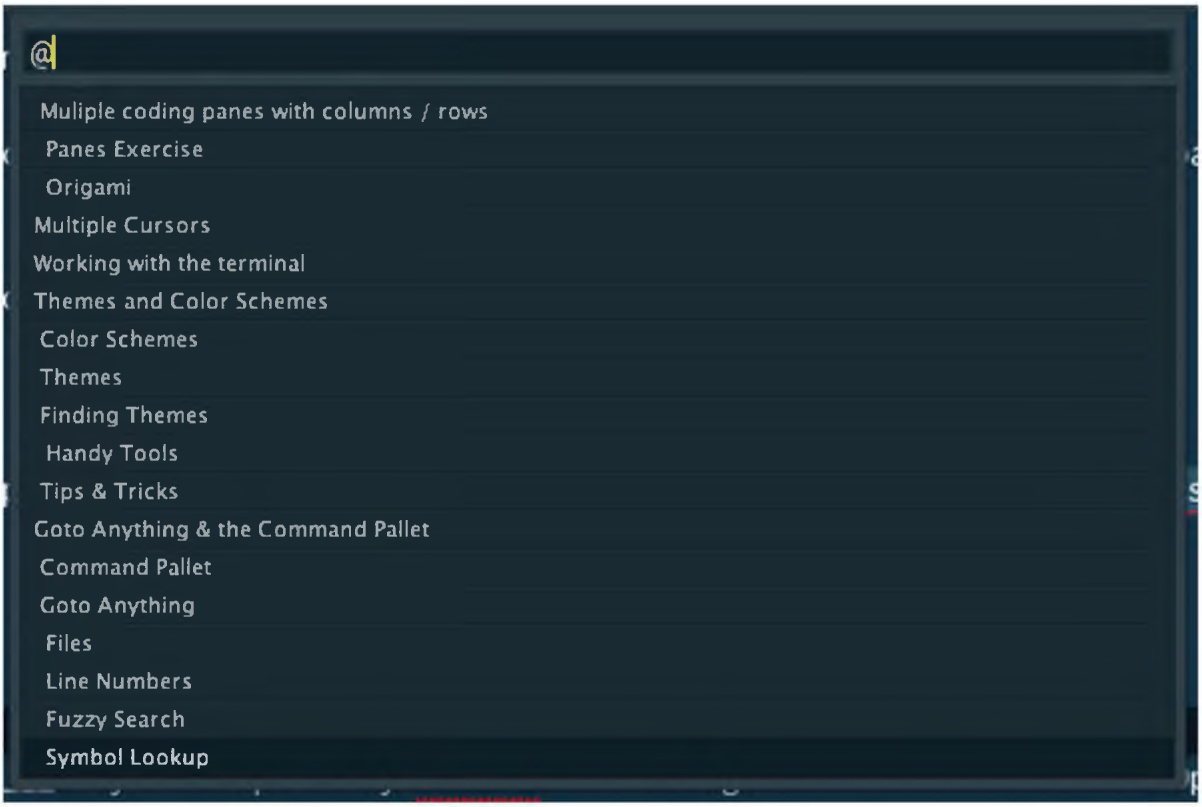


Example viewing a Markdown file:



As you can see, this works great in pretty much every language. For instance, in a JavaScript file you will see a list of all available functions whereas in a CSS file all selector classes and IDs are shown.

The implementation for Markdown is also extremely helpful as it allows you to jump between sections of content, as seen below:



Even better, **Sublime Text 3** has introduced the ability to use this feature across your entire project and open documents. This means if you created a function or class in one file, you can quickly find or reference it while viewing another.

> **Note:** You can use `⌘` + `R` to open Goto Anything and pre-populate the search with `@`

Chaining Commands

All of the features discussed so far are great as standalone features but become even more powerful when used together.

For example, if you had a file in your project named `plugin.py` and wanted to find that file, open it and then find a specific function within that file named `runCommand()` you could by simply typing something similar to `plug@rc`.

Excluding Files & Folders From Search

There may be files or folders that you do not want to see in your Goto Anything searches. Things like compiled JavaScript from CoffeeScript, compiled CSS from SASS or any other assets. If you'd like to exclude specific files you can by modifying your user settings file (*Preferences → Settings - User*) and defining the `binary_file_patterns` property.

```
"binary_file_patterns": [".DS_Store", ".gitignore", "*.psd"]
```

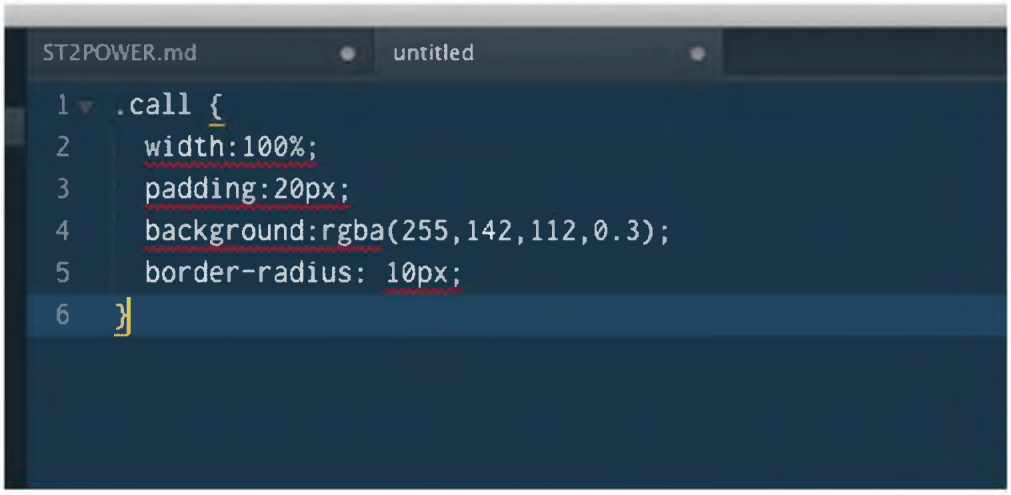
The example above ignores any `.DS_Store`, `.gitignore` or `.psd` files. To exclude entire folders, you can append a forward slash `/` to the end of a folder name.

```
"binary_file_patterns": ["node_modules/", "vendor/", "tmp/"]
```

> **Note:** You may be tempted to use the `file_exclude_patterns` or `folder_exclude_patterns` properties to exclude files from Goto Anything instead of `binary_file_patterns`; While these both do the job, they also happen to remove those files and folders from the sidebar - which may not be a desirable outcome.

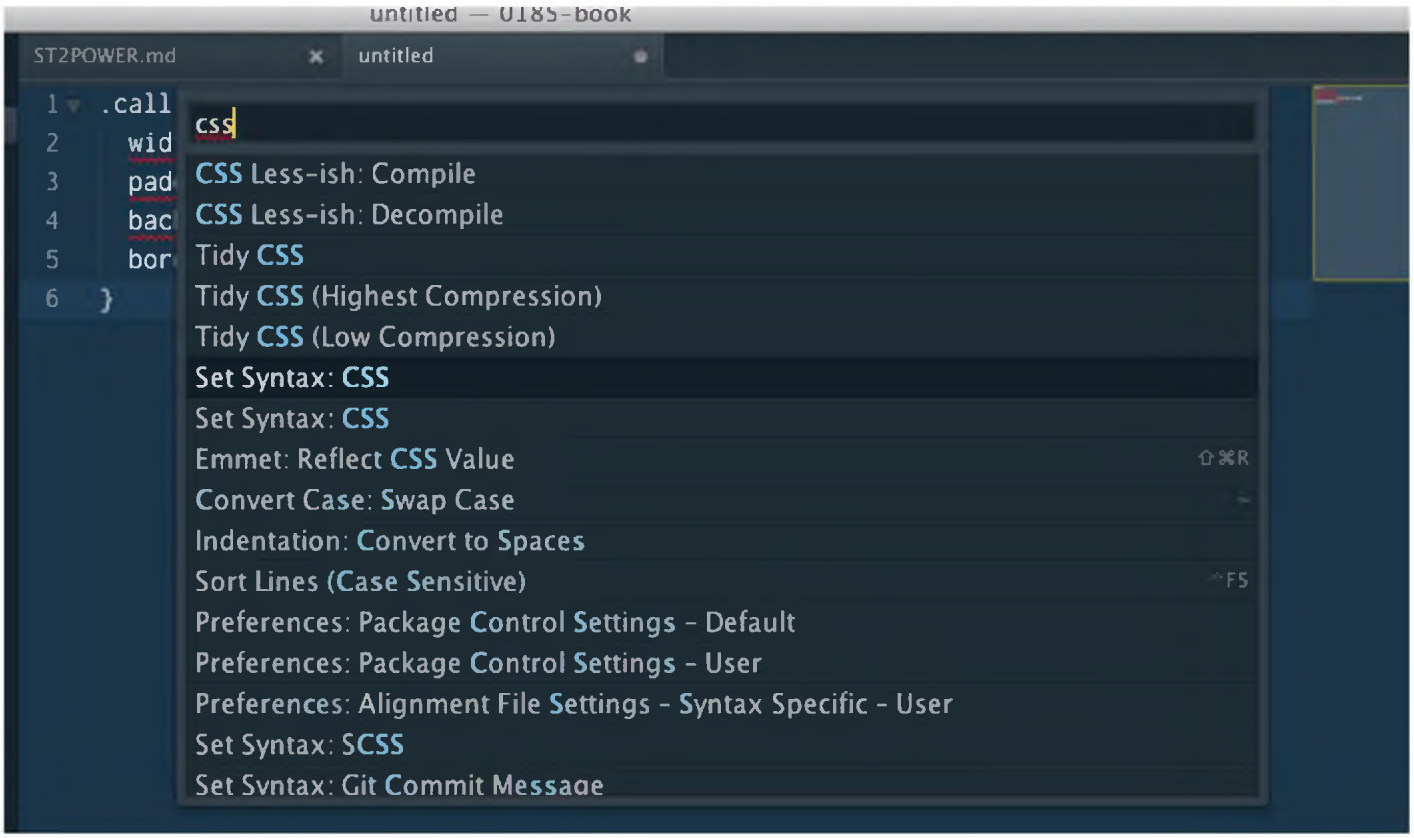
3.2 Changing Syntax

Often you find yourself opening new tabs to quickly jot down temporary copy or code. Unfortunately, your tab has no knowledge of what language you're typing in and none of your packages will kick in. Unless you save the file, which is not your intention, you will have to set the document's language manually to see proper syntax highlighting.

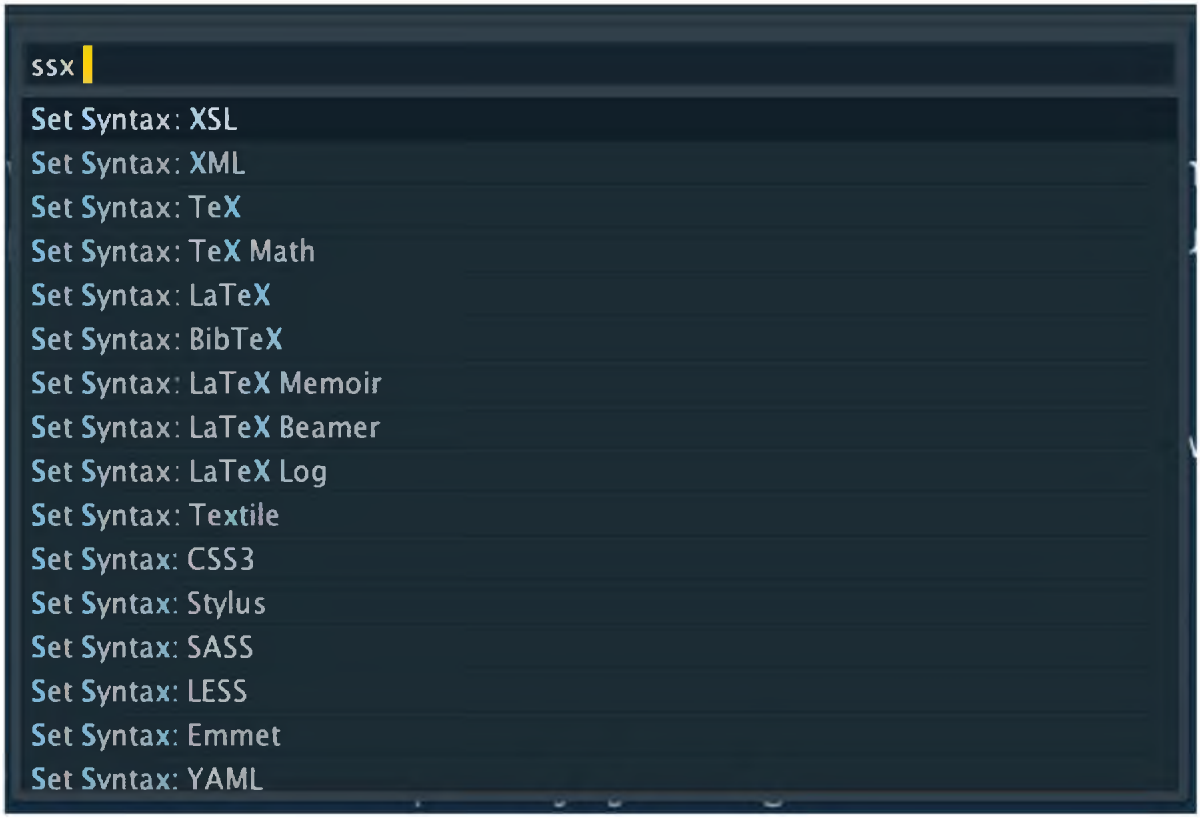


Fortunately, you can see what syntax the tab is currently interpreting your code as in the bottom righthand corner. Clicking on that name will allow you to choose between the list of supported syntaxes.

Eliminating unnecessary use of your mouse is always top of mind; So, opening the command palette and using the *fuzzy search* feature, noted earlier in this Chapter, is the ideal way to find and set your desired syntax.



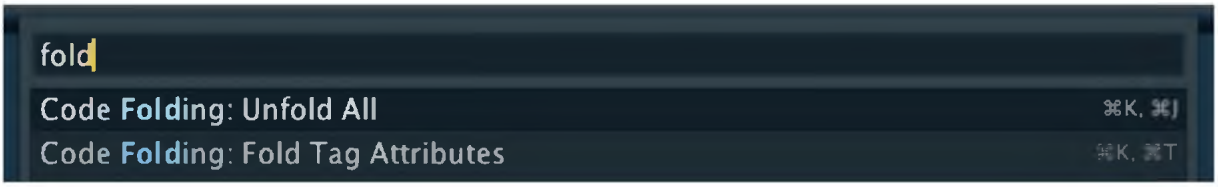
> **Note:** Using a *fuzzy search* that contains `ss` or `syntax` helps quickly narrow down syntax highlighting commands as they all begin with *Set Syntax*.



Keyboard Shortcuts

If you're having trouble remember certain keyboard shortcuts you can, again, utilize the command palette for this.

> **Example:** Forgot what the code folding and unfolding commands were? Type `fold` and the corresponding keyboard shortcut reference will be neatly displayed to the right of the search results.



Snippets

Most of your snippets should have keyboard shortcut associated with them but when you install a package with snippets, it can be hard to know what is available without digging through GitHub repositories, wikis or docs. Use `snip` to filter your snippets.

Practice

As previously mentioned, you can open the command palette by clicking on `Tools` in the top menu and then selecting `Command Palette`. However, the whole point of the command palette is to limit your use of the mouse. That known, remembering the command palette keyboard shortcut is essential to being a productive Sublime Text user.

Take a moment to memorize and try out the command palette keyboard shortcut a few times. `⌘` + `Shift` + `p` on OSX and `Ctrl` + `Shift` + `p` on Windows and Linux. If you would rather use a different keyboard shortcut, check out Chapter 15.

Editor Settings & Customization

4.1 Settings Files

Customizing your text editor to exactly suit your needs is one of the best things you can do for your productivity as a developer. Unlike most editors, there aren't fancy GUI menus that allow you to pick and choose settings - there are settings files.

There are two types of settings files, both of which are formatted in JSON.

All settings files are located inside of the Sublime Text packages folder. It's important to know where these files are so you can easily reference them. The easiest way to open this folder is from the menu bar.

`Preferences` → `Browse Packages…` OSX users will find Preferences under `Sublime Text 2`

In here there are quite a few different files and folders. The ones you use to configure Sublime Text are usually found in the `User` folder - this is your folder and will never be overwritten when you upgrade.

Everything from which font is being used and color schemes to the tab sizes and ignored folders can all be set using custom editor preferences.

To find out what all the available settings are, you can take a look at the default file by opening it up via `Preferences` → `Settings - Default`. This file is only used as a reference. Setting custom preferences will override the settings in this default file.

> **Note:** The default settings file may look large but is worth reading through to get an idea of what changes are possible to enhance your work flow.

*.sublime-settings Files

These files are where we specify all of our preferences. We will be doing most of our customization in `Preferences.sublime-settings` which you can bring up in the menu `Preferences` → `Settings - User` or by simply hitting `⌘` + `,` on OSX.

There are a few types of settings files and the editor references them in this order:

1. **Default Settings** (do not edit this file)
2. `Packages/Default/Preferences.sublime-settings`
3. **Platform-Specific Settings** (for those who jump between operating systems)
4. `Packages/Default/Preferences (<platform>).sublime-settings`
5. **User Settings** (where we will make almost all of our edits)
6. `Packages/User/Preferences.sublime-settings`
7. **Syntax-Specific Default Settings** (do not edit this file)
8. `Packages/<syntax>/<syntax>.sublime-settings`
9. **Syntax-Specific User Settings** (for those who wish to have different settings per programming language)
10. `Packages/User/<syntax>.sublime-settings`

If you open up your `Preferences.sublime-settings` (found in `Preferences` → `Settings - User`) you will see a blank settings file like this:

```
// Settings in here override those in "Default/Preferences.sublime-settings", and
// are overridden in turn by file type specific settings.
{
}
```

All of our settings will follow JSON syntax and look something like this:

```
{
"setting_name" : "Setting Value",
}
```

Syntax / Language Specific Settings

For some languages you may want different settings. For example, when editing JavaScript files you might want to use spaces instead of tabs and turn off line wrapping. On the other hand, when writing in Markdown, you still want hard tabs and to see wrap lines. Instead of switching these settings between languages, you can simply use the above reference order and create a new file called `/Packages/User/Markdown.sublime-settings`. Inside you can put the following:

```
{
    "word_wrap": true,
    "translate_tabs_to_spaces": false
}
```

Settings Files JSON Gotchas

A few things to keep in mind if you are new to Sublime Text or JSON formatting in general.

1. Use "**double quotes**" when defining both keys and string values (single quotes are not valid JSON).
2. Each key, value pair requires a comma between them

Example:

```
{
    "setting_name" : true,
    "another_setting" : 12,
    "font_face": "inconsolata"
}
```

> **Note:** Notice that the above JSON doesn't have a comma after the last line? You don't need one so leave it out! This is one of the most common mistakes when editing JSON objects.

.sublime-keymap Files

Sublime and its packages come with some great maps for keyboard shortcuts. These keymap files work very similarly to the settings hierarchy noted above. We'll go into detail on how you can write your own shortcuts, using keymaps, later in Chapter 15.

4.2 Syncing Your Settings

If you have multiple computers that run Sublime Text, you will want to have the same settings and packages on both of them. The two most popular, and automatic, ways to do this is either with Git or Dropbox.

Dropbox is the preferred method as it is done passively and automatically, whereas Git requires actively pushing and pulling.

Each operating system has a different set of instructions which you can find on the Package Control Website. At the end of the day, we essentially need to link the folder on computer 1 to computer 2. Since Dropbox and Sublime Text are in different folders, we can create a *symlink* (symbolic link) which will tie the Sublime Text `/Packages/User` folder to our Dropbox folder.

It's important to note that we don't want to link the entire `Packages/` or `Installed Packages/` directory as it contains the actual code for the packages. We only want to sync the `User/` folder which contains `Package Control.sublime-settings` - the file that tells Package Control which packages should be installed. Package Control will make sure to grab the correct version of each package for your computer and install it.

By syncing the entire `User/` folder, we also make sure the following user preferences get moved over:

- Your master settings file - `Preferences.sublime-settings`
- Your language specific settings files - `CSS.sublime-settings`
- Your custom snippets
- Your custom build tasks
- SFTP server information
- And other package specific information and settings

4.3 Tabs, Spaces & Indentation

Specifying Tabs Or Spaces

By default, Sublime Text uses tab characters with a size of 4. If you prefer to use spaces instead of a tab character, simply add this to your preferences file:

"translate_tabs_to_spaces": true

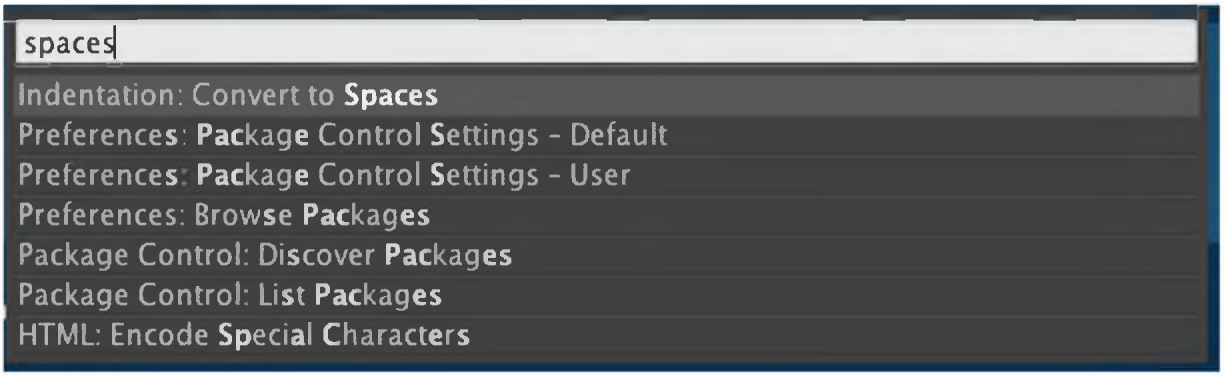
You can also change the tab size or the number of spaces a tab takes up. It defaults for 4 but I prefer to use 2

"tab_size": 2

Converting From Tabs → Spaces Or Spaces → Tabs

You will often run into a file or snippet of code that contains tabs or spacing the isn't in tune with your preferred style. Rather than trying to reformat everything manually, you can easily convert the files spacing preference by doing the following:

1. Open the file
2. Open the Command Palette (`⌘` + `shift` + `p`)
3. Type `spaces`
4. Find **Indentation: Convert to Spaces** in the search results
5. Hit enter



This command will convert the entire document from tabs to spaces. Type **tabs** to find the command to do the opposite.

Detecting Indentation

You may have a personal preference between tabs and spaces, but not every project will share that preference by default; Because of this, Sublime will attempt to detect the tabs or spaces used. If you wish to turn this detection off, you can toggle this following option in your settings file:

```
"detect_indentation": false
```

Detect Settings with Editor Config Package

Preferences such as tabs vs spaces and new lines at the end of files can vary from project to project.

For instance, the **jQuery Project** uses a full tab character while **Node** uses 2 spaces. The editor config project is a package that is available for Sublime Text (as well as all other major editors) which can help to bring some sanity to project-specific formatting nuances. Project authors include a `.editorconfig` file in the root of their project. When the Editor Config package is installed, it will detect this file, read the config settings and update your Sublime Text settings to reflect the project's preferences.

No more headaches keeping all of your team members or contributors on the same page with formating!

Paste And Indent

One of the best tricks I've ever learned in Sublime Text is **paste and indent**. If you have ever pulled a piece of code off the Internet and pasted it into your editor, you'll no doubt feel the pain of having it paste in all weird leaving you to have to go in and fix it.

Easy fix to this, instead of pressing the normal `⌘` + `v` to paste, is to simply switch to using `⌘` + `Shift` + `v` to paste. This will automatically indent your code block and switch it to your current use of tabs or spaces.

Like that one? I loved it so I remapped my key combos to paste and indent by default on `⌘ + v`

Paste this little snippet into your key bindings file located at `Preferences → Key Bindings - User`

```
{ "keys": ["super+v"], "command": "paste_and_indent" },
{ "keys": ["super+shift+v"], "command": "paste" }
```

4.4 Fonts and Type Sizing

Picking a font is one of the most important parts of making your editor **feel just right**. Sublime gives us very fine grain control over how text is displayed.

To change your font, simply set the value of "font_face" like so:

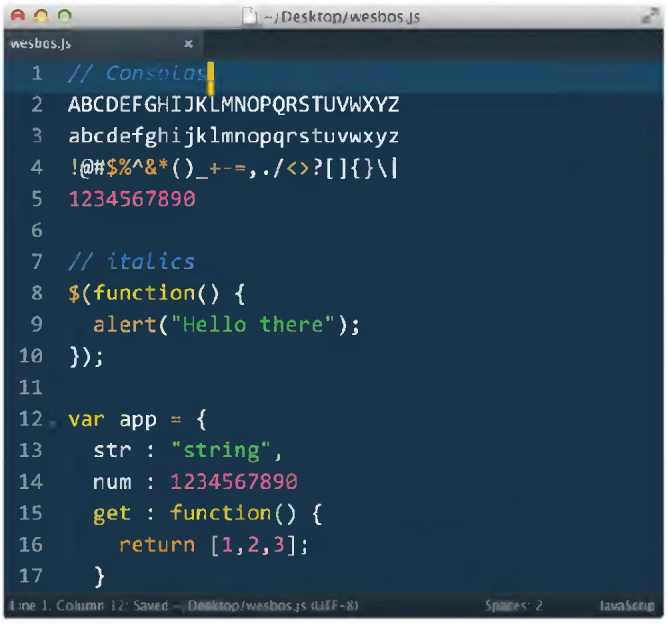
"font_face": "inconsolata"

Picking which font to use is a very personal thing. Some developers place importance on fonts that have an italic style, while others are picky about the visual difference between `0` and `o` or how the equals and open bracket make a fat arrow `=>`.

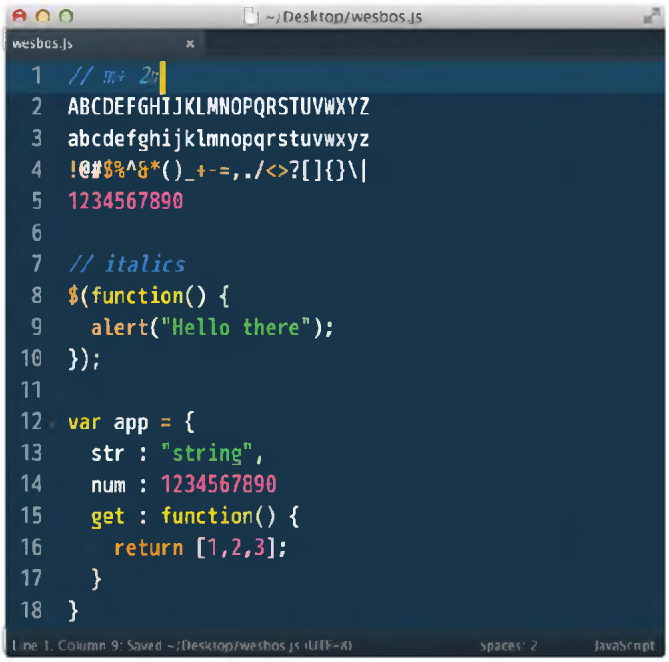
I've personally hopped around and used a few in the past few years and I'm currently between inconsolas and M+2m.

Here are a few of best and most popular programming fonts.

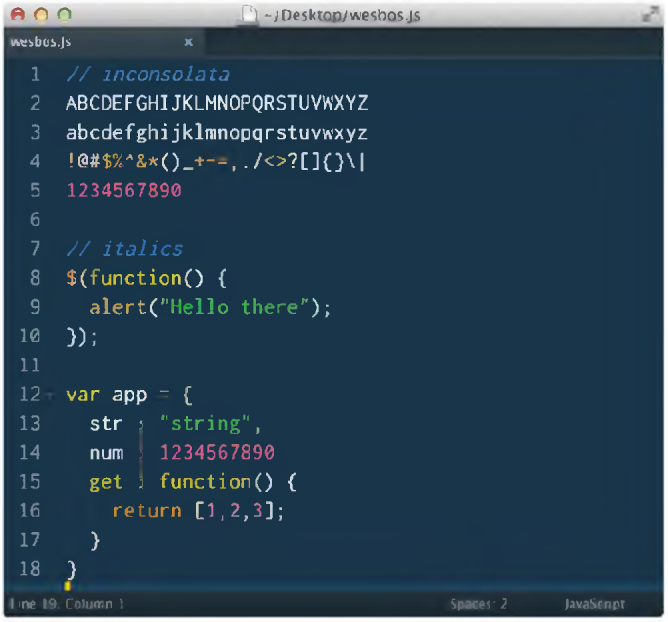
Consolas



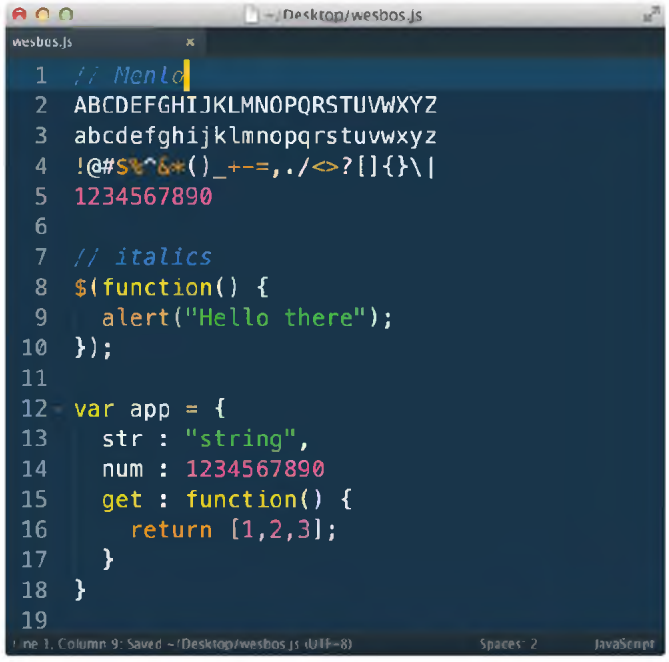
M+2m



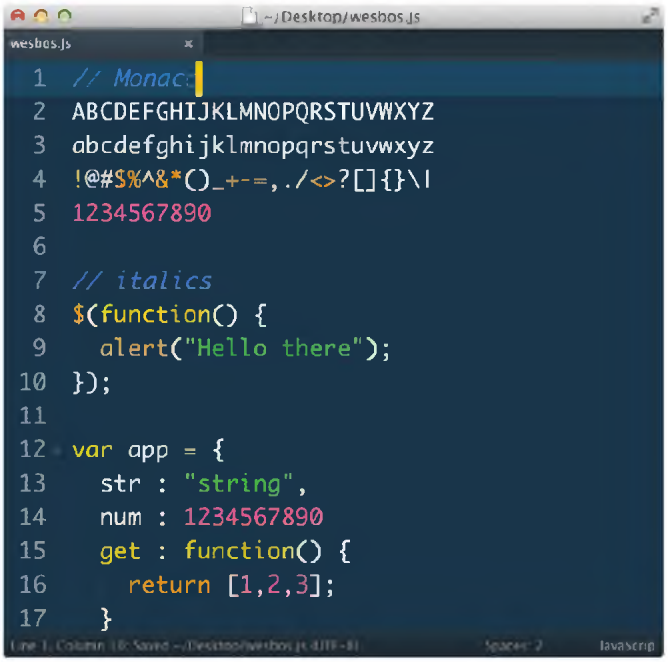
inconsolata



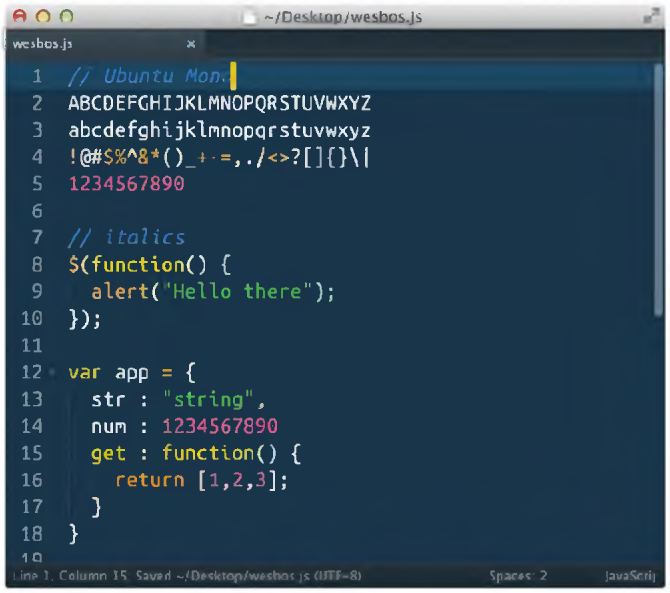
Menlo (sublime default)



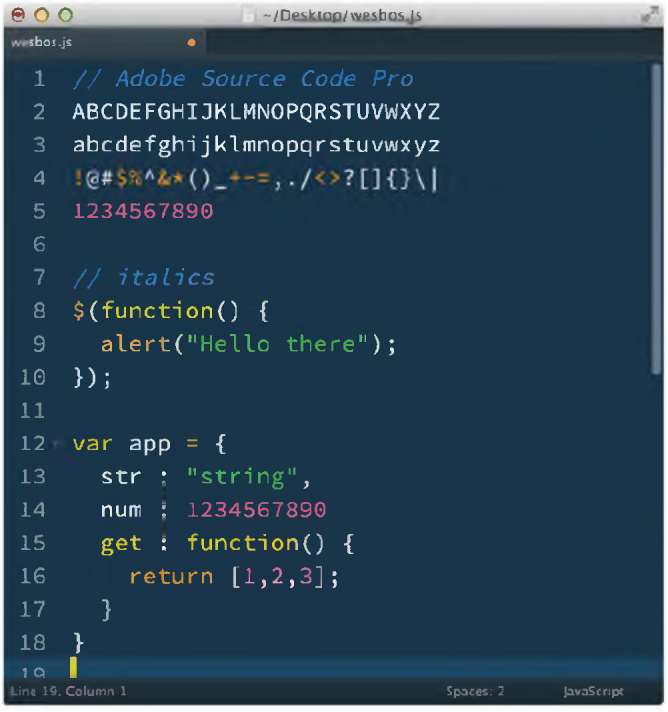
Monaco



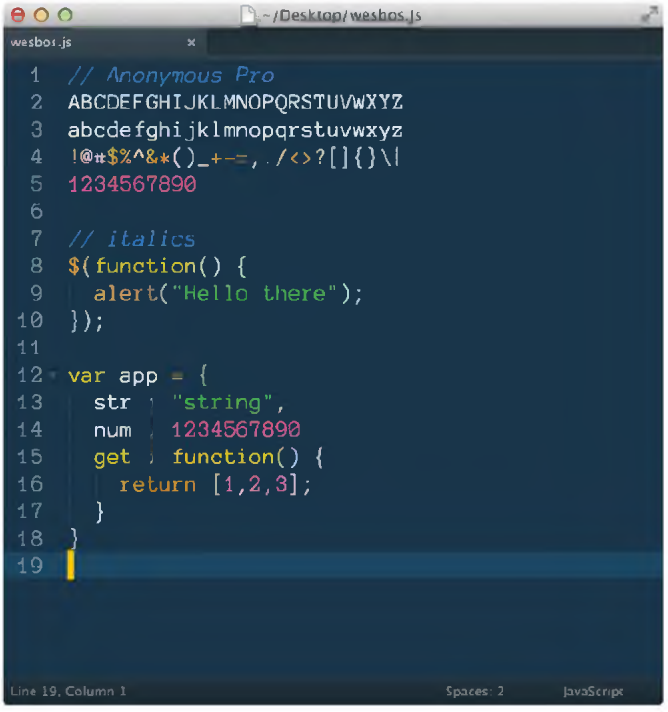
Ubuntu Mono



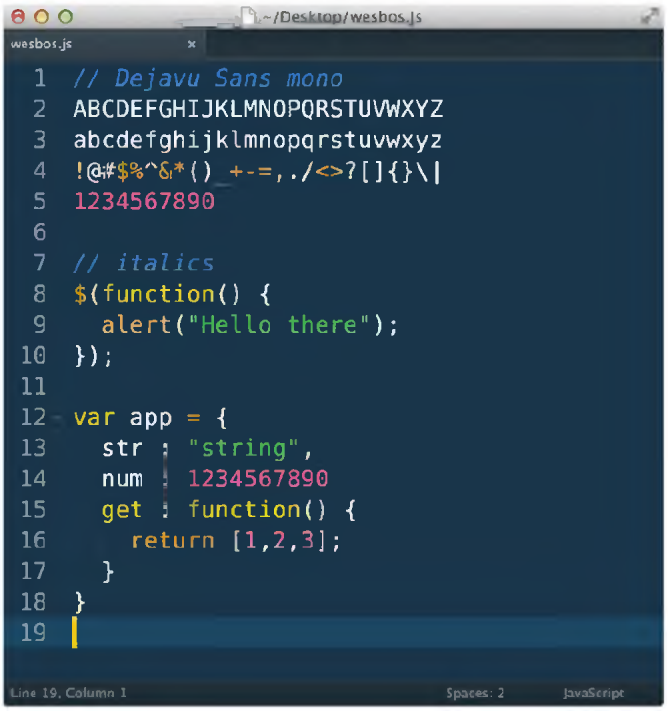
Adobe Source Code Pro



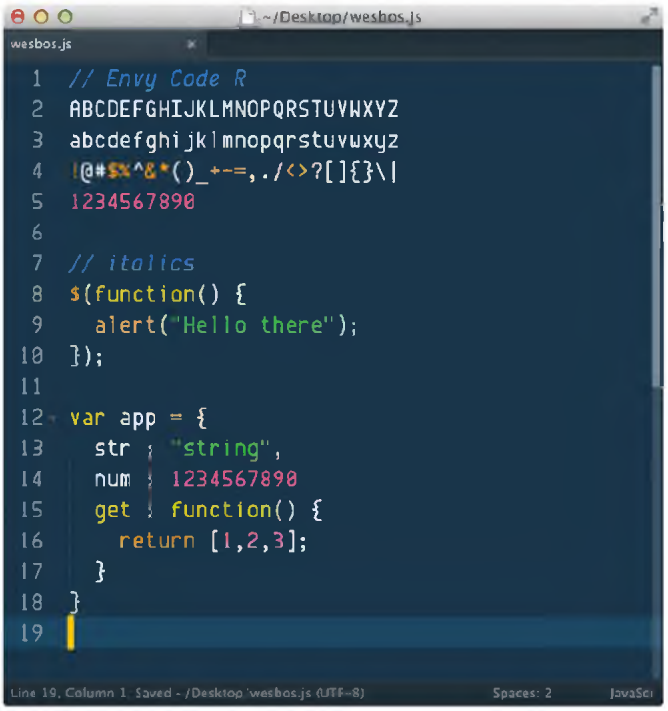
ANONYMOUS PRO



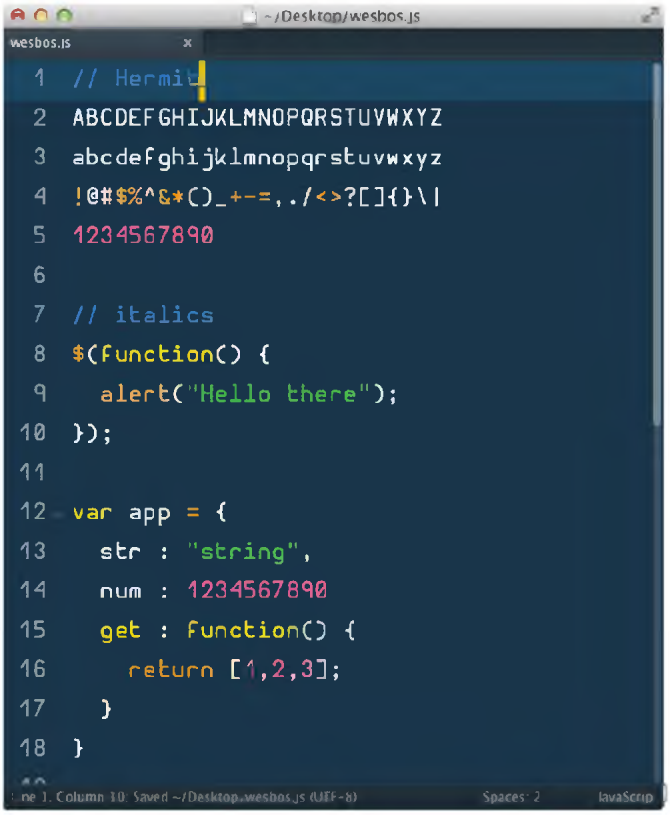
Dejavu Sans Mono



Envy Code R



Hermit



Tweaking Fonts

The first and most obvious thing you will want to do is change the font size:

```
"font_size": 16.0
```

You can also automatically add and change this value with `CMD/CTRL` + `+/-`

Another improvement I've made over Sublime Text's defaults is in increasing the line height. This allows for greater readability.

```
"line_padding_bottom": 1,
"line_padding_top": 1,
```

Finally, there may be a few specific use cases where you will need to specify some of these font options, although they generally provide no benefit to the user.

```
"font_options": ["no_bold", "no_italic", "no_antialias", "gray_antialias","subpixel_antialias", "no_round",
"directwrite"],
```

4.5 Sidebar

If you are coming from another editor, chances are you will want to set up Sublime Text very similar to how you used to have it. For me, having that trusty left hand sidebar was mine. I like being able to see the folders and files for the projects I'm working on.

By default, Sublime doesn't come with the sidebar to show open folders and files. To turn this on, simply hit `⌘ + K`, `⌘ + B` (Windows and Linux users use `Ctrl`) or access it from the menu under `view` → `Side Bar` → `Show Side Bar`

4.6 Minimap

The minimap is something that I've never seen in a code editor before. It allows you to view your code from 10,000 feet and works like a scrollbar on steroids.

Some love it and some hate it. Ultimately, it's just personal preference. Many developers have said it's great as they can easily pick out comments or code blocks without scrolling forever. This is especially helpful to Windows developers who don't have inertial scrolling.

The minimap draws a block around the currently visible content. Depending on your theme, you may want to add a border around that block with the following setting.

```
"draw_minimap_border": true
```

As I write this book, I'm able to see my progression. I have the border turned on to easily display where I'm currently at.

Code Completions and Intelligence

A big part of a text editor's job is staying a few steps ahead of you. That means when writing code, it should start to suggest possible endings to help you speed up your development.

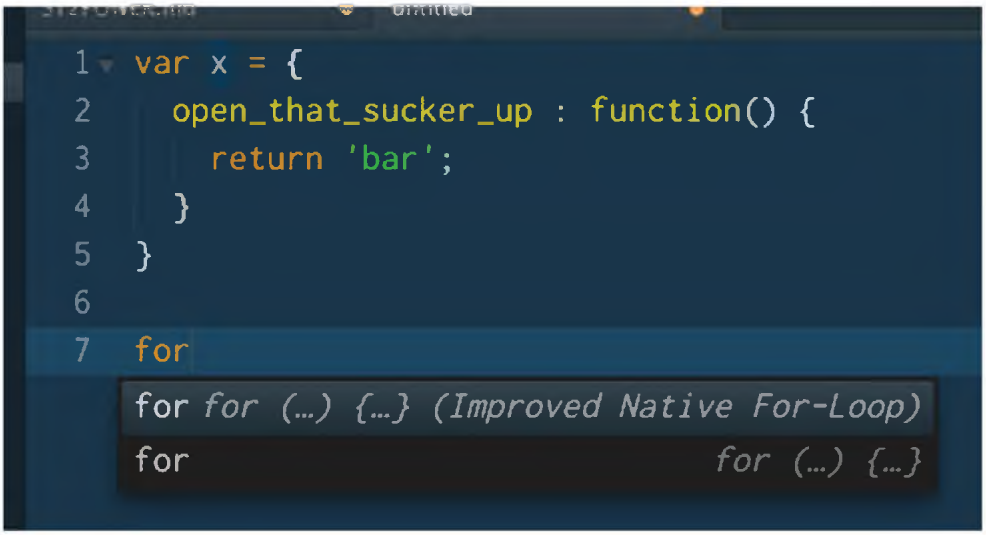
5.1 Code Hinting / Auto Complete

Out of the box, Sublime Text does an okay job at making suggestions for what you want to use. Auto complete is enabled by default, so there is no need to turn anything on. As you type, Sublime will offer hints for possible completions. Take this snippet of JavaScript for example:

```
var x = {
    open_that_sucker_up : function() {
      return 'bar';
    }
}
```

Typing `open_that_sucker_up` would take a long time, so as I type `ope...`, Sublime will suggest the completion of it. Hitting `tab` will auto complete the function name.

Sublime will also suggest possible snippets that are associated with the current language. In my example below, as I type `for`, Sublime will look through my JavaScript snippets and suggest two of them:



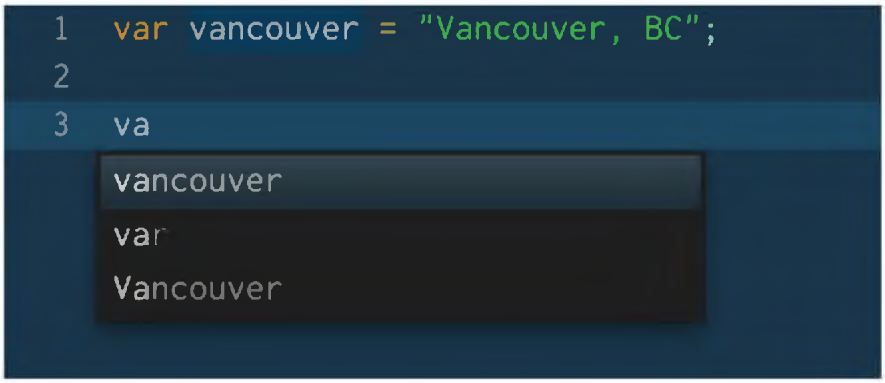
If you find yourself in a situation where auto complete isn't opening, you can trigger it manually with `Ctrl` + `SPACE`.

Where Auto Complete Fails

Auto complete isn't very smart so it will often suggest things that don't make any sense. Take the following single line of JavaScript as an example:

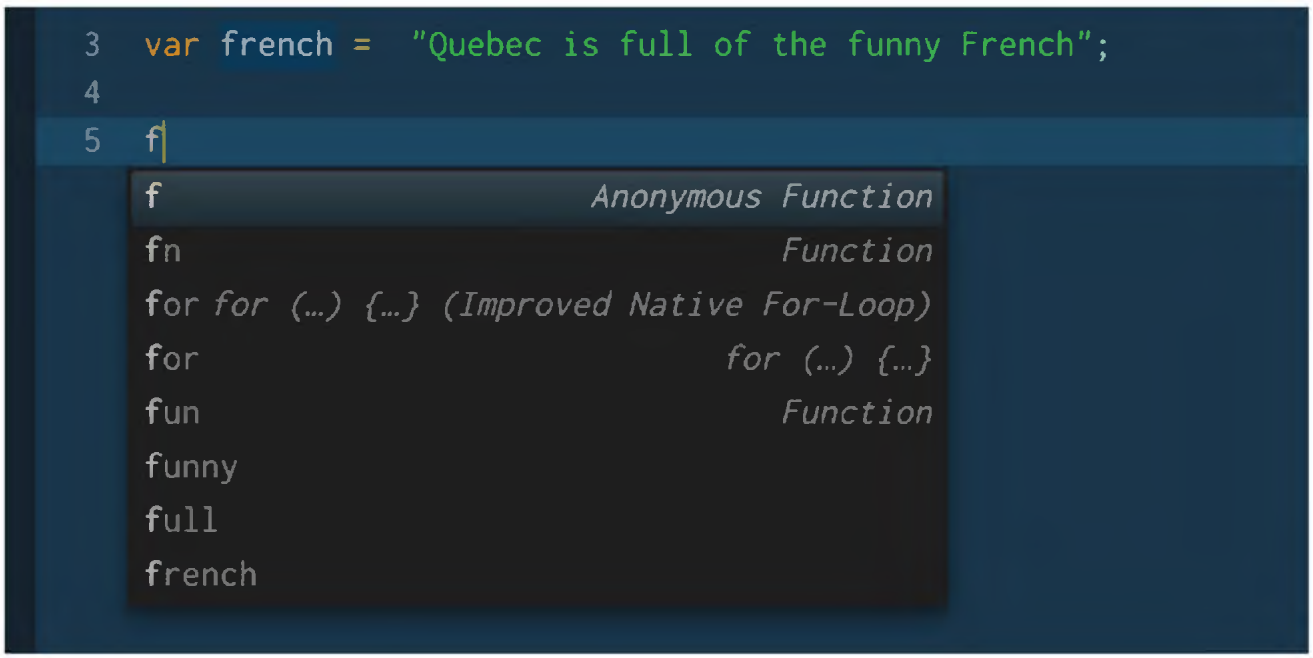
```
var vancouver = "Vancouver, BC";
```

A simple variable string assignment we have all seen in any language before. Now let's say I wanted to do something with `vancouver` so I start typing `va...`. Up pops auto complete with three suggestions: `var`, `vancouver`, and `Vancouver`.



First of all, I don't think I'd need to access `var`. Second, `Vancouver` (capitalized) is only part of the string, so I won't ever need that. The one I'm after is `vancouver`, which is in the middle of everything.

The problem compounds when snippets get in the way:



So, while the auto complete is good, there are better options available to speed up your development time.

Settings

Want to tweak the way auto complete works? There are a few options that you can set in any of your `.sublime-settings` files.

`"auto_complete": false` will turn off auto complete entirely. If you wish to do this specifically for a language, keep it out of your general user settings and place it in your language settings file. More on this available in the settings file.

`"auto_complete_size_limit"` allows to to set the threshold size of a file where auto complete works. Larger files with auto complete can slow down your editor so you may want to change this depending on how fast/slow your computer is.

`"auto_complete_delay"` allows you to set a delay in milliseconds before it opens. Want it right away as you type? Set it to 0. Want a 1 second wait? Set it to 1000.

`"auto_complete_commit_on_tab"` allows you to turn off the "tab to select" functionality. Some developers want their tab key to insert a tab character, so they turn this off.

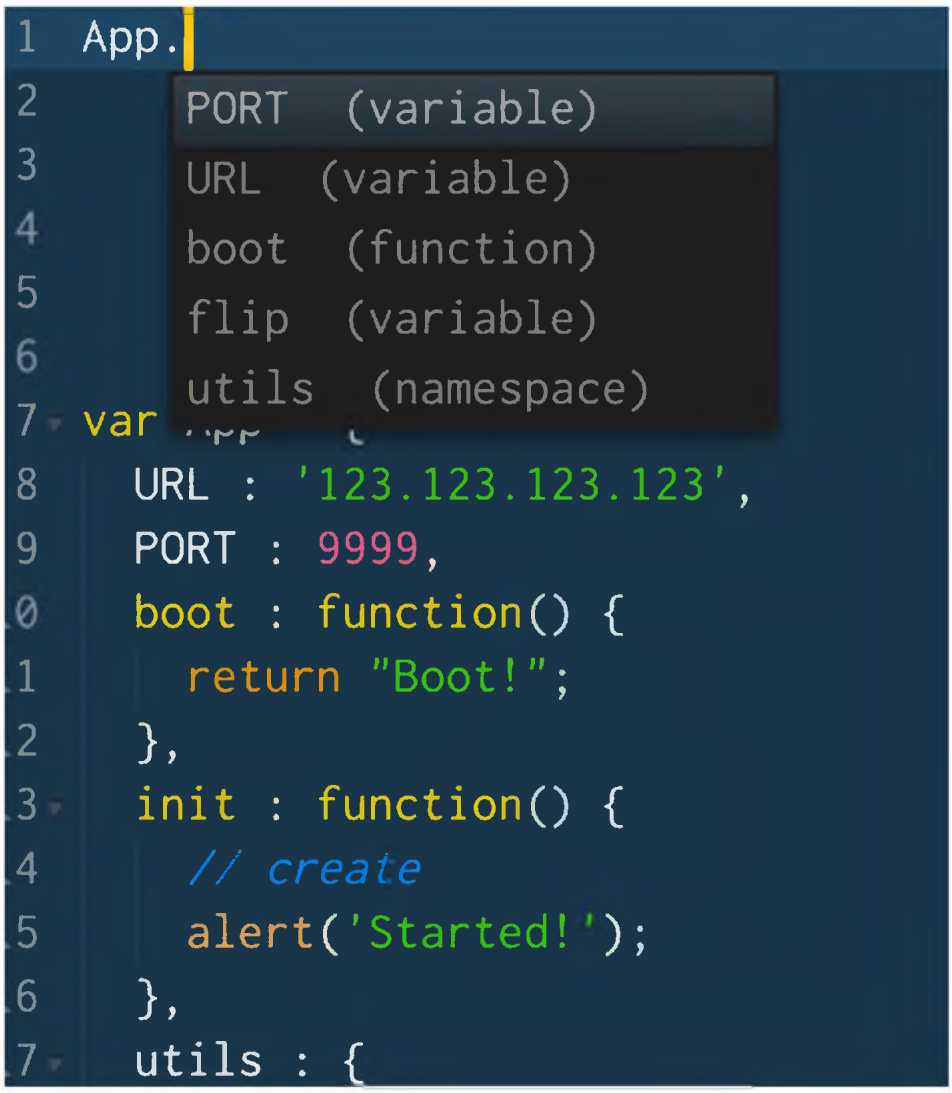
If you are getting really custom with auto complete, you can also tweak the selectors and triggers with `auto_complete_selector` and `auto_complete_triggers`. For more information on these, view the Default settings file available at `Preferences` → `Settings - Default`.

5.2 SublimeCodeIntel

Sublime Code Intel is a package that provides smart completions. The package provide support for all major languages including JavaScript, Mason, XBL, XUL, RHTML, SCSS, Python, HTML, Ruby, Python3, XML, XSLT, Django, HTML5, Perl, CSS, Twig, Less, Smarty, Node.js, Tcl, TemplateToolkit, PHP.

If you are using Sublime Text 2, go ahead and install it from Package Control. If you are writing JavaScript, make sure to you have Node.js installed on your computer. For those using Sublime Text 3, jump down to the end of the chapter and read the steps to get it working.

Once installed, Sublime Code Intel will scan your projects and provide smart auto completions to your code as it knows your structure of your application.



5.3 Installing on ST3

For those using Sublime Text 3, the package has been ported over, but support and ongoing development seems to be lacking. Thankfully, the package *is* stable and working on Sublime Text, it just takes a bit of work to get it going.

Thanks to John Blackbourne for detailing the process on his blog at https://johnblackbourn.com/sublimecodeintel-st3. I've based the below instructions off of his findings:

1. Use the development branch

Don't use the Package Control version of SublimeCodeIntel. Instead, use a Git checkout of the development branch.

To do this, you will need knowledge of Git and a terminal window. In the Sublime Text menu, go to *Preferences → Browse Packages* - this is the folder where you will need to move to with your terminal window.

Once in that folder, type

```
git clone -b development git@github.com:SublimeCodeIntel/SublimeCodeIntel.git
```

This will pull down the latest version of the code and switch to the development branch.

2. Clear your CodeIntel cache

If you've had older versions of SublimeCodeIntel installed, your CodeIntel cache probably needs clearing.

To do this, quit Sublime Text then go to your home directory and delete the entire .codeintel directory (eg. on OS X this is at ~/.codeintel).

5.4 3. Fix the language-specific config

Inexplicably, SublimeCodeIntel's default config disables project scanning for PHP and JavaScript files, which is what most users have been using it for.

To fix it, open the Sublime Text menu and go to `Preferences → Package Settings → SublimeCodeIntel → Settings - Default`. At the bottom of this file are language-specific settings for Python, JavaScript, and PHP.

Do not edit this config directly. Instead copy it's contents completely and paste it into your user settings (`Preferences → Package Settings → SublimeCodeIntel → Settings - User`). Then change the PHP and JavaScript settings from the bottom to `codeintel_scan_files_in_project` settings to true.

5.5 4. Be patient

When you first begin typing a function name or other entity that triggers CodeIntel, the status bar will show you that the initial scan is taking place. I've noticed that this status message disappears before scanning is actually complete, or it'll state that scanning is complete when it's not. Be patient, and it will get there eventually.

Terminal and Command Line Integration

6.1 OSX

If you work heavily in the command line, it's worth taking some time to both implement the `subl` command as well as learn all of the available arguments which will tightly knit your terminal environment and Sublime Text.

First, you need to link up the `subl` command. `subl` allows you to send files from terminal to Sublime Text in a variety of ways. Open up your terminal window and paste this little snippet in:

Sublime Text 2

```
ln -s "/Applications/Sublime Text 2.app/Contents/SharedSupport/bin/subl" ~/bin/subl
```

Sublime Text 3

```
ln -s "/Applications/Sublime Text.app/Contents/SharedSupport/bin/subl" ~/bin/subl
```

> **Having Trouble?** Your mac may complain that there isn't a `~/bin` folder. If this is the case, try to run `sudo ln -s "/Applications/Sublime Text.app/Contents/SharedSupport/bin/subl" /bin/subl`. This will both ask you for a password and install it on your systems `/bin` folder rather than the current user.

If you wish to use another command in place of `subl`, just replace the last instance of the word `subl`. I've seen some developers use `slime` instead.

6.2 Windows

If you are on a windows machine, fear not! There is an easy way to have similar functionality.

Simply open up the command line (`start` → `run` → `⌘`) and paste the following line:

Sublime Text 2

```
doskey subl="C:\Program Files\Sublime Text 2\sublime_text.exe" $*

![](http://wes.io/M7r3/content.png)
```

Sublime Text 3

As of build 3065, Sublime Text now includes support to `subl` on windows. `subl.exe` comes in the installation folder of Sublime Text - you must move it over to your system path so it is available whenever you open the windows console.

While your install may vary, most windows computers will go something like this:

1. Copy `subl.exe` from `C:\Program Files\Sublime Text 3`
2. Paste into `C:\Windows\System32`

6.3 Linux

If you are running Linux, chances are you already have this setup, but here is how to do it should you need it. Pop these into your bash profile.

Sublime Text 2

```
alias subl='/usr/bin/sublime-text-2'
```

Sublime Text 3

```
alias subl='/usr/bin/sublime-text'
```

6.4 Using subl from the command line

The two most common use cases are to:

Open any directory

```
subl ~/path/to/folder
```

Open the current directory

```
subl .
```

or a single file

```
subl index.js
```

We can also specify the line of a file

```
subl weather.js:50
```

And even the exact column if you are tracking down a pesky bug that is breaking your script

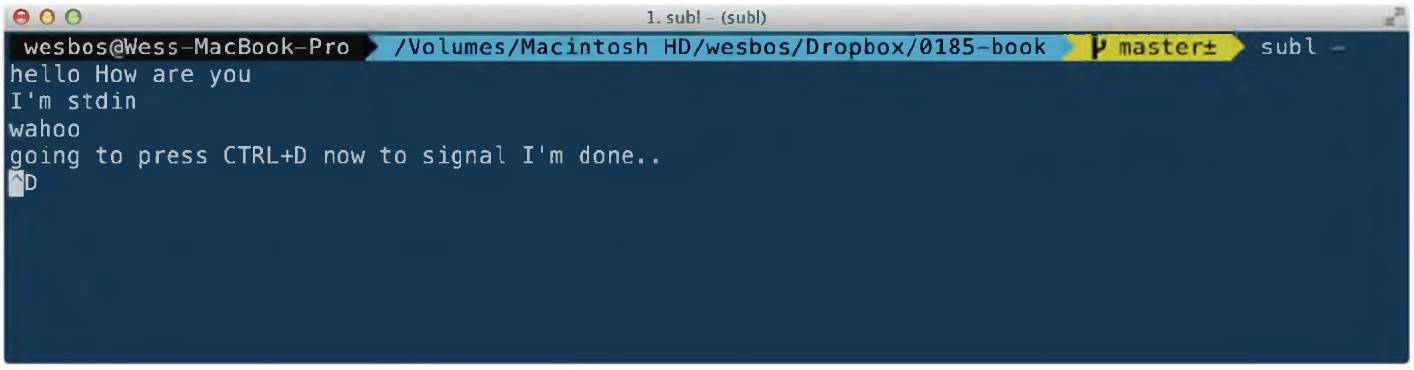
```
subl routes.js:34:20
```

Arguments

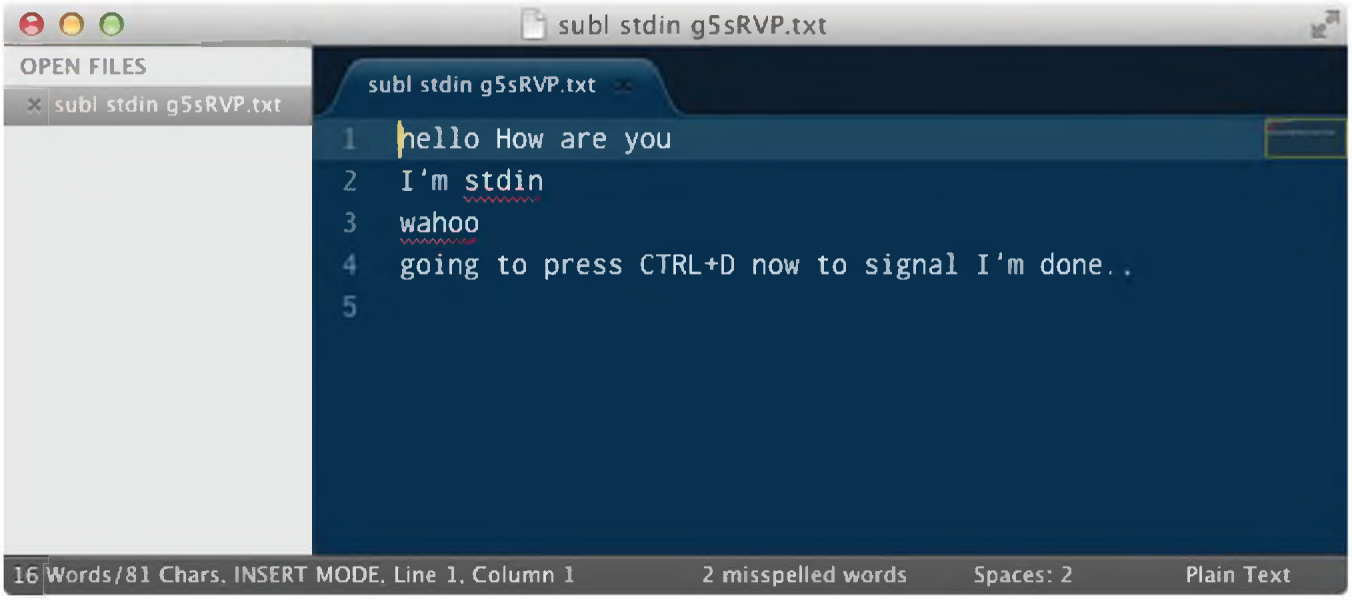
Sublime also allows us to pass a number of arguments to the `subl` command. Let's take a look at all of them now:

- (std in)

This is super handy if you run scripts from the terminal and want to pipe the output right into Sublime Text. Here is a quick demo:



When I press `Ctrl` + `D` to signal I'm done, it pipes the info into a temporary .txt document



--project <project>:

Load the given project

```
subl --project my-project.sublime-project
```

--command <command>:

Run the given command

```
subl --command css_tidy
```

-n or --new-window:

Sometimes you want to open a fresh window rather than add the file to the existing project.

```
subl -n totallyDifferentFile.js
```

-a or --add:

Add the current file or folder to the active / last used sublime text window. I often use this along with touch:

```
touch newFile.js
subl -a newFile.js
```

-w or --wait:

Waits for the files to be closed before returning. Helpful if you are writing to a file and don't want to open a semi-finished file.

-b or --background:

Don't bring Sublime Text into focus. Handy when you are opening several files one after another.

-s or --stay:

Keep the application activated after closing the file.

-h or --help:

Show the help - modified version of this.

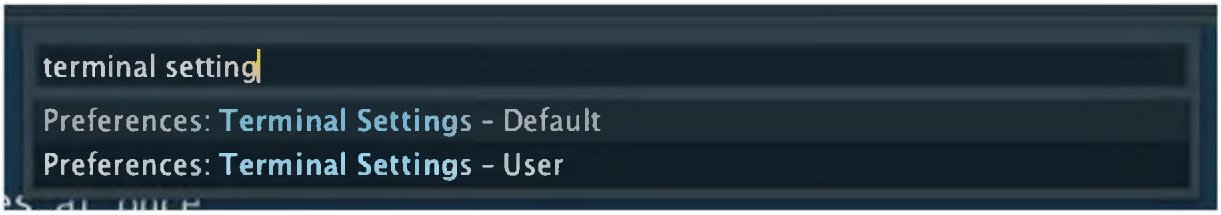
-v or --version:

Show version and exit

6.5 Terminal Package

On the other side of things, you can also go from Sublime to the terminal in one easy step with Will Bond's Sublime Terminal package. It works on windows, linux and mac.

If you are an OSX user, it will use the built-in terminal.app that comes with OSX. However, you have the option of using iTerm2, which is a more feature-rich version of terminal.app. To turn this on, set the following under `Terminal Settings - User` (available in the command palette).




```
{
  "terminal": "iTerm.sh"
}
```

Once installed, you can either open a file's parent folder or a project's folder. The quickest way is to use the keyboard shortcuts:

OSX:

- `⌘` + `Shift` + `t` to open the current file's parent directory in terminal
- `⌘` + `Shift` + `Alt` + `t` to open the current project directory in terminal

Windows & Linux:

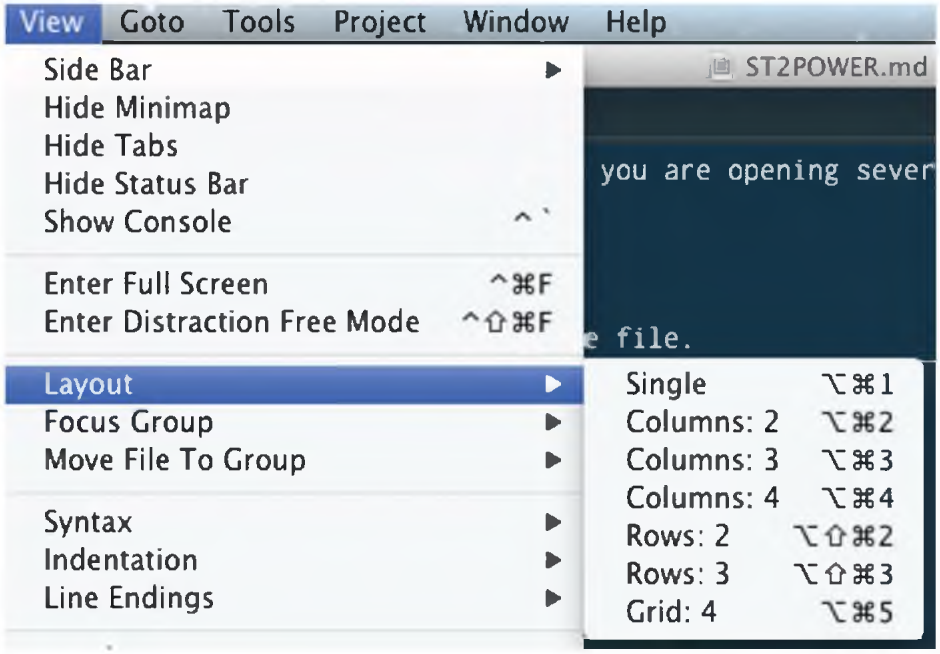
- `Ctrl` + `Shift` + `t` to open the current file's parent directory in terminal
- `Ctrl` + `Shift` + `Alt` + `t` to open the current project directory in terminal

Commands are also available via the context menu by right clicking a file or folder and selecting `Open Terminal Here...`

Maximizing Screen Real Estate with Multiple Panes and Origami

If you work on a nice big monitor or keep a strict ~80 char width limit in your code, you probably like to code with multiple panes of code open at once. Multiple panes are particularly helpful when working with styles and templating at the same time. Since CSS/LESS/SASS aren't very wide, having both panes open at once is an option, even when on smaller screens.

Panes in Sublime are pretty straight forward. You can have **up to four columns**, **up to 3 rows**, or a **2×2 grid** of panes open at once. You can find all of these options available under `view` → `Layout`



Menu-schmenu. We are efficient - lets learn the keyboard shortcuts for them.

OSX

`⌘` + `Option` + `[1-4]` will split into 1-4 columns accordingly

`⌘` + `Option` + `5` will give you a 4x4 grid

`⌘` + `Option` + `Shift` + `[2-3]` will split into 2-3 rows accordingly

Windows and Linux

`Alt` + `Shift` + `[1-4]` will split into 1-4 columns accordingly

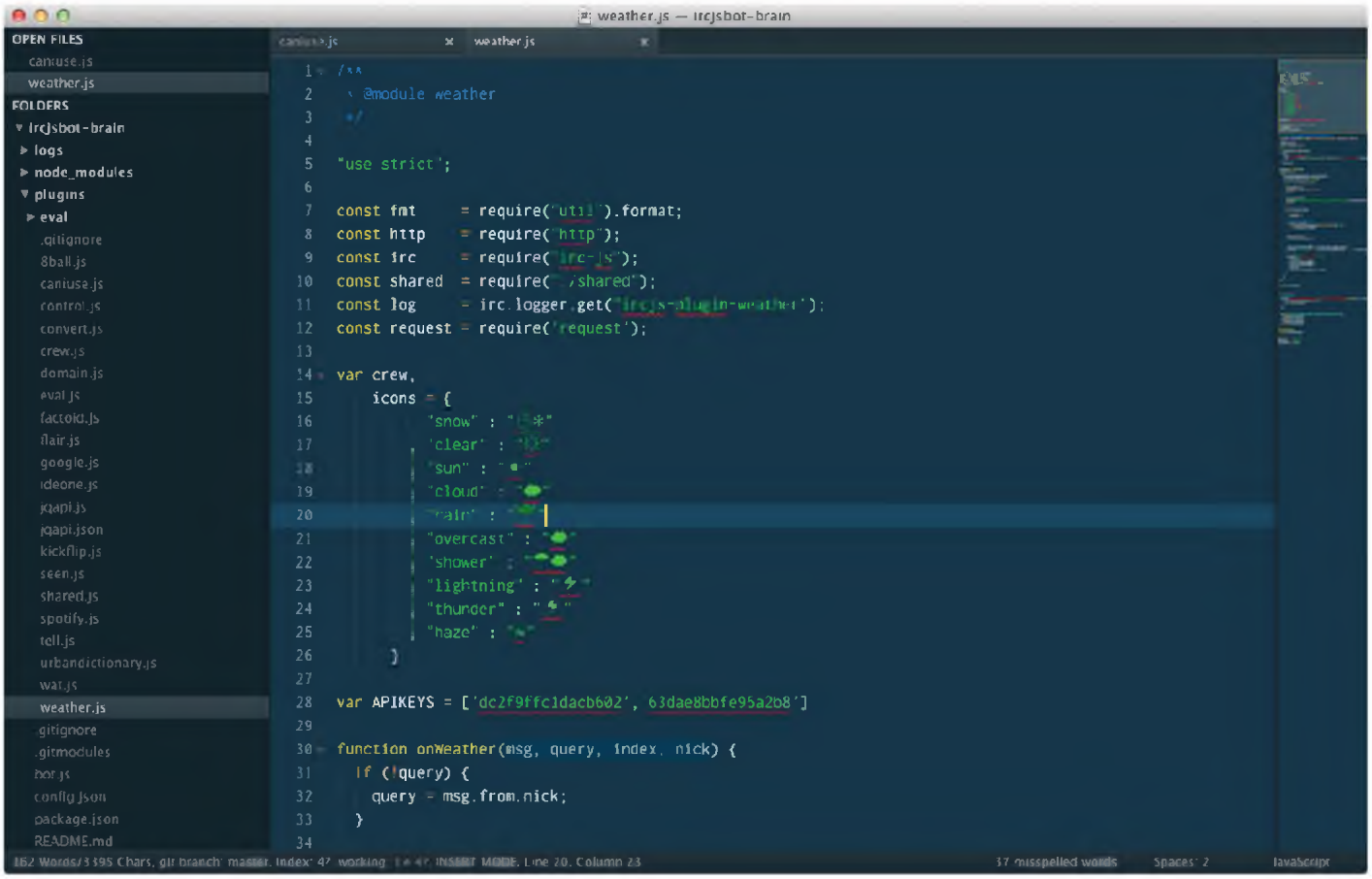
`Alt` + `Shift` + `5` will give you a 4x4 grid

`Alt` + `Shift` + `Shift` + `[2-3]` will split into 2-3 rows accordingly

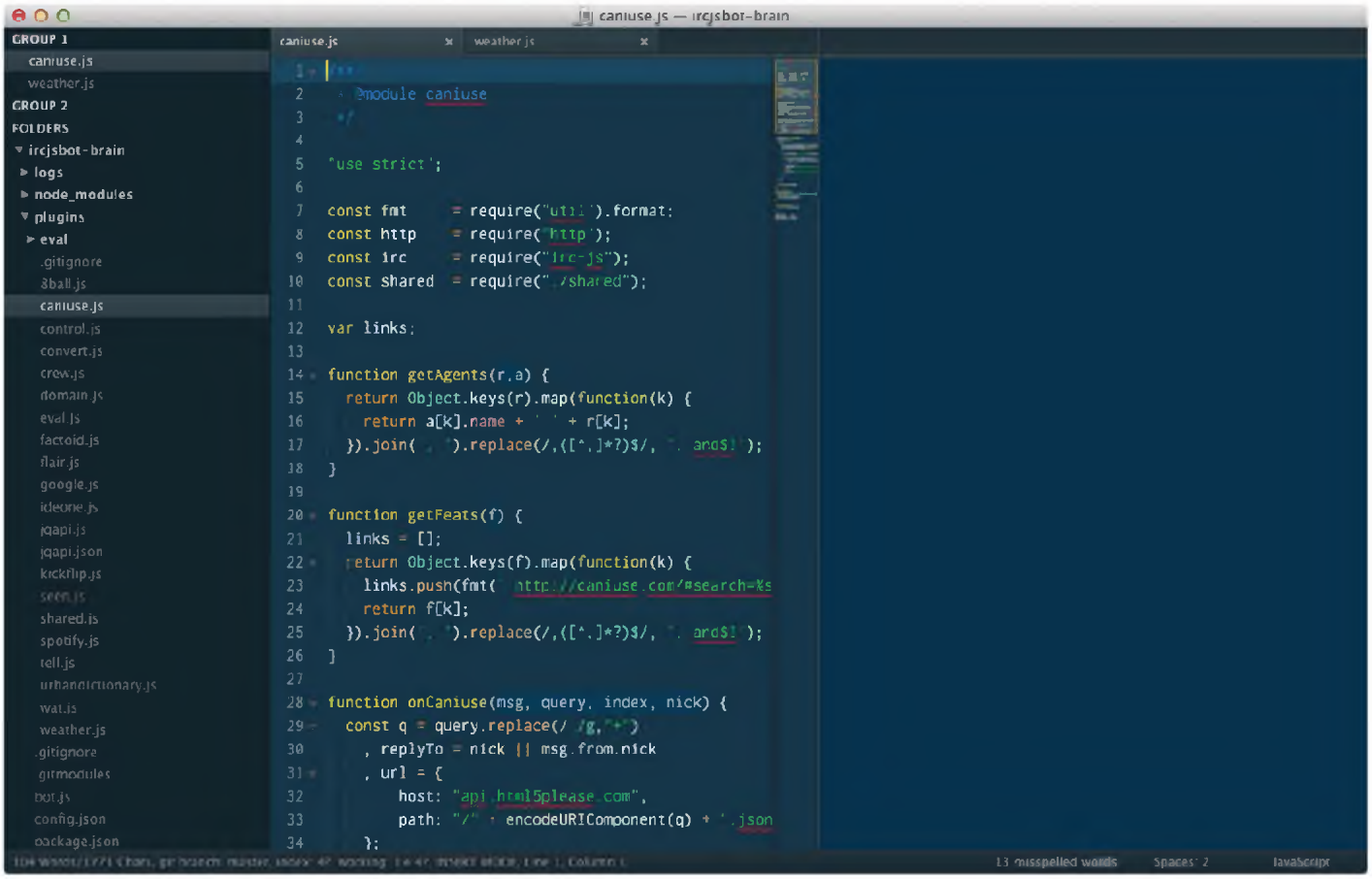
Panes Exercise

Once you have your editor split into panes (note, Sublime calls these **groups**), we can move focus from pane to pane. Let's give it a shot.

Start with a single pane and open up two different files. Focus the second file.

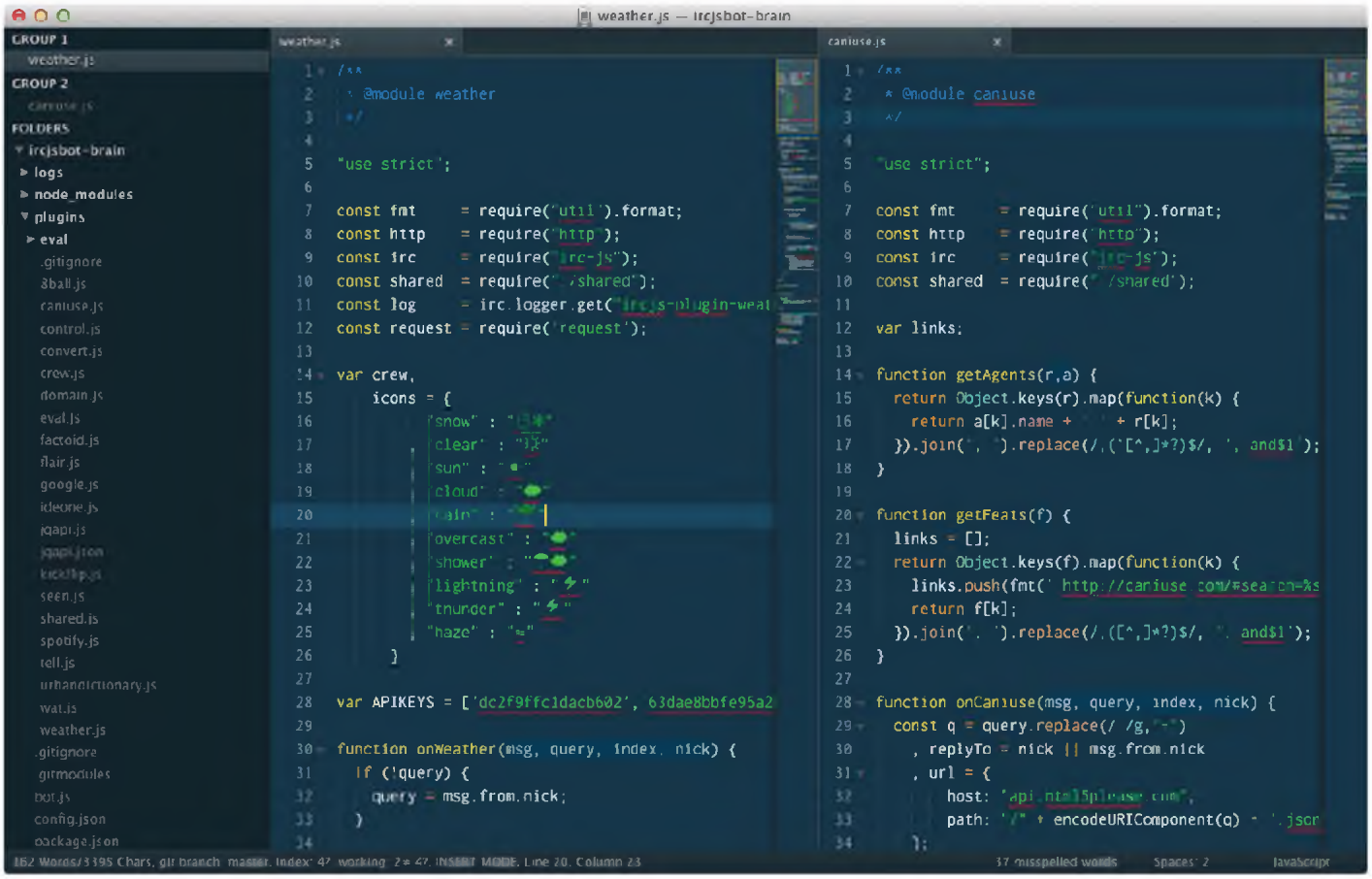


Now split the current window into two with the keyboard shortcut `⌘` + `Option` + `2` and focus the first tab with `⌘` + `1`



What we want to do now is switch the first tab over to the second pane without touching our mouse.

Pressing `Ctrl` + `Shift` + `2` will move the current file to the second pane.



Now we want to make a new file, but in the first pane. First we need to focus the first pane and then create a new file/

`Ctrl` + `1`

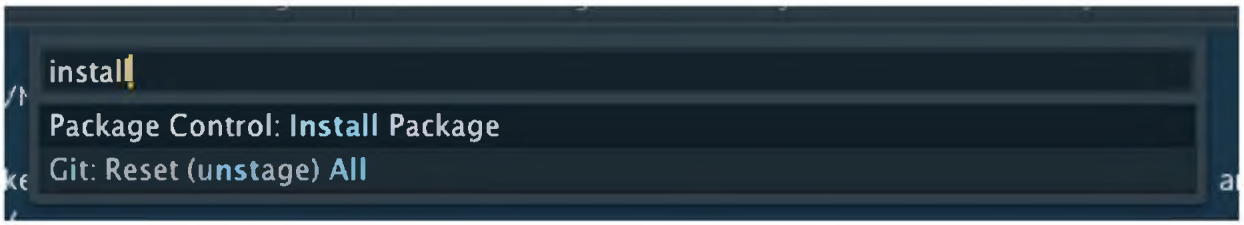
`⌘` + `N`

Try this 10 times over to get the hang of the shortcuts. Soon enough they will become second nature.

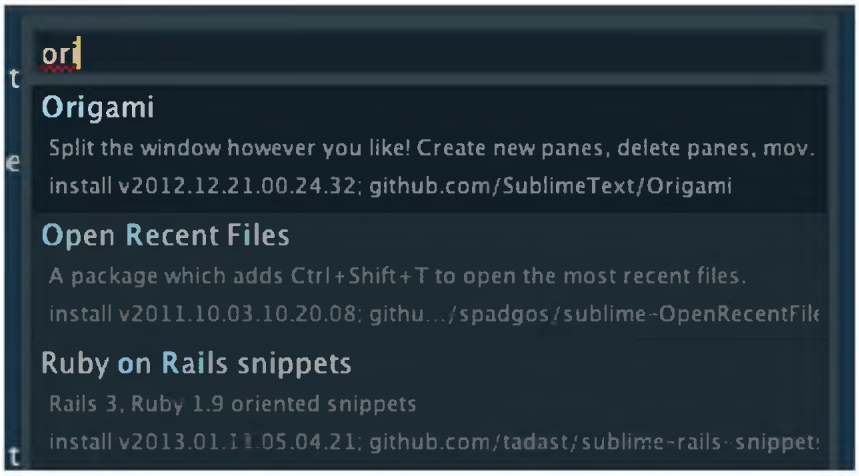
Origami

The above may be enough for you, but if you rock a large 27" or 4k display and have space to spare, you may wish to have a little more control over splitting up your screen. Enter *Origami*, a package that makes this possible by allowing users to split and resize panes into infinity. It also offers a nice set of commands for moving files from pane to pane.

To install bring up the command palette with `⌘/CTRL` + `Shift` + `P` and type install.



After about two seconds you'll be prompted to type the name of the package. Go ahead and search for **origami** and hit enter. It will take just a few seconds to install. No need to restart.



Origami is simple, the functionality can be broken down into **pane actions**: *create*, *destroy*, *focus* and **file actions**: *move* and *clone*.

> Windows and Linux Users: Replace `⌘` with `Ctrl` for the following keyboard commands

The Origami keyboard shortcuts are twofold, you must always press `⌘` + `K` followed by the keyboard shortcut. At any time you can open the Goto Anything palette (`⌘` + `P`) and type `Origami` to see all available commands and their respective shortcuts.

Create

To create a pane to the right: `⌘` + `K`, `⌘` + `→`. Use this with any direction - `←`, `↑`, `→` or `↓`

Note that it is not necessary to take your fingers off the `⌘` key when switching between `K` and `←`, `↑`, `→` or `↓`

Destroy

To destroy a pane, add `Shift` to the above keyboard shortcut. So, `⌘` + `K`, `⌘` + `Shift` `→` would destroy the pane to the right.

Focus

Focusing a pane will allow you to create and view tabs within that pane. For this, we just drop the `⌘` from the second keyboard shortcut. `⌘` + `K`, `→` will focus the pane to the right.

Move

To move a file from one pane to another, we use the `Shift` key along with the direction. `⌘` + `K`, `Shift` `→` would move the file to the right pane.

Of course, you can also just drag and drop your tabs from one pane to another.

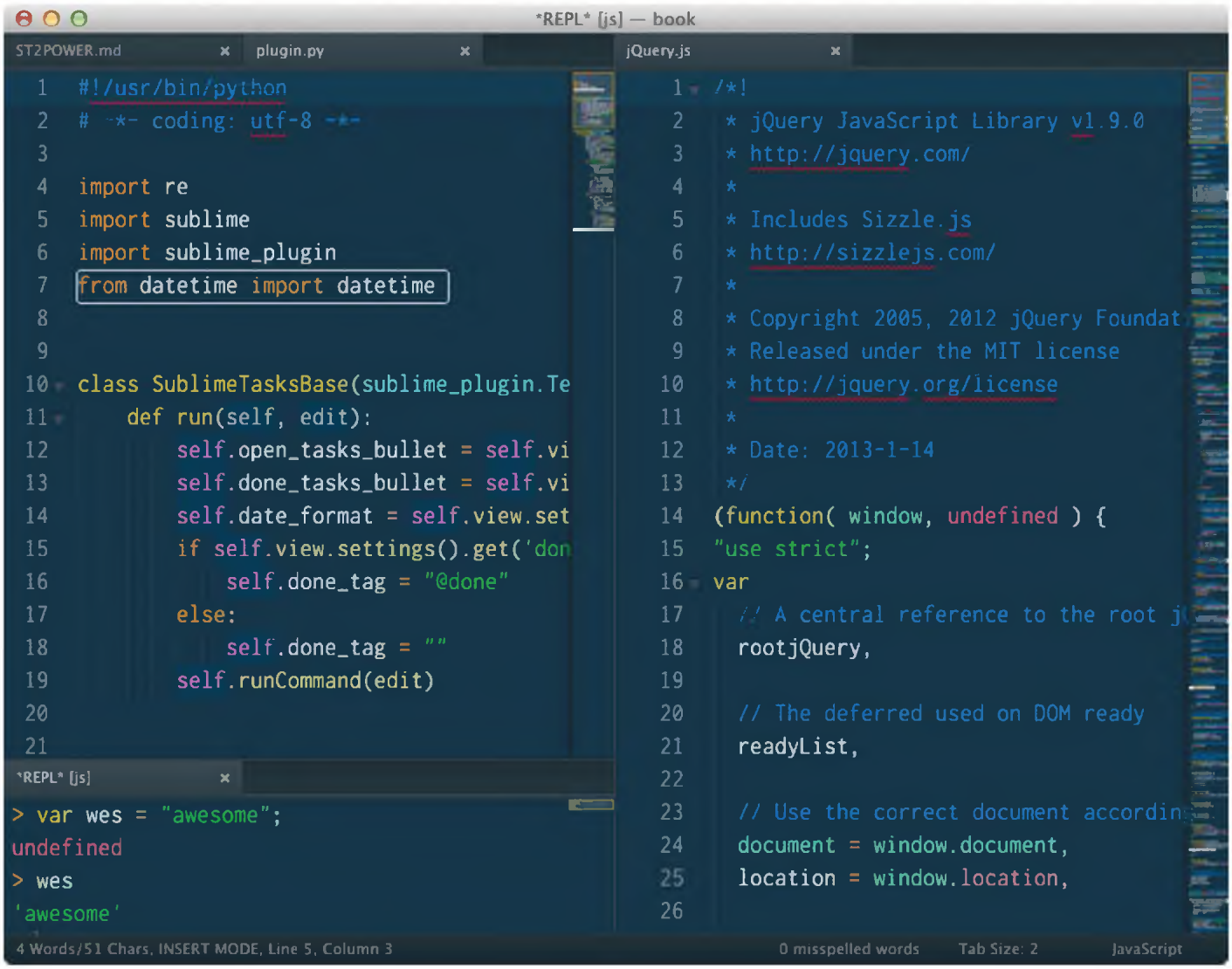
Focus

Finally, the last option that Origami makes available to us is something I haven't seen in a text editor before: Cloning. Cloning isn't a great name for it, it's more "opening the same file twice without making a second copy". I guess that was too long. Cloning a file allows you to open a mirror of the file and display it in another pane - you can edit either file and both will be updated. This is helpful when you want to reference the same file you are working on without having to scroll around and lose your spot.

To use clone, we use the `control` key. `⌘` + `K`, `control` `→` would clone the current file to the right pane.

Resize

To resize your panes, simply drag an edge to the size that you desire. I like to use Origami to have a small JavaScript REPL open at the bottom of my screen (More on Sublime REPL in the packages section).



7.1 Moving Between Tabs

OSX

Sublime is no different from most other programs with tabs (like Firefox or Chrome). To move from tab to tab, you can use `⌘` + `[1-9]` to move to that numbered tab, or the `⌘` + `Option` + `←` or `→` to move to the previous/next tabs.

Windows and Linux

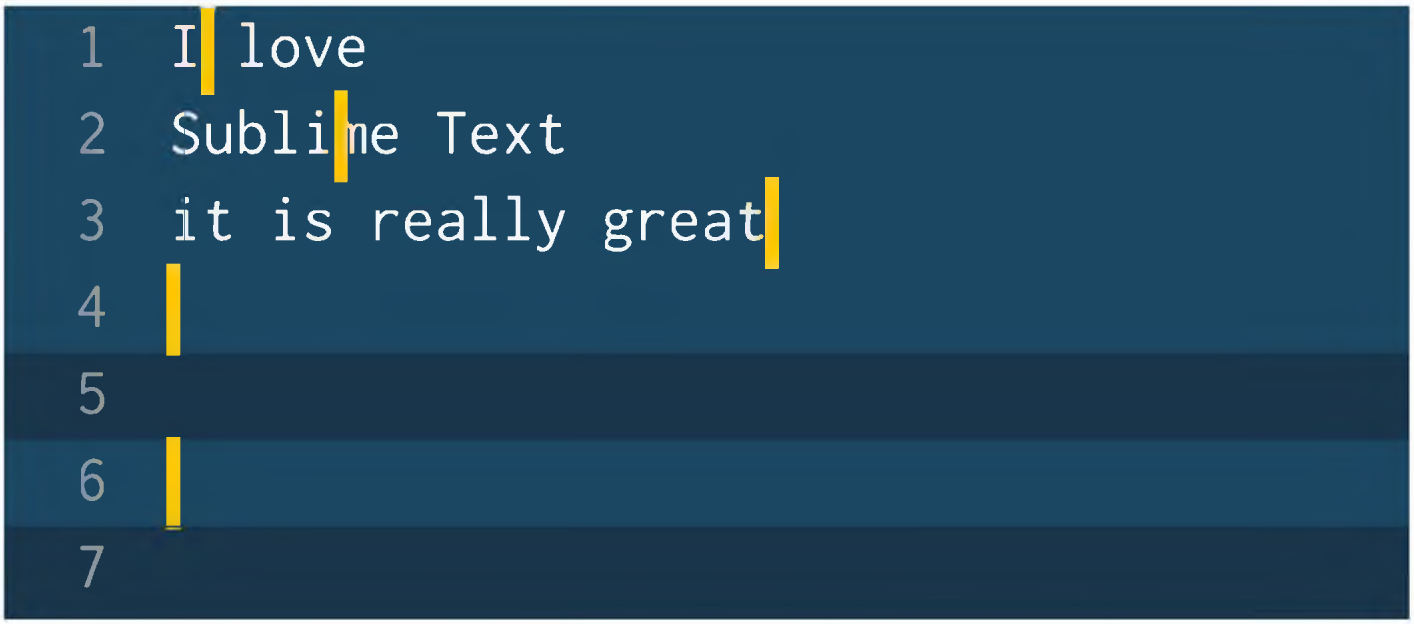
Sublime is no different from most other programs with tabs (like Firefox or Chrome). To move from tab to tab, you can use `Alt` + `[1-9]` to move to that numbered tab, or the `Ctrl` + `tab` to move forward a tab, and `Ctrl` + `Shift` + `tab` to move to the previous tab.

Working with Multiple Carets and Selection

> Note: The following section makes use of the ⌘ key. Windows and Linux users should replace ⌘ with `Ctrl` unless otherwise specified.

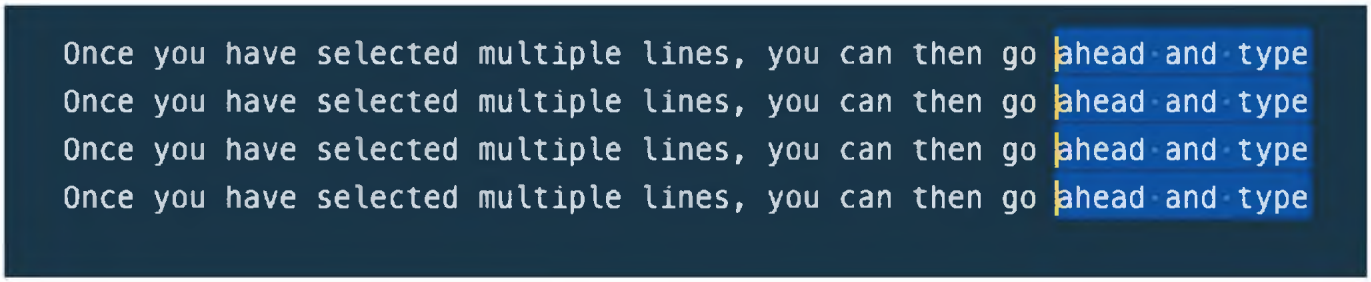
A caret is the little blinking line that indicates where you are currently typing. This is often mistakenly called a *cursor*, which is the little pointer or hand that tracks where your mouse is located. In Sublime Text, we can have as many carets as we want!

The simplest way to get started with multiple carets is to open a blank document, add a few lines and then select each line by holding down ⌘ and clicking wherever you wish to have an additional caret.



Alternatively you can also use the keyboard and use `Ctrl` + `Shift` + `Up/Down` (`Ctrl` + `Alt` + `Up/Down` to windows) to add additional carets to the previous / next lines.

Once you have selected multiple lines, you can then go ahead and type and see the output for each line



That is really cool, but where is that useful?

8.1 Replacing Words

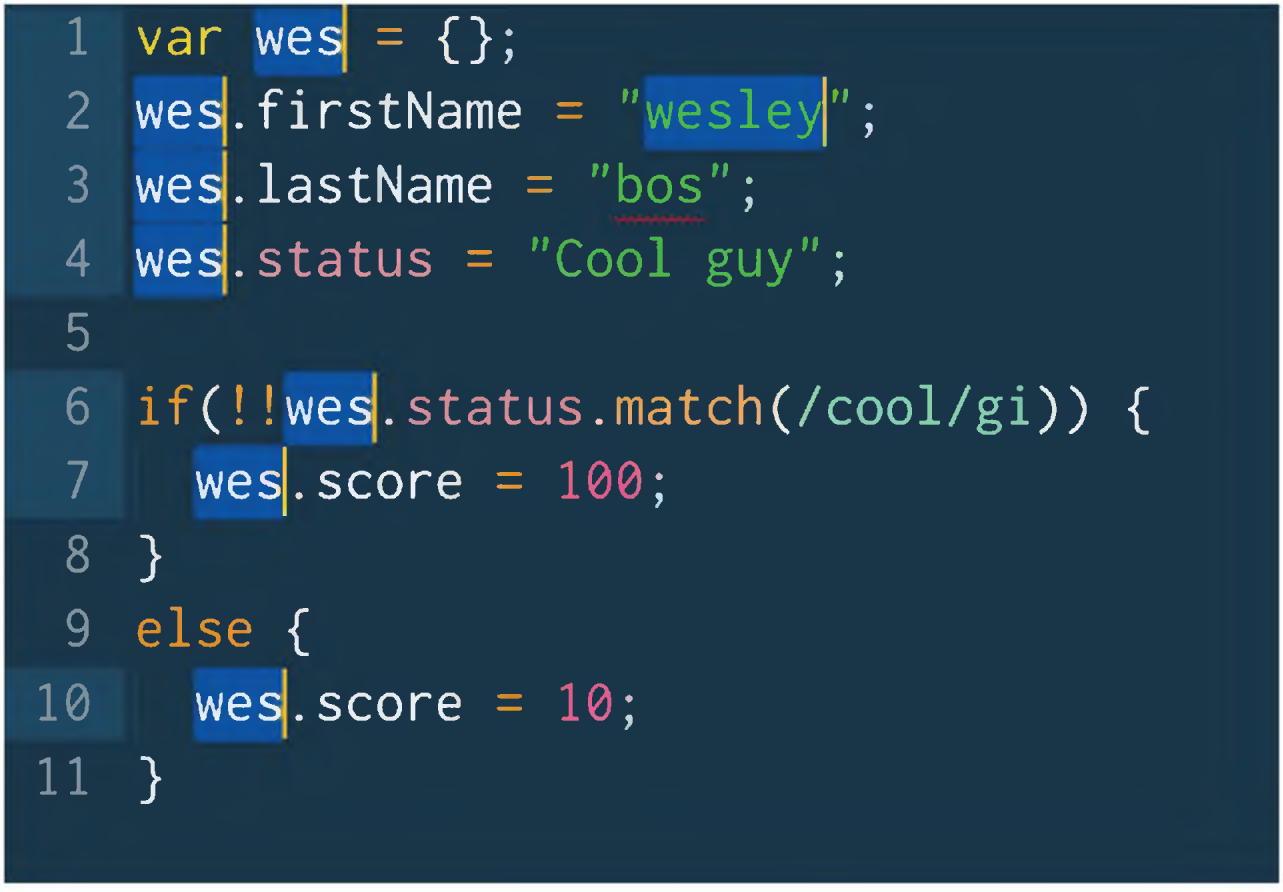
Let's take another example. Say we had a block of code and I needed to replace all instances of the variable `wes` with `alex` .

```
var wes = {};
wes.firstName = "wesley";
wes.lastName = "bos";
wes.status = "Cool guy";
```

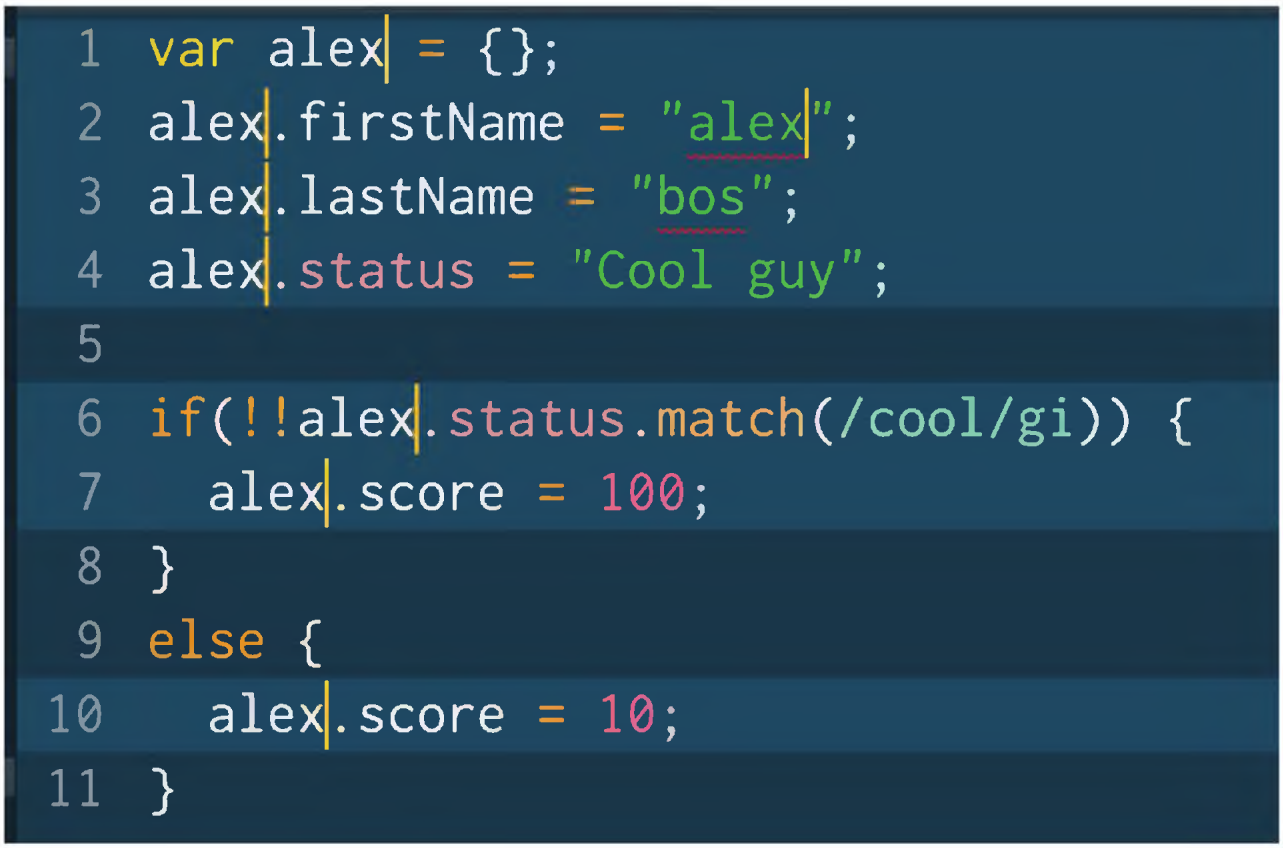
```
if(!!wes.status.match(/cool/gi)) {
  wes.score = 100;
}
else {
  wes.score = 10;
}
```

There are a few ways to swap out every instance of `wes`. The first one that might come to mind is a find/replace on just this selection - and you are right, we will cover that in the searching and finding section of the book. For this exercise, we are going to be using multiple selection to swap out every instance of `wes`.

To select all `wes`, we can hold down `⌘` and double click everywhere there is an instance of `wes`. We could also double click the first `wes` and then hit `⌘` + `D` to incrementally select each instance – practice both because you will find situations where you will use both.



Once you have them all selected, simply just type `alex` and you are done!



Quick Find Next / Quick Skip Next

Above we used `⌘` + `D` to select incrementally, which is great when everything you are looking for is matched. However, sometimes you will need to skip over a match and keep going.

For example, finding the word "wes" in the following list matches in "awesome" and "western". To skip a match, first select it with `⌘` + `D` to move to it, then use `⌘` + `K` + `D`

Use ⌘ + K + D to skip over matches

8.2 Modifying Multiple Lines at Once

Another case is when you have been given some HTML that looks like the snippet below. Ideally we would never have to work with something like this, but we have all come across something like this at some point:

```
1 <img src="/images/dog.jpg">
2 <img src="/assets/header.png">
3 <img src="/images/cat.jpg">
4 <img src="/i/harry.jpg">
5 <img src="/final/proof.jpg">
6 <img src="/2012/006.jpg">
7 <img src="/content/storefront.jpg">
```

We need to prefix each image with `thumb-`. Problem is, each image path is a different length and editing them manually would take forever!

Never fear! The following steps will quickly get every line selected:

1. Click on the first line
2. While holding down ⌘ and `Option`
3. Drag your mouse through the rest of the lines

You should now have a cursor somewhere on each line

Again, use `⌘` + `←` (`Alt` + `←` for win/linux) to bring all cursors to the front of the line, and then we will hold down `Option` while using our arrow keys to jump by word. When in front of the file name, simply type `thumb-`

8.3 Another Multi-caret Example

To show how awesome multiple carets are, let's take a look at some text that you may get in an email that we need to convert to HTML.

```
1. Go to the store.
2. Buy some apples.
```

```
3. Peel those apples.
4. Bake me a pie.
```

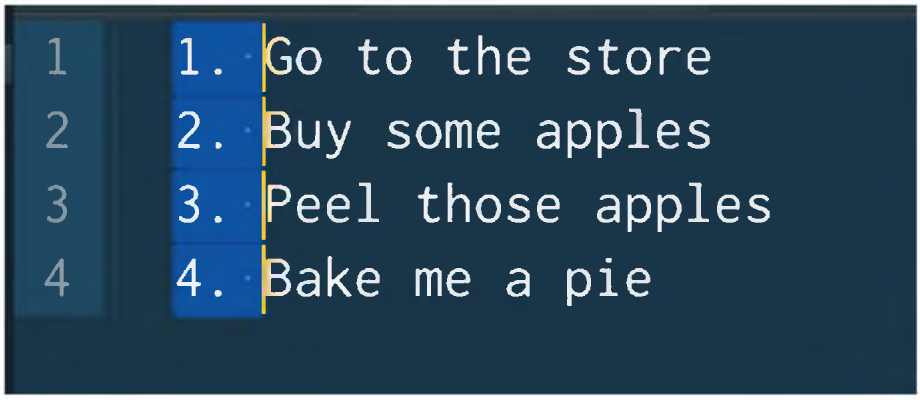
We need to convert this list to a legit ordered list. So our steps are:

1. Remove the 1,2,3 and 4 and trailing period which are typed as text.
2. Wrap each line with a list item `<li>` and `</li>`
3. Wrap all the lines in an unordered list `<ul>` and `</ul>`
4. Indent the list items

We *could* just do this manually. It might take a few minutes to wrap and indent, but since we are essentially doing the same thing 4 times – once for each line – we can use multiple carets and manage each line at the same time.

The first step is to get a caret on each line. In Sublime, hold down `⌘` and click wherever you wish to place a caret. You can also use the keyboard and use `Ctrl` + `Shift` + `Up/Down` to select lines.

`⌘` + `←` to get and bring all cursors to the front of the lines. Then hold down `Shift` and select the first three lines.



Type `<li>` in front of the content, hop to the end of the lines with `⌘` + `→` and finish it off with a closing . (Note: We will learn how to do this even faster with tag wrapping later in the book).

While you still have each line selected, use `⌘` + `]` to indent all the lines.

Finally, head to the top of the document with `⌘` + `↑` and type `<ul>` doing the opposite for the `</ul>` at the end.

Themes and Color Schemes

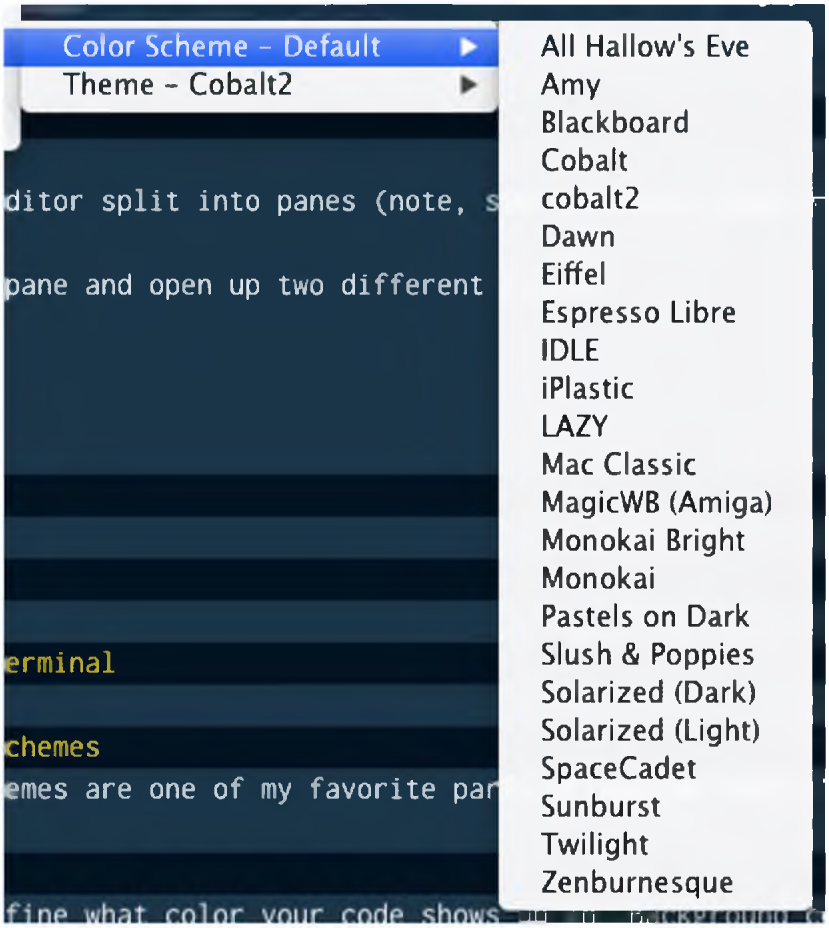
Themes and color schemes are one of my favorite parts of Sublime Text. As developers, we stare at our screen for 8+ hours a day, so it is important to spend some time evaluating our options and customizing the editor visuals to our liking. If you are still using the default colours that came with Sublime when you installed it, it's time to switch!

9.1 Color Schemes

Color Schemes define what color your code shows up in. Background color, font style and colors for things like comments, functions, arguments, scope and brackets are all defined inside the color scheme.

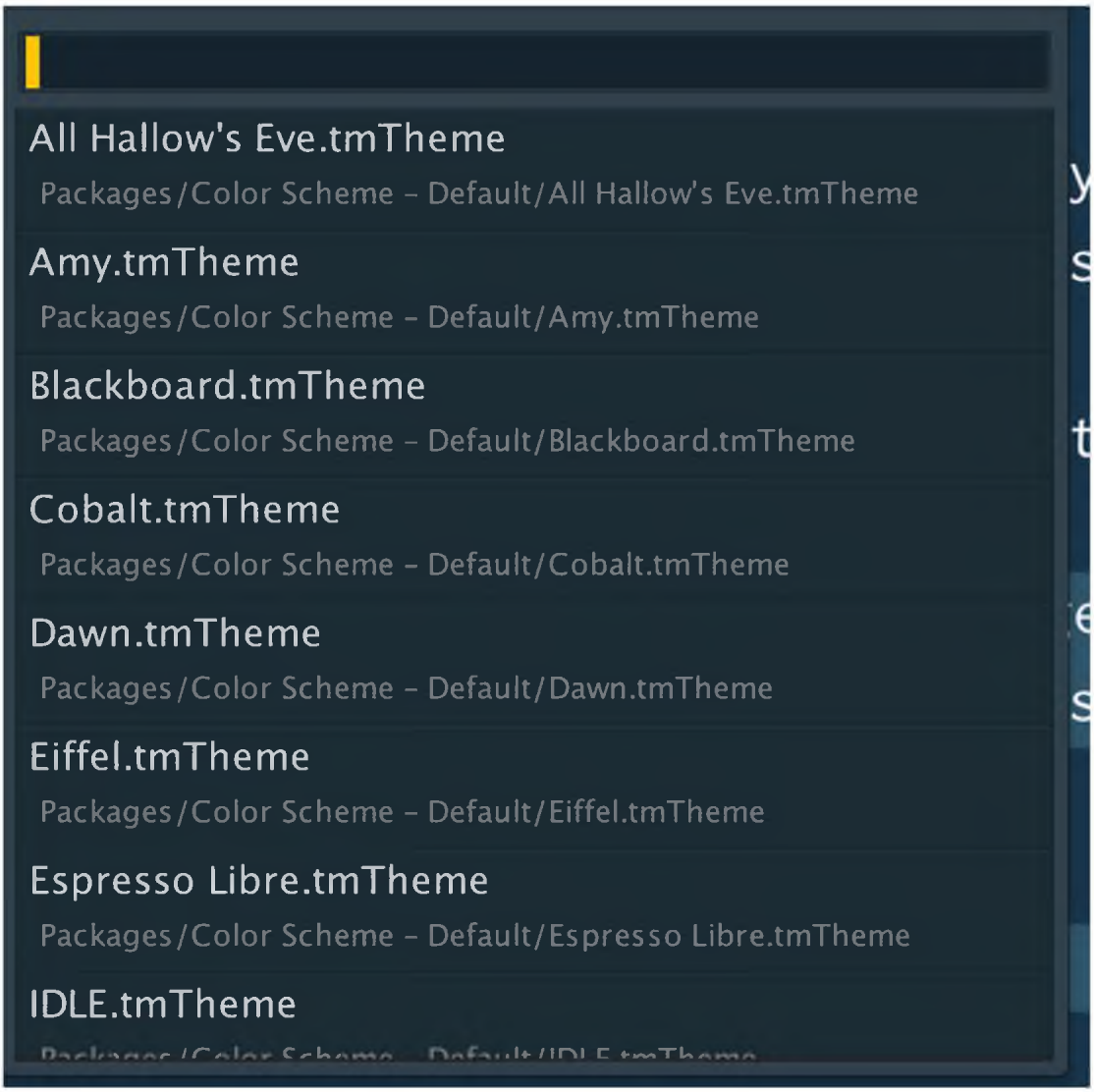
Since Sublime Text is a relatively new editor, the author has created it to be able to use TextMate theme files (`.tmTheme`), which has been around for years and has many themes available. You can see a list of available color schemes by going to `Preferences` → `Color Scheme` . These files are also located in `/Packages/Color Scheme - Default` . Switching a theme from the menu will automatically update your `Preferences.sublime-settings` file but you can code in the path like so if you please:

```
"color_scheme": "Packages/Theme - Cobalt2/cobalt2.tmTheme"
```



Color Scheme Selector Package

An even easier way to jump between multiple color schemes is to install the color scheme selector package. This will allow you change and preview color schemes right from the command palette.



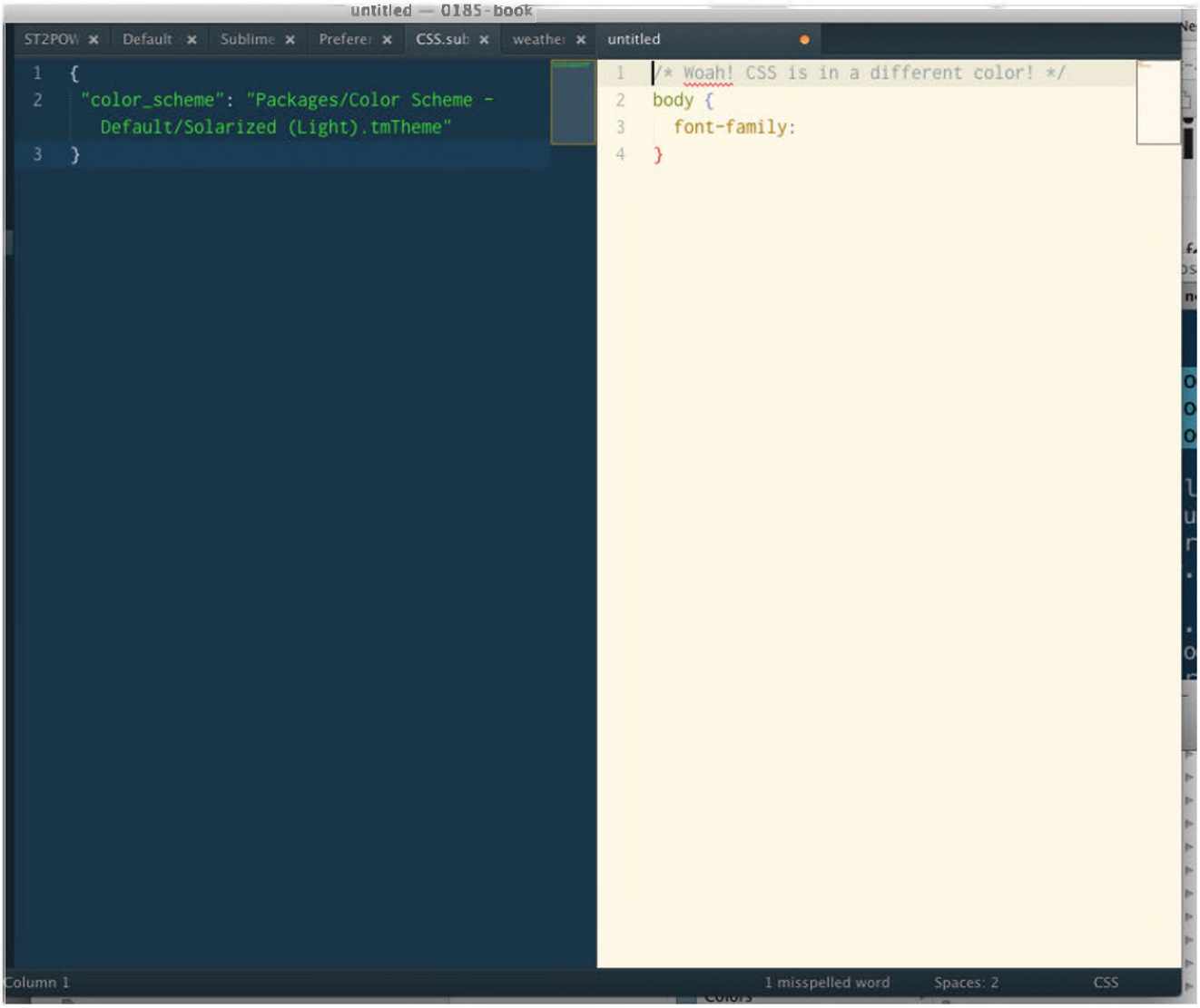
Want to use two different themes for different programming languages? Easy, just set the "color_scheme" to whatever theme you want to use in `/Packages/User/[syntax].sublime-settings`.

Here I have set the following in `/Packages/User/CSS.sublime-settings`

```
"color_scheme": "Packages/Color Scheme - Default/Solarized (Light).tmTheme"
```

It overrides the default of

```
"color_scheme": "Packages/Theme - Cobalt2/cobalt2.tmTheme"
```



The best part about color schemes is that they are just XML, which means you are free to crack open your color scheme and edit it to your liking. To create your own theme, I recommend starting from an existing theme that is similar to how you enjoy coding. It's a good idea not to edit the existing `.thTheme` file inside the previously mentioned folder - you will run this risk of having this overwritten with future updates to Sublime Text. Instead copy the `.thTheme` file into your `/Packages/User` folder and rename it to something like `myTheme.tmTheme`.

Finally, it's a good idea to replace the following six lines with a little bit about your theme.

```
<key>author</key>
<string>Wes Bos</string>
<key>comment</key>
```

```
<string>Tweaked and refined Sublime Text theme based on the original cobalt</string>
<key>name</key>
<string>cobalt2</string>
```

Sublime text monitors the `Packages` folder for any files changes. So, like your user settings, any change to a theme file will automatically be reflected upon each save, this makes editing color schemes a breeze!

9.2 Themes

Themes are something that are totally unique to Sublime Text. Themes control how the actual chrome of the editor looks. Things like the tabs, sidebar, search bar and buttons are all controlled with a theme. Themes are made up of JSON `.sublime-theme` files which include settings for width/height pixel sizes, opacity values, RGB color values and paths to images.

The first theme past the default one Sublime Text comes with, Soda Theme, was done by the talented Ian Hill of Buy Me a Soda. This theme includes entirely redone icons and took a slimmer approach to tabs.

I've also created one based on the Soda Theme, with the help of Kyle Knight, to go along with my Cobalt2 theme. Either the Soda theme or Cobalt2 are a huge improvement on the default Sublime Text UI. I'd like to encourage you to fork one of these themes and tweak it until it's **just right**.

9.3 Finding Themes

Themes can be found all over the net. If you aren't happy with one of the prepackaged ones, take a look at the following resources to find a theme that suits your fancy.

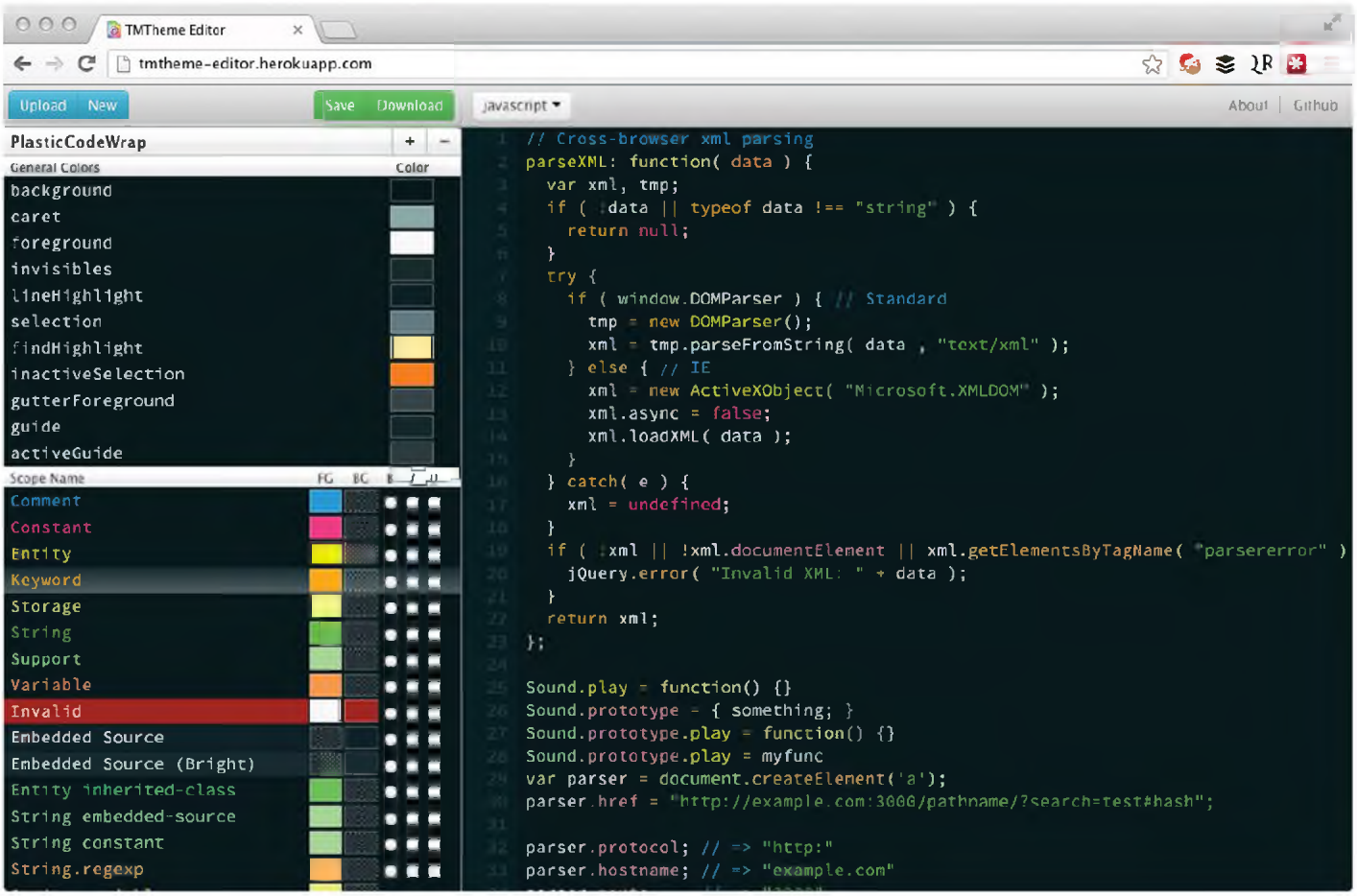
User submitted Textmate/Sublime Text Themes Repo on github: https://github.com/daylerees/colour-schemes http://textmatetheme.com/

Handy Tools

For a more visual approach to editing your theme, there is a fantastic tool created by Allen Bargi that lets you visually edit your color scheme.

The entire thing is open source and written in HTML5 with Node.js on the back-end, so you can run your own version and improve upon it. Code available at https://github.com/aziz/tmTheme-Editor

You can try the live version at http://tmtheme-editor.herokuapp.com/



Snippets

Snippets are small (or large), predefined pieces of code that you use repeatedly to speed up development productivity. If you find yourself writing the same or at least similar code over and over, you may wish to create or source a snippet to make that faster.

At their simplest, snippets consist of a `tab trigger` that opens up into a larger piece of code. Snippets can be peppered with **placeholders** which let you set defaults and spots which you can tab through to.

10.1 Creating Snippets

To create a new snippet, go to `Tools` → `New Snippet...` and you will see the snippet boiler place which is just a little XML. By default most of it is commented out, but let's uncomment the snippet and step through each part.

```
<snippet>
  <content><![CDATA[
I ${1:love} to make ${2:snippets!}.${3}
]]></content>
  <tabTrigger>hello</tabTrigger>
  <scope>source.python</scope>
  <description>A sample snippet</description>
</snippet>
```

Content

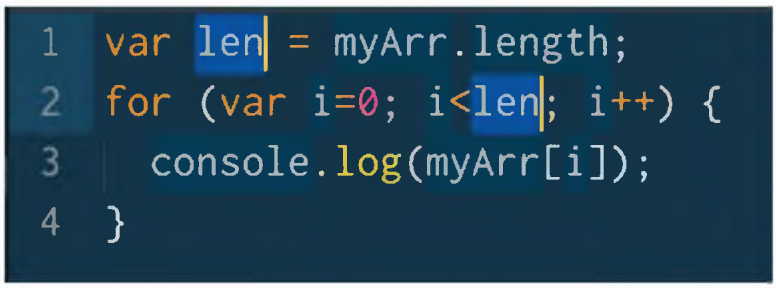
The content is the code that will show when the snippet is triggered. Because snippets are XML, they need to be inside of `<![CDATA[` and `]]>` in order for them to work properly. Inside these tags, you can paste whatever code, in whatever language, you wish.

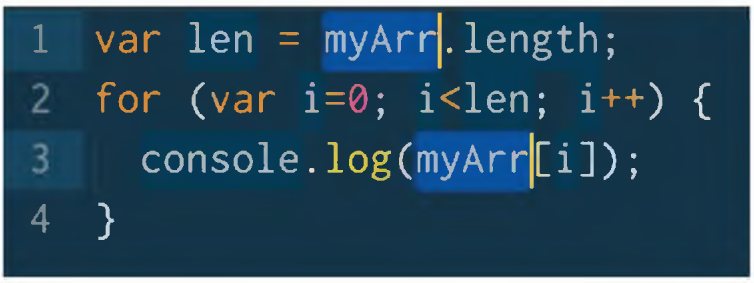
Snippets can be peppered with `placeholders` which let you set defaults and spots which you can tab through to. Placeholders are noted with `${1}` where the number is the order in which you tab through. We can also set the default text with a similar syntax of `${1:default text for this placeholder}`

Placeholders can also be used multiple times within a snippet. This is helpful when you need to type the same variable name a few times, such as in a JavaScript for loop:

```
var len = myArr.length;
for (var i=0; i<len; i++) {
  console.log(myArr[i]);
}
```

Chances are you may want to use the variables `len`, `myArr`, and `i`, but if you don't, you can just tab through and change them all at once!

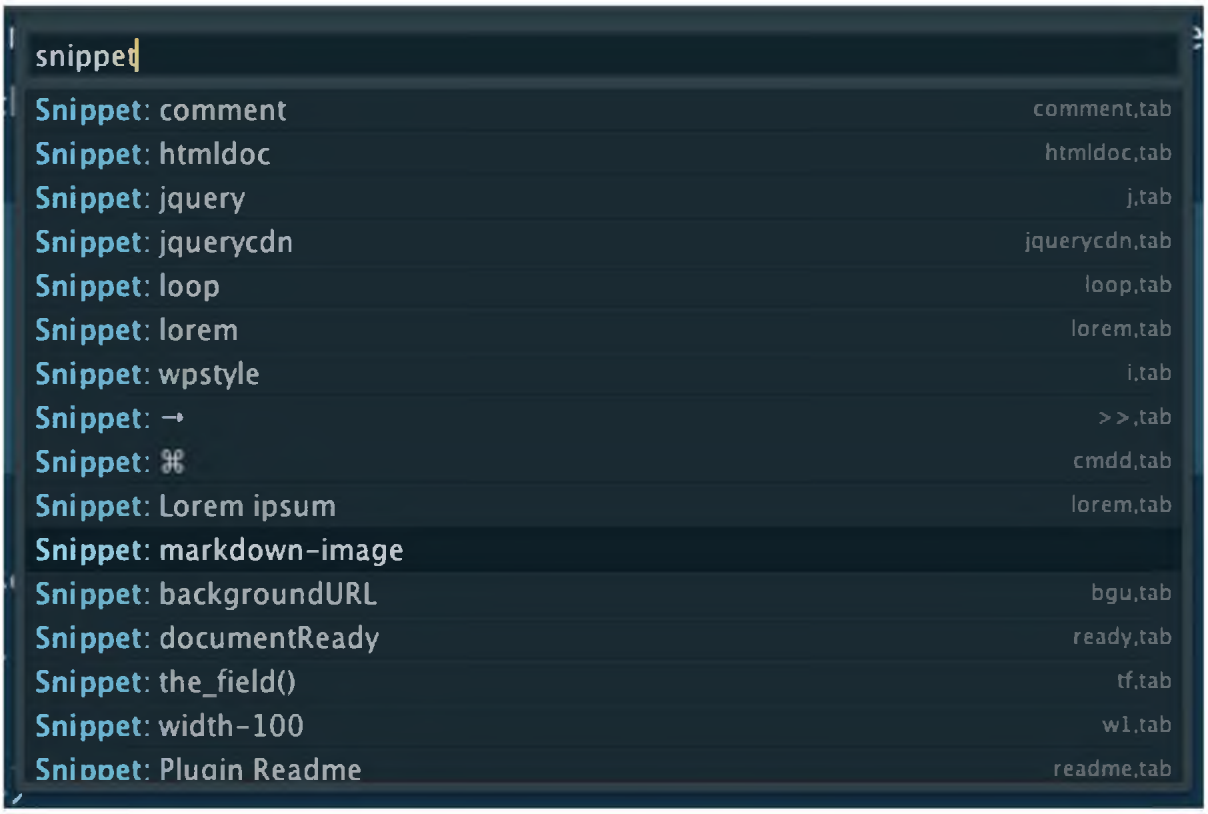




```
1  var len = myArr.length;
2  for (var i=0; i<len; i++) {
3    console.log(myArr[i]);
4  }
```

Tab Trigger

We have already learned that all the snippets are available through the GoTo Anything menu, and this is a good way to find a snippet that you don't use very often.



For more frequently used snippets, you should assign a **tab trigger** that allows you to type and expand on the go.

For the above javascript snippet, I assigned the tab trigger of `jsfor`. Now, when I wish to use that snippet, I simply type `jsfor` followed by the tab key to expand.

```
<tabTrigger>jsfor</tabTrigger>
```

Snippet Scope

You may be wondering why I had to call my last snippet `jsfor` instead of just `for`. What happens when we have a for loop snippet for JavaScript but we also want one for use in python?

Well, I could have just called it `for` and assigned a scope to it. Setting the scope as `source.js` will expose this snippet to you only when editing JavaScript files or writing JavaScript within `<script></script>` tags.

The scopes are sometimes hard to pin down and they aren't always as clear as `scope.js`.

If you are having trouble finding the scope, an easy way to find it is to open your packages folder `Preferences` → `Browse Packages…` and find the folder associated with your language. Inside of that folder will be a `.tmLanguage` file. Open it and search for `scope`. You will find a line that looks similar to this, revealing what the scope name for that language is.

```
<key>scopeName</key>
<string>source.css.less</string>
```

Description

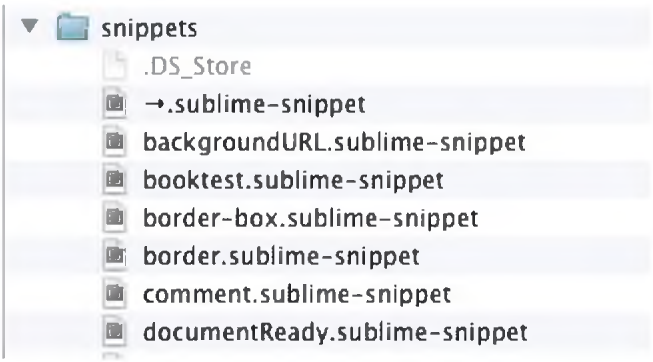
The description is pretty straightforward. If specified, the description will show up in the command palette. When not specified, the filename is used. I recommend using keywords so it's easy to reference your snippet in the command palette.

I use the fancy Unicode arrow → quite a bit in this book so I've made a simple snippet for it, giving it a keyword description and an easy tab trigger of `>>`

```
<snippet>
  <content><![CDATA[
→
]]></content>
  <tabTrigger>>>></tabTrigger>
  <description>→ Fancy Unicode Arrow</description>
</snippet>
```

Saving

Finally, when it comes to saving your snippets, you can put them anywhere in your packages folder. If your snippets are part of a package you are developing, keep them within your package, otherwise I like to create a folder called `snippets` inside of my `User` folder.

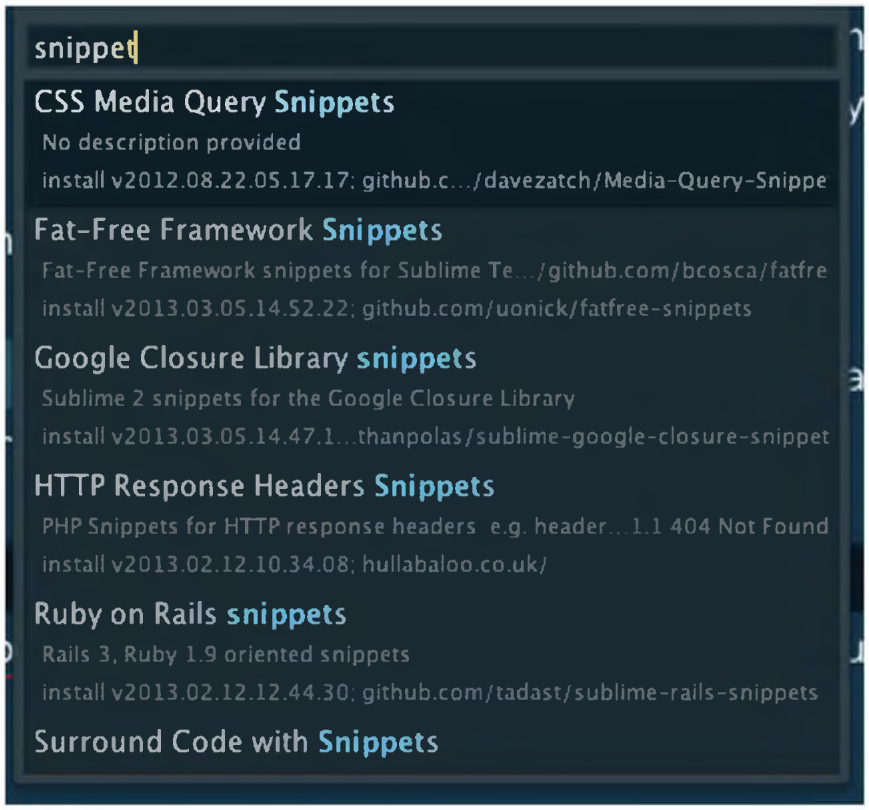


Snippets are the file type `.sublime-snippet`, so make sure you append that onto the end, as sublime doesn't assume you are saving a snippet

10.2 Finding Snippets

The are *loads* of snippets online for your enjoyment, and I encourage you to go out and find a set that meets your type of development.

The best spot to find snippets is in package control. You can browse right inside the editor by typing "Install" into the command palette and searching for "snippet", or browse the online community directory located at Will Bond's website



You can also find snippets by doing a search on github. The new search feature lets us filter for files or repos that include `.sublime-snippet` files.

https://github.com/search?utf8=%E2%9C%93&q=jquery+extension%3A.sublime-snippet&type=Code&ref=searchresults

Efficient Searching, Finding and Replacing

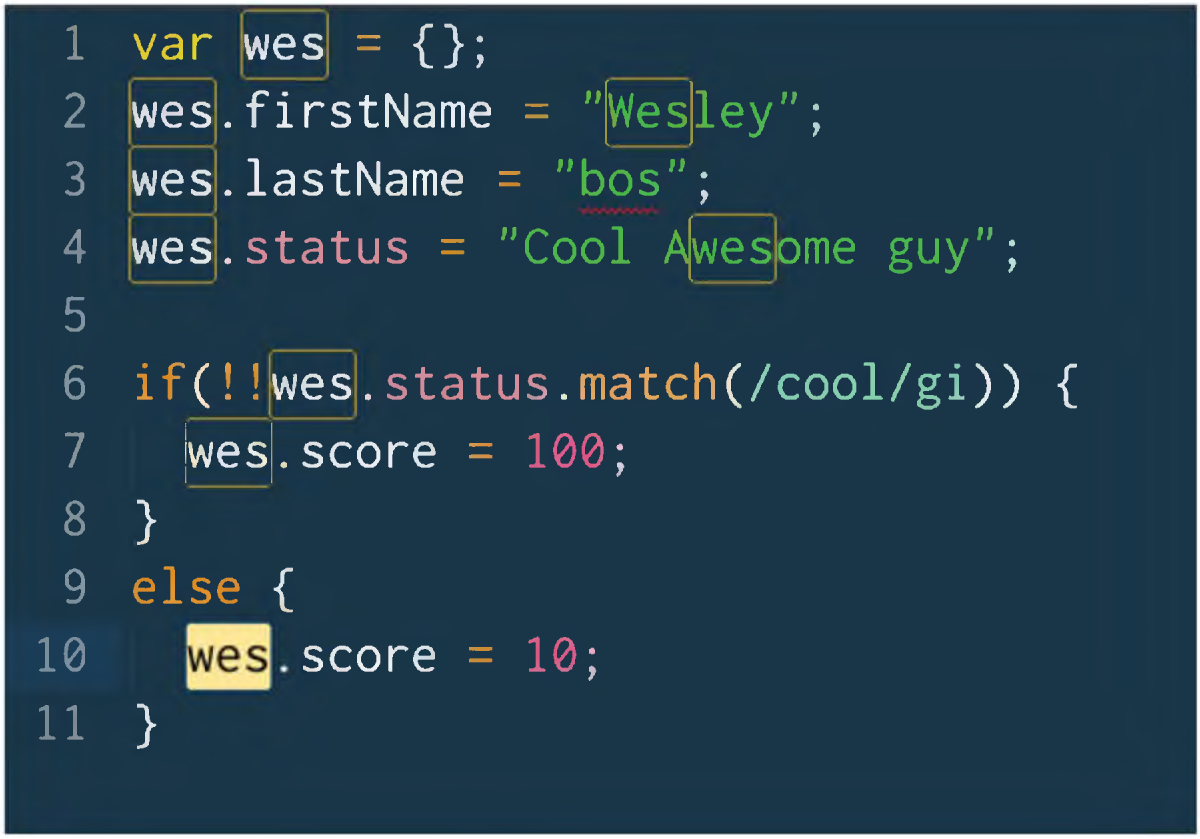
Search and replace you say? Does this even need to be mentioned, let alone an entire chapter in this book?!

Yes, yes it does! The search functionality within Sublime Text is both very powerful and extensive. It isn't exactly clear how to use everything and I often talk to people who are confused about its functionality. So, let's dive in and get a better understanding.

For this section's examples, I'm going to use a piece of JavaScript that has the word "wes" multiple times throughout it.

```
var wes = {};
wes.firstName = "Wesley";
wes.lastName = "Bos";
wes.status = "Cool awesome guy";

if(!!wes.status.match(/cool/gi)) {
  wes.score = 100;
}
else {
  wes.score = 10;
}
```

Here is a quick visual on all the instances of the three characters `w`, `e`, and `s` in a row.



11.1 Searching Inside of a Document

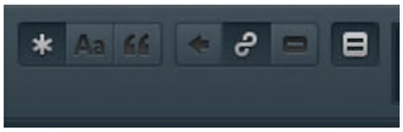
Keyboard shortcut in `Cmd/Ctrl` + `F` will open the most basic of search panes at the bottom of the editor where you can then type in your search. If you have a word selected at the time you hit `Cmd/Ctrl` + `F`, that phrase will be auto-populated into the search box.

To cycle through search results you can either keep hitting the `enter` key or use `Cmd/Ctrl` + `G`. If you go too far, adding `Shift` to either of these key combos will reverse your direction.

Once you have found your match and you want to "get off the ride" and start editing your document, simply hit the `esc` key and the editor will return to your document with your search term highlighted.

11.2 Search Options

There are a number of options available when you are searching. Most of these are fairly self-explanatory, but I will run though them just for reference sake:

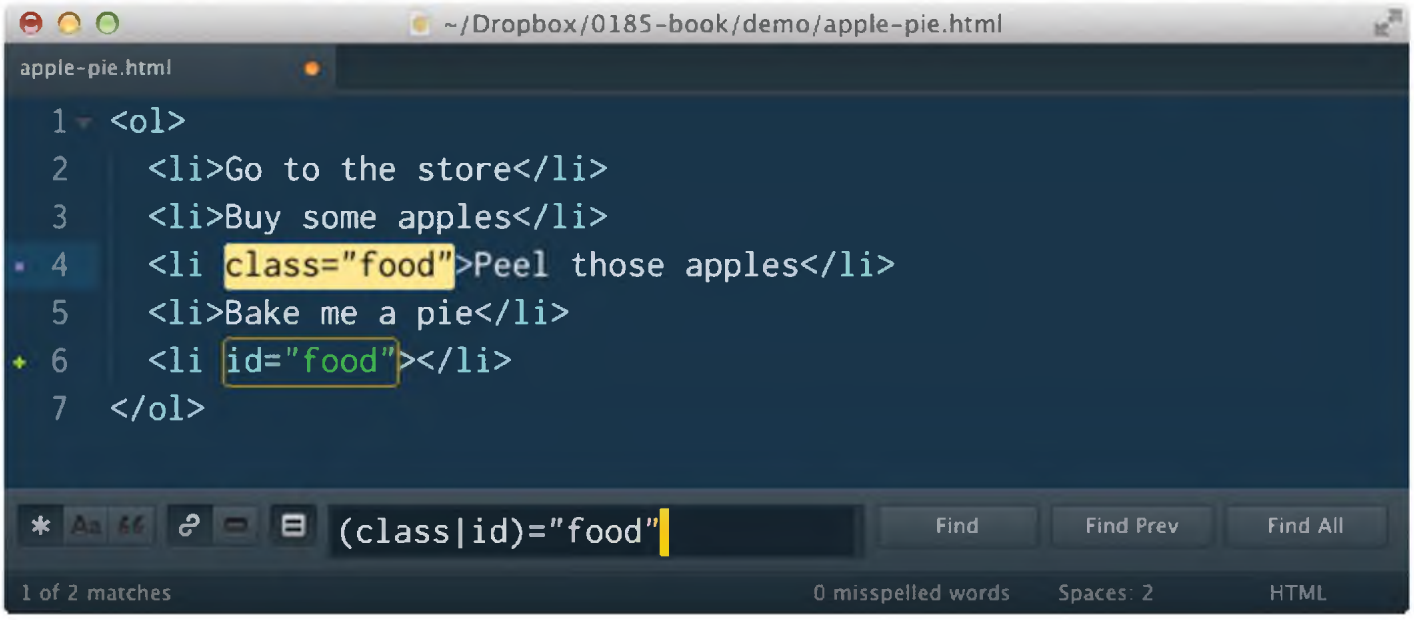


Regex Search

Rather than just matching a string, there are times that you will need to match a bunch of content that can only be accomplished with a regular expression. If I flip on this setting, I'm able to search against any regex within my document.

You may have a document where the author went whitespace crazy, instead of manually going through each line and deleting them, I can do a search for lines that start with newlines – `^\n`

Another example would be searching for multiple attributes that are set in a certain way. For example, all classes or ids set to food would be matched with `(class|id)="food"`



If you are new to regular expressions, I recommend using the regexr and regexpr tools to help you learn and get yourself up to speed.

Case Sensitive

Do you want the search to ignore the case of a word? For example a search for `Wes` matches `wes`, `WES`, `wES` and so on. If so, leave it off.

Whole Word

Do you want to match partial words? For example if I search for `text`, should it match the last four letters of `sublimeText`?

Show Context

When show context is turned on, the search results will include the previous two lines of code and the next two lines of code for a total of 5 LOC. This is helpful in understanding the context of a variable but can make your search results very long.

When it is turned off, the results only display the line with the match.

In Selection

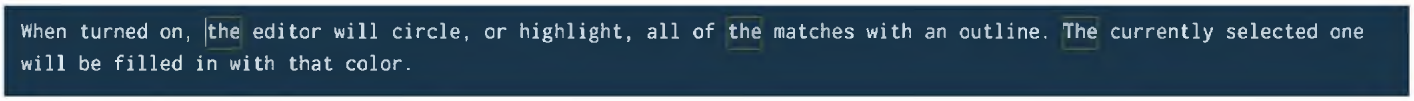
When enabled, Sublime will limit the scope of the search to the currently selected text. This is useful when you are looking to find all instances of a variable within a certain selected area.

Wrap

When searching within a document, the first selected result will be the one that comes first after the caret. When wrap is turned on and the last result in the document is tabbed through, the search will start again at the top of the document, essentially wrapping around.

Highlight Matches

When turned on, the editor will circle, or highlight, all of the matches with an outline. The currently selected one will be filled in with that color.



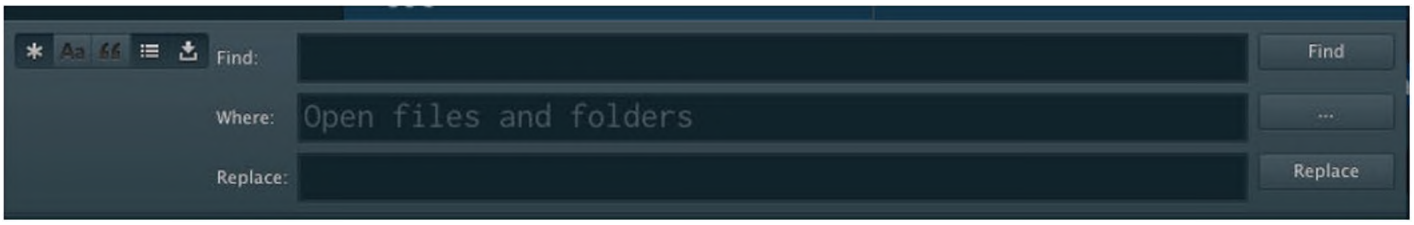
Use Buffer

When searching inside an entire project, the default of Sublime will open it in a small panel at the bottom of the editor. With *use buffer* turned on, search results will open in a new tab, or as Sublime sometimes calls it, a buffer.

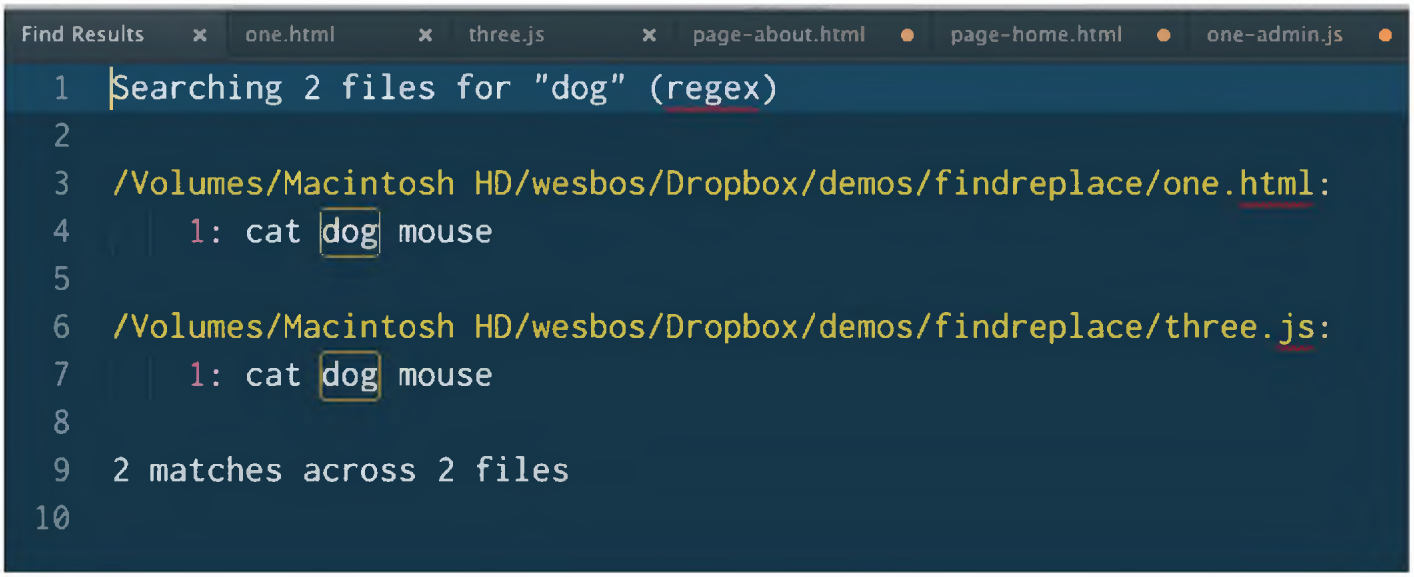
11.3 Search & Replace Inside Projects and Folders

After the last section, searching inside a single document should be a piece of cake, but what about when you want to search across an entire folder or project?

`Cmd/Ctrl` + `Shift` + `F` will bring up an extended panel that allows for this extended searching.



Using the find button here will open a new pane that lists the results that were found. You are then able to click on these results to open the corresponding file and location.



The important field here is the **where** input. By default it will search all open files and folders. So, if you have a project open it will search the entire folder, or if you have dragged several files/folders into a Sublime window, the scope is limited to all those files/folders.

Now if you want to limit the scope of your search, we have a few options:

Specific Folders – Multiple absolute folder paths each separated by a comma. Helpful for when you are doing a find/replace on multiple similar codebases.

Project – Similar to the default, you can search the current project by entering `<project>` into the *where* input.

Include Filter – Limit your search to a specific file, file type, or file name. Here you can enter a full file name `index.js`, and then use an asterisk to wildcard a file type or name. Examples of this include:

`*.js` – all .js files

`index.*` – index.html, index.js, index.php, etc...

`page-*.*` – page-home.html, page-about.php, etc...

Exclude Filter – Same as above, but append a minus sign `-` before the search. Examples of this include:

`*.js, -*.min.js` – all but minified .js files

`index.*`, -*.js – all index files except `index.js`

`page-*.*, -*admin*` – page-[anything].[anything] except files with *admin* in the name.

open folders – `<open folders>` will set the scope to open folders and not include any other currently open windows.

open files – `<open files>` will set the scope to currently open files and not include any currently open folders.

Combining Filters

All of the above filters can be stacked together by separating them with a comma. Putting multiple *where* inputs will evaluate files to match one or more match – not all of them. Make sure to use the minus operator `-` to exclude files from search.

11.4 Incremental Find

Another great feature of Sublime's search is the *incremental find* which allows you to combine multiple carets and search into a multi-select based on your search term. For more on this, using this same example, read the section on multiple carets.

11.5 Other Searching Tips

Just like a terminal environment, pressing your keyboard's up arrow in the search field will allow you to cycle through your previous searches.

Moving Selecting, Expanding and Wrapping

This section is filled with tips that will make you say "Ah! I didn't know you could do that!". Little tricks that will help with with everyday coding tasks. This is where Sublime really starts to shine.

12.1 Moving Lines and Code Blocks

12.2 Line Bubbling / Swapping

Line Bubbling is one of my favorite parts of Sublime Text. It allows you to easily move sections of code up or down through your document without having to cut and paste.

To use line bubbling, hold down `⌘` + `Ctrl` (`Ctrl` + `Shift` on Windows) and use your `↑` up and `↓` down arrow keys to move the lines around. This works for single and multiple lines.

12.3 Reindenting Code Blocks

Now, if for some reason you end up with either a block of code or an entire file filled with hard to read, messy and poorly indented code, reindent will be your best friend.

To automatically re-indent your code, simply select the code you want to indent and then from the menu `Edit` → `Line` → `Reindent`

Sublime doesn't ship a keyboard shortcut for this by default. No problem, we can make our own. Pop this little snippet into the `Preferences` → `Key Bindings - User` file.

```
{ "keys": ["super+shift+r"],  "command": "reindent" }
```

For more on indentation, refer for the tabs vs spaces section of this book.

12.4 Joining

Joining two lines together is something developers do all the time. I often see users just hitting the backspace key until the line below jumps up onto a single line.

To join lines, place your cursor on the top line and hit `⌘` (`Ctrl` for windows/Linux) + `J` . Remember J for join.

This also works with multiple cursors, you you can bring everything onto a single line extremely quickly.

12.5 Duplicating

Need to duplicate a line of code? Chances are you are may be currently using this method:

1. Selecting the entire line with your mouse
2. Copy via `⌘` + `c`
3. Paste via `⌘` + `v`

That is too much work! Duplicate a lines quickly with `⌘` + `Shift` + `D` or `Ctrl` + `Shift` + `D` for Windows/Linux users.

12.6 Deleting

No longer need a line? Similar the one above, save yourself the manual work and delete lines with one quick key combo of `Ctrl` + `Shift` + `K`.

Using `Ctrl` + `K` without the shift key will delete from the caret to the end of the current line.

`⌘` + `backspace` will delete from the caret to the beginning of the current line.

> **Side note:** `⌘` + `x` will cut the current line without having to select it. Handy if you need to paste somewhere else.

Deleting Words

Deleting words seems easy, right? Select a word with your mouse and hit the backspace. Or maybe you have figured out how to select word-by-word and then hit delete. We have even easier ways to do this.

Backspace word-by-word with `Ctrl` + `backspace`.

Forward delete word-by-word with `Ctrl` + `fn` + `backspace`.

Deleting Letters

Backspace obviously removes a letter at a time.

`fn` + `backspace` does a forward delete. Or just the `delete` key on windows.

12.7 Inserting a line before

Need a new line? `Enter` works just great if your cursor is at the end of the line, but if your cursor is in the middle of a line you can use `⌘` + `Enter` and `⌘` + `Shift` + `Enter` to add lines after and before the current lines respectively. Windows and Linux users swap `⌘` with `Ctrl`.

This is especially helpful because you do not need to be at the end or start of a line for it to work - your cursor can be anywhere inside the current line and they will jump above/below without disrupting the current line.

12.8 Wrapping with tags

We will learn all about Emmet, which will hel you code HTML tags much faster, among many other things, but what if you already have content that needs to be marked up in HTML?

Well, Sublime comes with an extremely handy "wrap with tags" functionality. To use it, simply select the text you wish to wrap with tags, and hit `Ctrl` + `Shift` + `W`. This is also accessible via the menu under `Edit` → `Tag` → `Wrap selection with tag`.

By default, Sublime will wrap your content is a paragraph `p` tag.

```
<p>By default, sublime will wrap your content is a paragraph `p`
tag. </p>
```

Type any tag name and both the opening and closing tag will be updated:

```
<blockquote>Type any tag name and both the opening and closing tag
will be updated:</blockquote>
```

You can even continue adding attributes without de-selecting the closing tag. Just press space and keep typing!

```
<p class="so-awesome">You can even continue adding attributes
without de-selecting the closing tag</p>
```

12.9 Jump to BOL or EOL

Jump to the beginning / end of line in sublime text is simply `⌘` + `→/←`.

If you wish to select text as well, add `Shift` to the above combination.

You may have noticed that if you have **line wrapping on**, sublime only brings you to the visual EOL/BOL, not the hard EOL/BOL. So, if you are trying to select an entire paragraph, it only grabs the top line:

```
1109    You may have noticed that if you have **line wrapping on**, sublime
        only brings you to the visual EOL/BOL, not the hard EOL/BOL. So, if
        you are trying to select an entire paragraph, it only grabs the top
        line:
1110
```

Sometimes this is the desired outcome, but when you wish to select the entire line, simply hit your arrow key twice.

`⌘` + `→` + `→` to jump to hard end of line

`⌘` + `Shift` + `→` + `→` to select to hard end of line

> Windows/Linux users Users replace `⌘` with `Ctrl`. You may be rolling your eyes at this tip because you have the almighty `end` key on your keyboard.

12.10 Moving to ends and starts of lines and files.

Jumping or selecting an entire file work just the same, but instead using the `↑` and `↓` arrows.

`⌘` + `↓` to jump to end of file

`⌘` + `Shift` + `↓` to select to end of file

> Again, Windows and Linux users replace `⌘` with `Ctrl`.

12.11 Selecting, Jumping & Expanding

Selecting text and expanding your selection is a large part of making your work flow as efficient as possible. Every time you reach for your mouse to select a segment of text, you are slowing yourself down.

Most text selections are things you do over and over, so it is worth figuring out how we can do it with just a few keystrokes. Let's take a look at the many ways to select and expand your text.

> Windows and Linux users, when unspecificed, replace `⌘` with `Ctrl`.

Jump by Word

Let's start things off easy: jumping around by word. Instead of grabbing your mouse, we can use `Alt` + `→` to jump by word. This is great when you wish to get a few words over - try to use this instead of mousing over.

Do this over and over until you are comfortable with the keyboard shortcut.

Select & Expand word by word

Now let's say you want to select a few of these words as you move along. Simply add in `Shift` to the mix to select and jump by word.

`Alt` + `Shift` + `→`

Select & Expand to certain words

Selecting a word comes in handy when you want to replace multiple occurrences of a word in a file, but do not want to use the potentially destructive find and replace.

To select a word, make sure your caret is somewhere inside that word and hit `⌘` + `D`. Hitting it multiple times will select the next occurrences of that word.

```
Selecting a word comes in handy when you want to replace multiple
occurrences of a word in a file, but do not want to use the
potentially destructive find and replace.

To select a word, make sure your caret is somewhere inside that word
and hit `⌘` + `D`. Hitting it multiple times will select the next
occurrences of that word.
```

You can then go ahead and perform any number of operations on the selection, such as wrapping them with a tag:

```
Selecting a <strong>word</strong> comes in handy when you want to
replace multiple occurrences of a <strong>word</strong> in a file,
but do not want to use the potentially destructive find and replace.

To select a <strong>word</strong>, make sure your caret is somewhere
inside that <strong>word</strong> and hit `⌘` + `D`. Hitting it
multiple times will select the next occurrences of that <strong>word
</strong>.
```

Like this? Read more on this feature in Chapter 11.

Jump by line

To jump to the beginning or end line in Sublime Text we use `⌘` + `←/→`. Windows users can use the `end` key in place of this.

Select & Expand to Line

Selecting an entire line is easy. We can use the technique stated above with `⌘` + `Shift` + `→` + `→` or simply use `⌘` + `L` to select the entire line.

If you have word wrapping on, you may find yourself in the situation where `⌘` + `Shift` + `→` only selects the first visible line and not the entire line:

```
1169    If you have word wrapping on, you may find yourself in the situation
        where `⌘` + `shift`+ `→` only selects the first visible line and not
        the entire line:
1170
```

To remedy this, add an extra `→` key press to select to the real end of line. So, `⌘` + `Shift` + `→` + `→`.

Using `⌘` + `L` or `Shift` + `↓` multiple times will add the next line(s) to your selection. Use `Shift` + `↑` to select lines above the current selection.

Select & Expand to Tag

This one is similar to wrap in tag, but the opposite. Need to select all text inside a tag? Hit `Shift` + `⌘` + `a`.

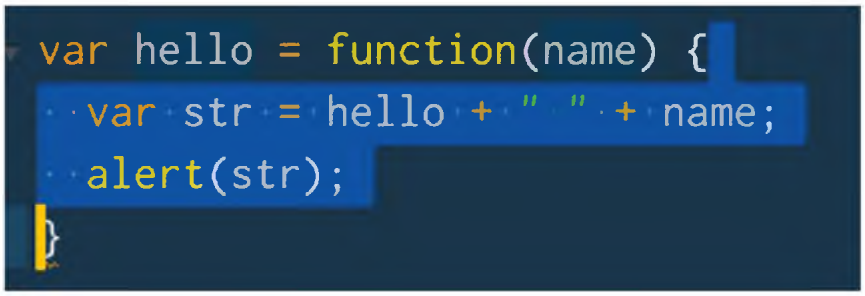
Using `Shift` + `⌘` + `a` again will select the tag as well, useful if you are working with nested tags.

Once more will select all sibling tabs and any content inside the parent.

Select & Expand to Brackets

Not writing HTML? No problem, this one will select to brackets for languages that are based on brackets.

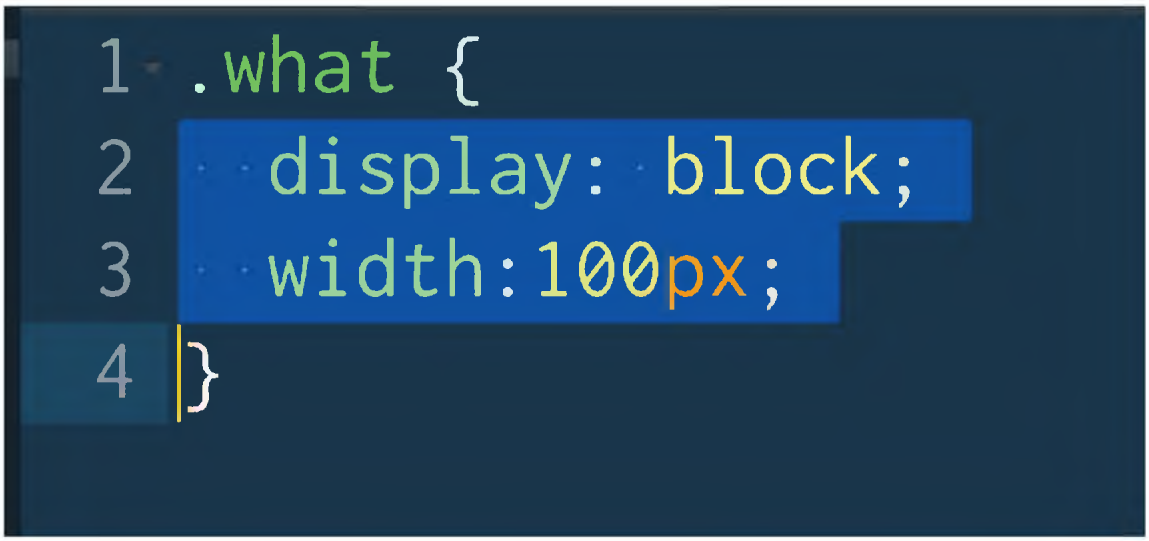
`Ctrl` + `Shift` + `M` will select to your square and curly brackets. As always, doing it a few times will increase select up multiple levels.



Select & Expand to Indent

Using a language based on indentation such as Python, Slim or HAML? Just want to select an entire block of indented code? Not to fear, sublime has fantastic indentation detection built in.

`⌘` + `Shift` + `J` will select your indented block. You can also use this in conjunction with brackets. For example, in CSS if I wanted to select everything but the selector and curly braces:



> **Note:** If you are in Sublime Text 3, the keyboard shortcut was disabled in version 3. If you wish to have it back, add the following code to your key bindings file located at `Preferences` → `Key Bindings - User`. `{ "keys": ["super+shift+j"], "command": "expand_selection", "args": {"to": "indentation"} }`

Select & Expand to Quotes

This is one that isn't built right into Sublime Text but is incredibly helpful, especially when dealing with HTML attributes or string variables.

This one comes in the form of a package. To install, simply open up Package Control and search for the "Select Quoted" package. It is available for both ST2 and ST3.

Simply hit `Ctrl` + `'`. Easy to remember because you want to control the quotes.

Once more will include the quotation in the selection:

Selection and beyond!

By this point, you get the point. There are many ways to select your text. There are a few more available under the `Selection` menu. Use this as a reference for remembering the keyboard shortcuts and take some time to get comfortable with the different types of selection.

Code Folding

When working with large documents, it is helpful to fold sections of the code so it is not visible. Code folding in Sublime text takes sections of your code and minimizes it into a single character.

The best way to get comfortable with code folding is to try it out. For this chapter, open up Sublime Text and follow along with the below code.

13.1 Practice Code

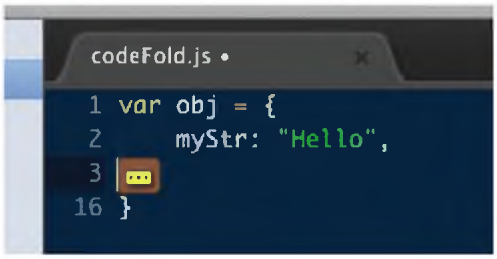
Code folding works in all languages. I've created a small snippet of JavaScript which we can use to practice. Copy and paste this into Sublime and set the document mode as JavaScript.

```
var obj = {
    myStr: "Hello",
    myFunc : function() {
        console.log("hello");
        $('.someDiv').animate({height: 500}, function(){}
            // trigger the callback
            console.log('Done');
        });
    },
    myFunc2 : function() {
        return "Just another Level 2 function";
    },
    myNum : function() {
        return "Level 2 function";
```

```
    )
}
```

13.2 Folding Selected Text

The simplest form of code folding is more of code compressing because it has nothing to do with the indentation. Go ahead a select a few lines of code and type `⌘` + `Option` + `[` (Windows users use `Ctrl` + `Shift`). You'll now see you have a little icon indicating you have folded some code.

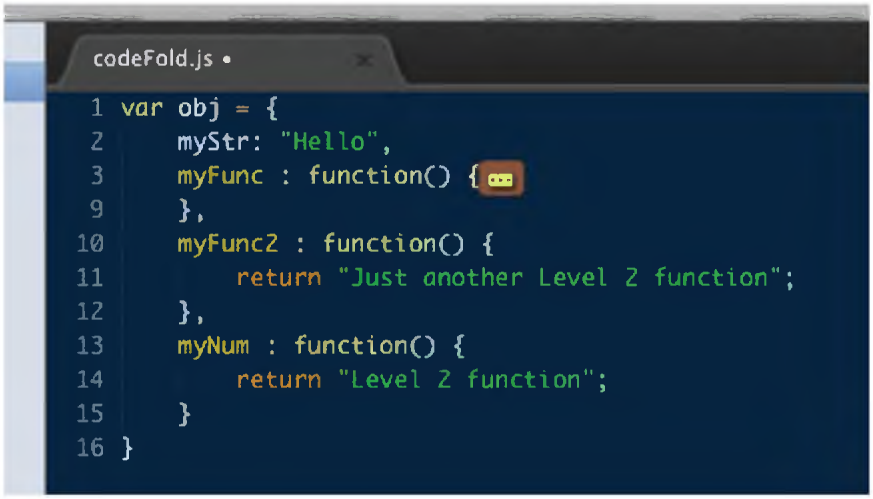


To *unfold* the code, repeat the above key combo with the closing square bracket – `]` .

Block level code folding

Selecting the code you want to fold can be a pain. If you maintain nice, clean indenting within your document, you'll be able to quickly fold and unfold several levels of code in just a few key strokes.

To fold a block, place your cursor anywhere within the block you want to fold. For our example, lets place it in the myFunc function. Now you just hit the same keys as before, Sublime will detect the nearest block level and fold it for you.



13.3 Fold Multiple blocks at once

Another handy feature of folding is being able to fold all blocks that are indented with a certain level. For this example, I want to fold all the and just see the properties/functions of `obj` .

The keyboard sho rtcut for this is a little different. We need to hold down `⌘` (Ctrl on Windows/Linux) and then tap `K` .

While you are still holding down `⌘` or `Ctrl` tap the level of code block you want to fold. In this case 2.

Confused? `⌘` + `K` , `⌘` + `2`

We now see that everything beyond 2nd level blocks are hidden. To unfold everything again, we do the same thing but with 0.

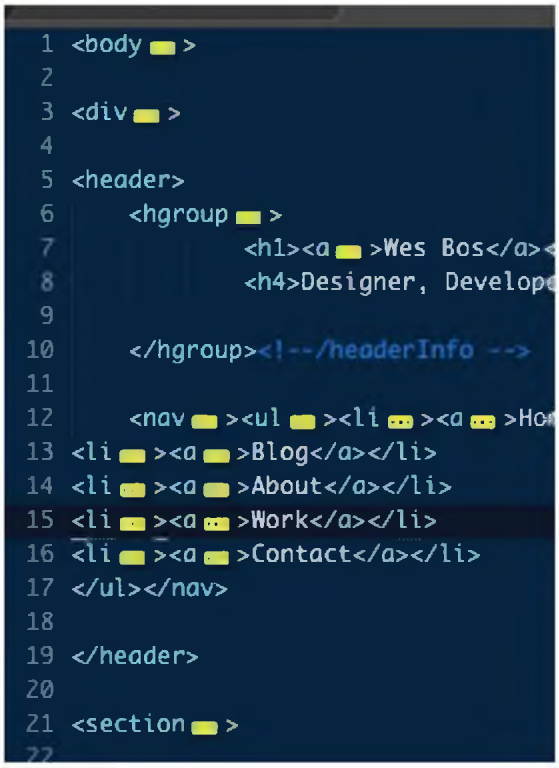
13.4 Folding with arrows

Another, somewhat new, feature to Sublime Text is the code folding arrows that show up in the gutter of the editor. These allow you to fold your code based on the indentation by clicking the arrow.

While these are fairly straightforward to use, the default setting of Sublime Text will only show them on hover. To make these visible all the time, you can use the following setting in your `Preference.sublime-settings` file.

```
"fade_fold_buttons": false
```

13.5 Folding element attributes



When working with an HTML document that is heavy on attributes, it can be helpful to hide everything but the element type. This is extremely useful when working with something like jQuery UI or Angular that have long element attributes,class names, titles, data, and aria attributes. Now, this one is all or nothing, so you can't hide them case-by-case.

To hide:

`⌘` + `K`, `⌘` + `T`

To show again:

`⌘` + `K`, `⌘` + `0`

13.6 Maintaining Folding State

If you close and re-open a file, you will notice that all of your folds have disappeared. This is frustrating when you spend lots of time folding your file. There is a package called *BufferScroll* that maintains the state of your code folds even after you close files or the editor. Read more about this package and it's many useful features in **Chapter 24**.

Projects

Projects in Sublime Text are a nice way to manage different websites or applications that you may be working on. The main benefit to using projects in Sublime Text are the ability to have specific editor settings that apply only to that specific projects. This is especially helpful when working with teams who may not have their editor setup properly for contributing.

There are two files that make up a Sublime Text project: the `.sublime-project` file and the `.sublime-workspace` file. The first being a file to hold your projects settings and the second being a place where the editor can dump user specific data. If you were to open the `.sublime-workspace` file, you would see all kinds of things from previously opened files to editor settings. You will never need to edit this file, so it's best just to ignore it.

When working with version control, the `.sublime-project` file should be checked in and shared while the `.sublime-workspace` file should not. I find it helpful to get myself into the habit of adding `.sublime-workspace` to all my `.gitignore` files.

14.1 .sublime-project file makeup

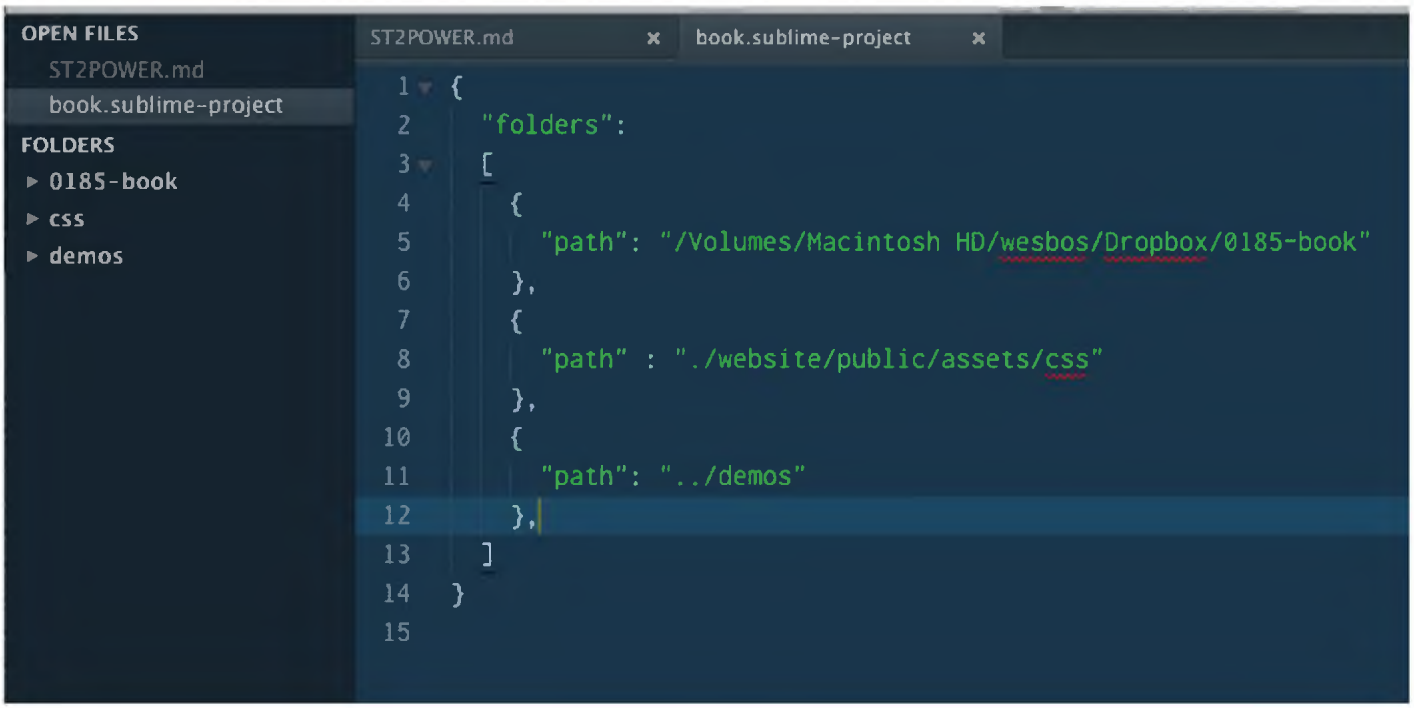
The `.sublime-project` file is, just like all other settings files in Sublime Text, a JSON formatted file. Inside of the file are stored settings that make up the three main functions

1. Folder Settings

Perhaps the most obvious, the `folders` setting tracks which folders are included in the project. This means that you are able to include multiple folders from all over your system while using absolute file paths.

If you are using absolute file paths, you do not need to house the file in your projects root directory. The `.sublime-project` file can live anywhere you please. The downside to this is that sharing the `.sublime-project` file with team members won't really work unless you have identical file systems.

If you do keep the `.sublime-project` folder in the root directory of your project, you are able to use relative file paths.



The above file, I have this Book's project open.

1. The first path I include the entire book folder

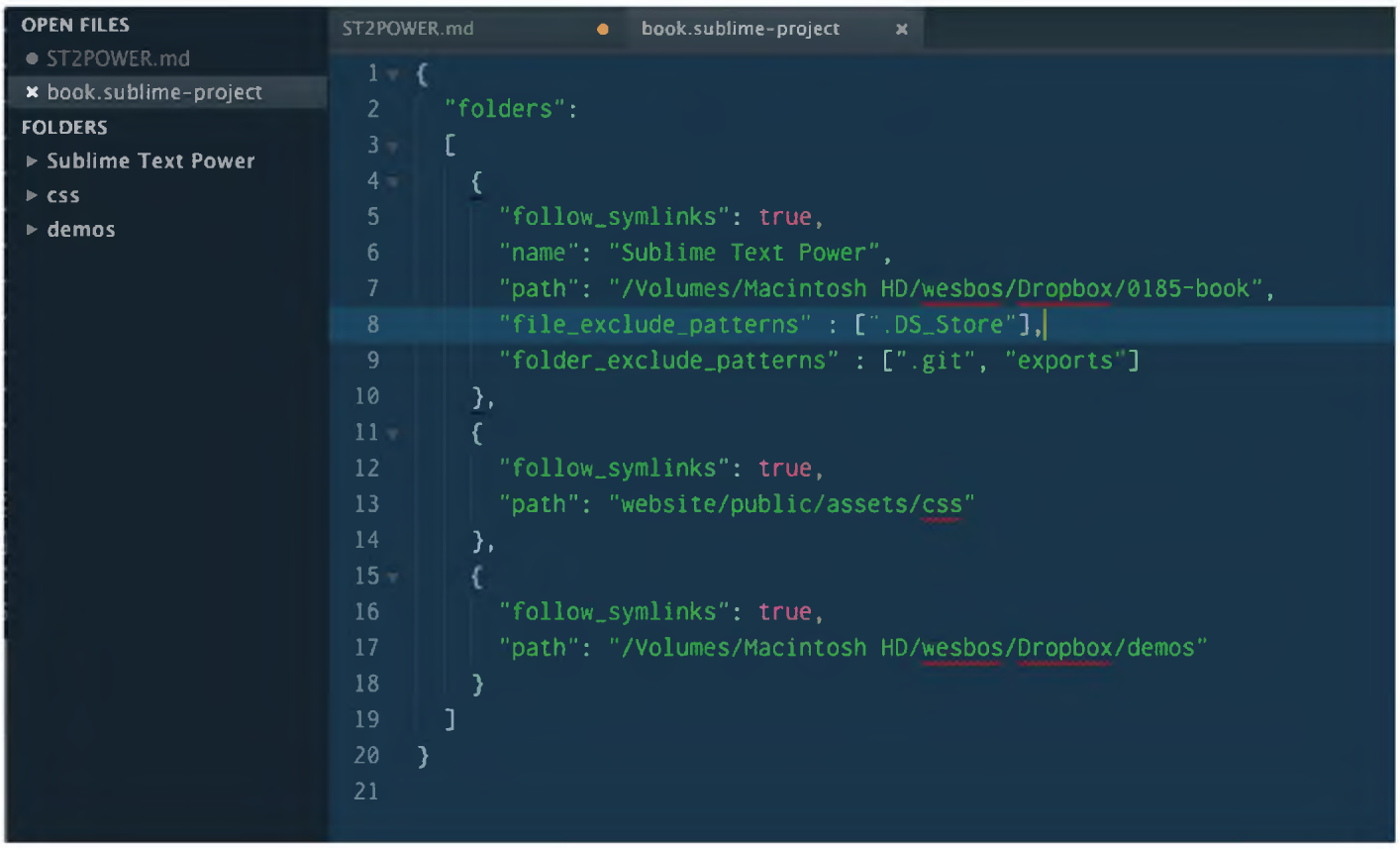
2. The second path, I've "pinned" a nested CSS folder that I access regularly. Even though the `css` folder is inside the first `0185-book` folder, I'm able to the use dot-forward slash `./` to access relative file paths.
3. In the third path, I'm able to navigate one folder up to my main websites folder where I house all my coding demos. To do this we use the two dots and a forward slash `../`

Taking it even further, there are a number of settings you can use to fine tune your folder list in the sidebar.

`"name"` allows you to rename each of these paths to a more user friendly name. In the example below I've renamed my book's folder to "Sublime Text Power User"

`"follow_symlinks": true` is set automatically when adding a folder to a project. If you even need to set that to false, it's there.

`"file_exclude_patterns"` and `"folder_exclude_patterns"` work just as they normally do but are constrained to the current folder. Many of these files/folders can be ignored at a global level, but this will be helpful when you want fine grain control over which files and folders you are ignoring.



2. Settings Overwrite

Settings are exactly the same as the ones found in the your `.sublime-settings` files. If you add settings to your `.sublime-project` file, they will overwrite any settings you have previously set. This is extremely helpful for code quality settings such as tabs-vs-spaces or linting rules, which can often differ from one project to another.

In terms of specificity, syntax specific settings will always overwrite project specific settings.

3. Build Systems

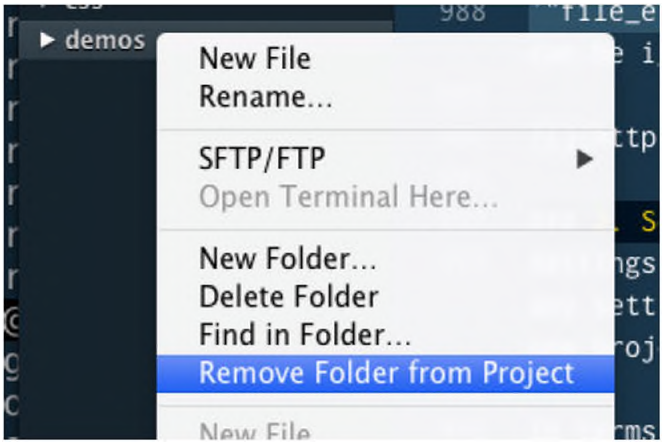
Here you are able to add build systems that are specific to this project. For more information on build systems, make sure to cover the Build Systems section of this book.

14.2 Creating and updating projects

Every time you open a blank window in sublime you are essentially creating a new project. However, to maintain your folders, settings and build system you must save the project into a `.sublime-project` file. Once you have opened at least one folder with sublime, you can save it via `Project` → `Save Project As...`

To add a folder to your current project, simply drag and drop the folder into the sidebar. You are also able to access this via the `Project` menu.

To remove a folder, edit the file manually or use the context menu when right clicking on the folder.


demos
New File
Rename...
SFTP/FTP
Open Terminal Here...
New Folder...
Delete Folder
Find in Folder...
Remove Folder from Project

Mastering Keyboard Shortcuts

Using the keyboard is far superior to using a mouse in almost all ways. Whether you are new to coding or been around for a long time, there are many benefits to re/training yourself to lean on the keyboard:

15.1 Negating carpel tunnel

I'm not a doctor, but I can tell you I had a really bad RSI / carpel tunnel problem. Moving your hand with a mouse for 8+ hours a day isn't great for you. The less you move your arms, the better.

15.2 Reducing mistakes

We are all human, and we all make stupid mistakes. Developers are error prone. The less we rely on typing out entire strings of code, the less errors we will make. We have all spent 30+ mins on a problem that turned out to be a simple syntax error or spelling mistake - let's stop doing that!

Spelling Mistaeks

```
<img scr="dog.jpg">
```

Improper Syntax

```
<a href"about.html">About ME</a>
```

Nesting Troubles

```
<p>Hello <em>world</p></em>
```

Forgetfullness

```
.wrapper {
    float:left;
    width:100%
    color:blue;
}
```

The above are *all* things we have done before in whatever langauge we are using. Can we cut down on them with keyboard shortcuts? Absolutely!

15.3 Becoming a more efficient coder

Finally, this might be the most obvious one is that using keyboard shortcuts can really speed up development time.

Replacing a 10 second task with a 2 second task may not seem like a huge improvement - what is 8 seconds saved? Over the course of a week, you can save yourself hours of unnecessary grunt work. Improve your productivity by just 30 minutes a day and you can add an extra *three weeks* onto each year.

15.4 The process of becoming a keyboard shortcut master

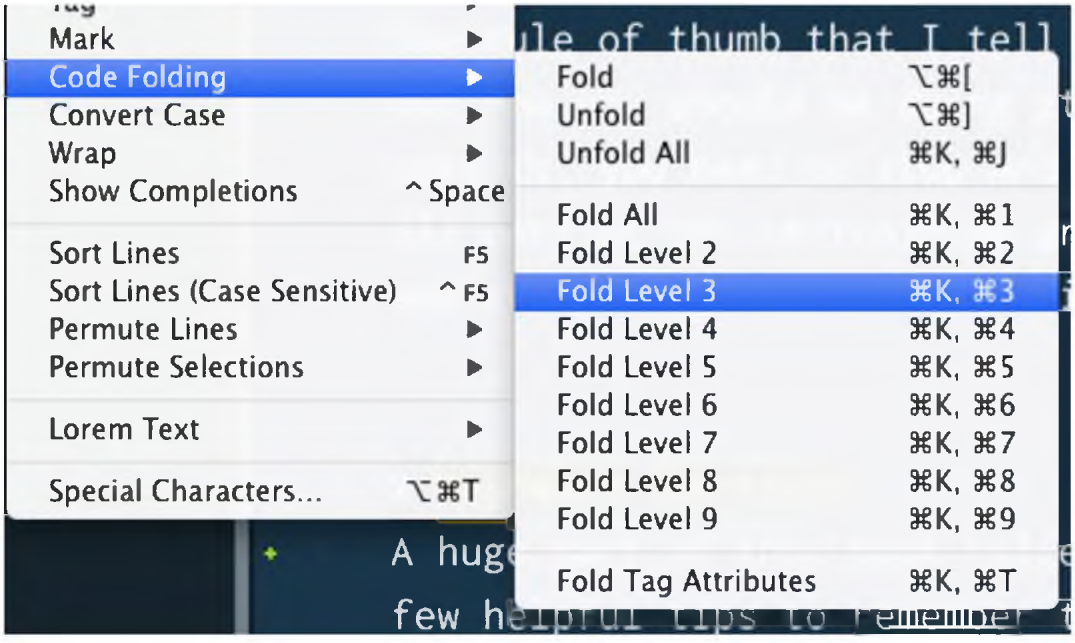
Becoming a keyboard shortcut master doesn't come overnight, you need to change the way that you work and break old habits. In most cases, you will actually slow yourself down for the first few times you use a shortcut. This is okay, you are breaking and re-forming new habits that will save you *tons* of time in the future.

The rule of thumb that I tell people is that when you do something twice that feels inefficient, **stop** and figure out how you should be doing it. This will be annoying at first because instead of doing it the 10 second way you are used to, you need to stop and spend upwards of a minute finding and remembering the correct keyboard shortcut. This is all part of the re-training process. After a few times of being slow, you will commit the shortcut to muscle memory and everything will be high-fives and sunny from there on out.

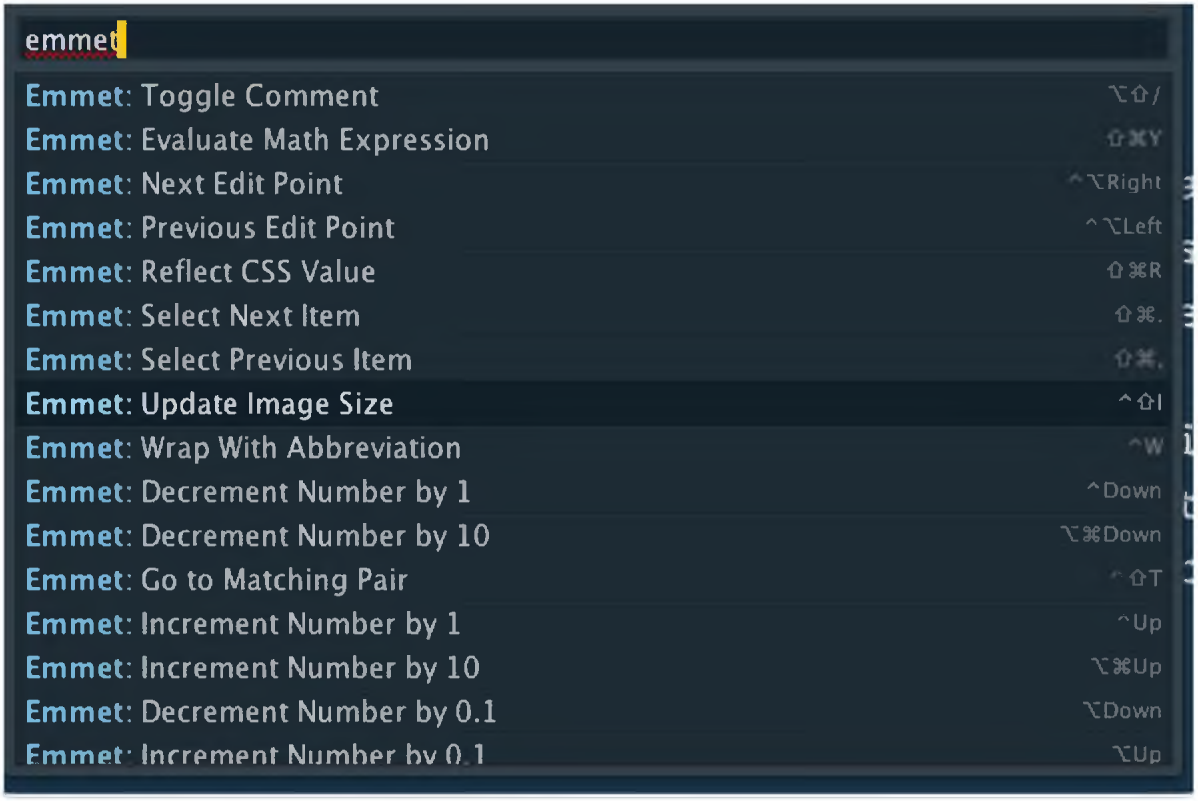
15.5 Referencing Shortcuts

If and when you forget what the keyboard shortcut is, always remember that almost every command is also available to you via the command palette or the Sublime Text menu.

This means that you can pop open the command palette or the menu at any time to find the command you are looking for. While you are there, don't just click the command you want, take note how most commands have a keyboard shortcut right next to it.



If a command has a keyboard shortcut, it will show it

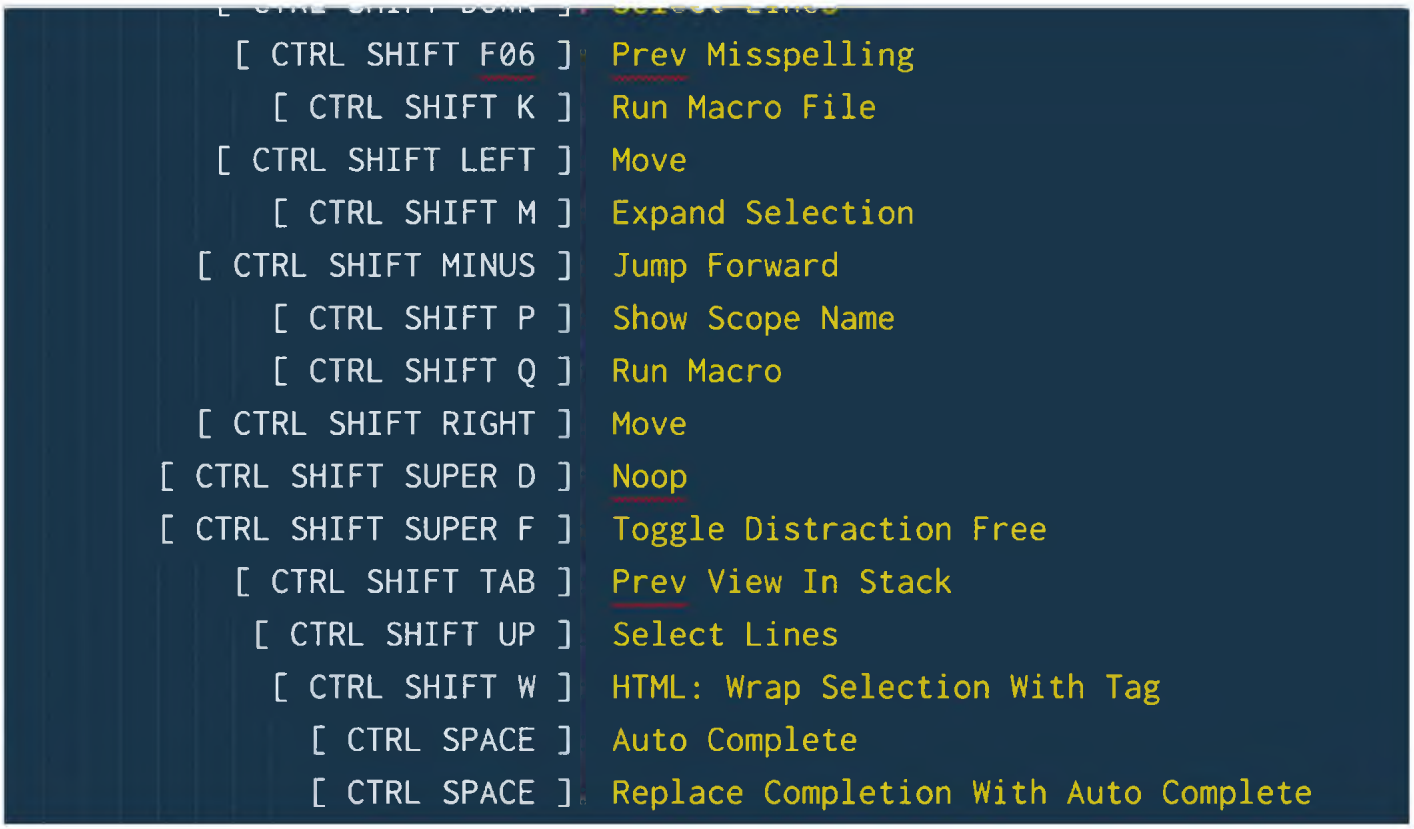


The same goes for anything in the command palette

Once you have done this a few times, you will start to commit the shortcut to memory and things will start to move much faster for you!

There is also a fantastic package called Keymaps which takes inventory of both native commands and ones introduced by packages compiling a list and cheat sheet of all available key maps.

filter for the command you are looking for



Part of a compiled list of my keymaps

15.6 What the heck are the ^ ⇧ ⌘ ⌥ ? + SUPER Keys!?

One of the most frustrating things for keyboard shortcut newbies and pros alike is looking at an image like the one in the last section and trying to remember what ^ ⇧ ⌘ ⌥ and SUPER are.

Spending some time to commit these to memory will greatly improve your quality of life. Each one comes with a really silly way to remember.

^

The **Control** or **Ctrl** key.

Think of it as a button on a game *controller*. You use the up arrow to **control** where your character in the game goes.

Everyone calls this a hat. Think of it as a hat someone in power, or *control* would wear. A mad crazy **control**lers hat.

⇧

The **Shift key**. Most windows keyboards have this symbol on it, but Mac users aren't as lucky. We usually use Shift to make a Capital Letter. So the **up** icon is the one we use to create uppercase letters.

⌘ / Super / ⊞

Most shortcuts will refer to something called a SUPER key.

On mac, this is the Command, or ⌘ key.

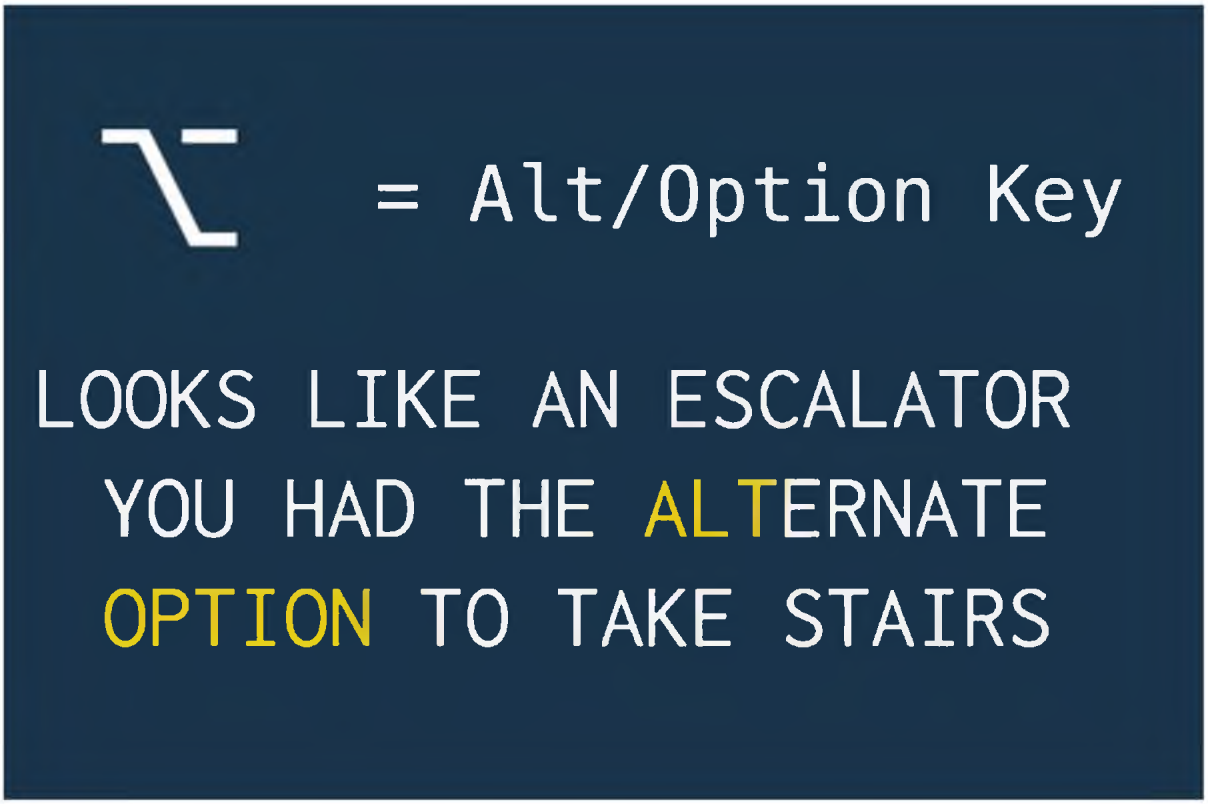
On windows, this is the Windows key ⊞

On Linux, it will differ depending on your OS and personal preferences. If you are on Linux, chances you already know what your super key is.

⌥

The **Option / Alt** key.

This one looks like a man riding on an escalator. He is lazy because he had the **alt**ernate **option** of taking the stairs.



?

Escape. This is only useful if you make use of the Vim plugin for Sublime.

The arrow is trying to escape from the circle.

15.7 Creating Custom Keyboard Shortcuts

A few times this book has referenced functionality that doesn't have a keyboard shortcut associated with it. Sometimes you may also wish to re-map specific functionality to a new keyboard shortcut. Creating your own keyboard shortcuts in Sublime Text is easy.

Default Shortcuts

Just like with our Sublime Text preferences, there is a file full of the default keymaps that you are encouraged to view, but should never edit. This file is over-written when Sublime Text is updated.

To view the default file, open it via `Preferences` → `Key Bindings - Default`. Scan though this file quickly to get a feel for the syntax and then we will break it down in the next section.

Your Custom Keymap File

Custom keymaps are operating system specific - this means that windows, Linux and OSX each will have their own keymap file called: `Default (Windows).sublime-keymap`, `Default (Linux).sublime-keymap` and `Default (OSX).sublime-keymap`. To create or open yours, open `Preferences` → `Key Bindings - Default` and the editor will open your os specific file.

This file is also JSON formatted as one big array of keymaps. So, if you have an empty keymap, you can start it like so:

```
[
    // start here
]
```

Each keyboard shortcut is stored as an object `{}` and take four properties.

For this example, I'm going to be creating a keyboard shortcut to open the Sublime Text Preferences folder. We know we can press `⌘/Ctrl` + `,` to open the settings file, so I want to use `⌘/Ctrl` + `Shift` + `,` to open the folder on my computer.

keys

Keys is an array of keys that make up the shortcut. Most of the time you will have a single value composed on two or more keys.

In this case, I want to use the combination of `⌘` `Shift` and `,`, so my object looks like this so far:

```
{ "keys" : ["super+shift+,"] }
```

Note how there is a `+` in between each key. If I wanted to have a two-stage keyboard shortcut, such as `⌘` + `K`, `⌘` + `B` to toggle the sidebar , we would have multiple entries in the array:

```
{ "keys" : ["super+k","super+b"] }
```

command

Once we have *which keys* need to be pressed, we need to specify the command that gets run.

There are hundreds of commands that could be run and while there is no official list of commands from Sublime Text, the docs project has done a fantastic job at collecting and documenting them as they find them.

The other way to find out which command you want is to browse the Default file for ones that are already implemented.

The command we want is `open_dir`, which as you may guess will open up a directory in Finder/Explorer. Our keyboard shortcut now looks like this:

```
{ "keys" : ["super+shift+,"], "command" : "open_dir" }
```

Many of these commands are for package developers, here are a handful of ones you may find helpful:

`move` and `move_to` advance the caret by any number of words/characters/lines/etc..

`run_macro` will run a specific macro, more on this in the Macros chapter.

`expand_selection` will expand your selected text to bol, hardbol, eol, hardeol, bof, eof, brackets, line, tag, scope or indentation.

`new_file` will create a new file - I have remapped this to `⌘` + `T` as every other program uses this convention to open a new tab.

`new_window` opens a new instance of Sublime Text - I have remapped this to `⌘` + `N` just as many other programs do build

args

Arguments provide more information to the command. Our command right now just runs "open_dir". Which one? We can use arguments to specify this.

Each command will have different argument names - it is best to look them up either in the unofficial docs or in the Default file. In our case, we need to set the `dir` property on the arg object. At this point, it's nice to format the object on multiple lines:

```
{
    "keys" : ["super+shift+,"],
    "command" : "open_dir",
    "args" : {
        "dir" : "$packages"
    }
}
```

Above we used a system variable - there are a number available to use in keyboard shortcuts and they are the same ones available in build systems. For more read the build systems chapter and check out the documentation for a list of all variables.

context

The final property is `context` and it allows keyboard shortcuts to be bound only in certain circumstance, or contexts.

For example, we may want to bind the `enter` key only when the autocomplete dialog is open, otherwise enter should do as it was normally intended.

Or, maybe a keyboard shortcut should only be bound when we are editing a bit of CSS, and not JavaScript:

Context has four property/values:

The `key` is the context you are testing for, such as `num_selections`, `setting.spell_check` or `selector` (current scope). Check the docs for a full list.

The `operator` can be `equal/not_equal`, `regex_match/not_regex_match` or `regex_contains/not_regex_contains`.

The `operand` is what is being tested against, can be `true`, `false`, a `number` or a `regex`.

`match_all` is by default set to false, but allows you to turn on matching for every selection when using multiple carets / selections.

example

Let's say I'm working with PHP and I need a quick way to open up my localhost server to the current project. I can use the context selector to keybind `⌘` + `K`, `⌘` + `O` to open the local server in my default browser, but only when editing PHP files:

```
{
  "keys" : ["super+k","super+o"],
  "command" : "open_url",
  "args" : {
    "url" : "http://localhost/project-name"
  },
  "context" : [{
    "key" : "selector",
    "operator" : "regex_match",
    "operand" : "source.php"
  }]
}
```

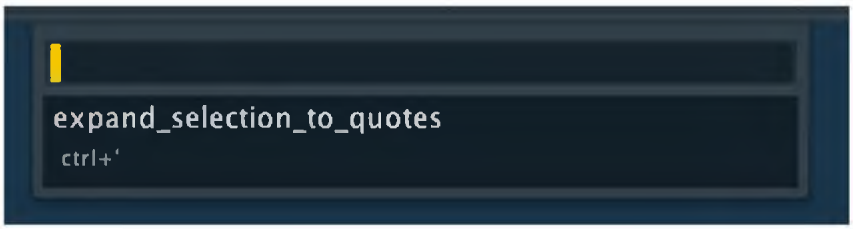
15.8 Dealing with Keyboard Shortcut Conflicts

Once you have enough packages, chances are that some keyboard shortcuts will conflict each other.

If you find you are having trouble tracking down which commands conflict, installing the package FindKeyConflicts will produce a list of conflicting key shortcuts:

```
139 [ctrl+j]
140   run_emmet_action                    Emmet
141   jshint                              JSHint
```

You can also search by command, so if there is a specific command you are having trouble with, it will produce a list of packages and then their available commands:


expand_selection_to_quotes
ctrl+'

Macros

Macros are a small, yet powerful, part of Sublime Text that allow you to record a series of steps and then play them back. Many developers think macros are just for large repetitive tasks, but the real power comes from shortening the number of keystrokes it takes to do a simple task. Taking 3-4 keystrokes into a single keystroke can greatly speed up your development time; this are micro improvements but really add up over the long run.

One example that I use very frequently is when writing JavaScript functions. Using Sublime, the function looks like this with the `|` denoting the cursor.

```
var someFn = function() {
    |
}
```

Sometimes I need to add a semi-colon to the end of the function. To do that, I would:

1. Arrow down
2. Type a semi colon
3. Arrow back up
4. Tab inwards
5. Start writing my code

Since I only want a semicolon some of the time, it doesn't make sense to create a snippet here, but rather record a macro that does these steps for me.

16.1 Recording a macro

To record a macro, simply setup a piece of code that you would normally be editing. In my case, it's the snippet of JavaScript above. Then follow these steps:

1. Go to `Tools` → `Record Macro`
2. The macro is now recording, so go ahead and go through the steps of the macro. For my macro, I will follow steps 1-4 above.
3. Stop recording when finished via , `Tools` → `Stop Recording Macro`
4. Save the macro via `Tools` → `Save Macro`. This will go in your `User` directory and just like snippets, I like to create my own folder for macros.

Thats it, you should now see your macro available under `Tools` → `Macros` → `User` which will execute when you click it.

16.2 Adding a keyboard shortcut

If you have to use the above menu technique to execute a macro, it doesn't really save you any time. The smart thing to do would be to bind it to a keyboard shortcut. To do this, open your keyboard bindings via `Preferences` → `Key Bindings – User`.

Depending on if this is your first keyboard shortcut, this file may or may not have text inside of it already. This file is an array of keyboard shortcut objects, so go ahead and add a line like this:

```
[
    { "keys" : ["super+;"], "command": "run_macro", "args": {"file": "function-semicolon}"} }
]
```

The above set the keys `super (⌘)` and `;` to run this macro. You can see we are using the command `run_macro` which accepts an argument `file` of whatever you named your macro file, less the `.sublime-macro` extension.

If you have multiple keyboard shortcuts in here it will look like below. Remember to put a comma after each item except for the final one.

```
[
    { "keys" : ["super+;"], "command": "run_macro", "args": {"file": "function-semicolon}"} },
    { "keys": ["super+shift+,"], "command": "open_dir",  "args": {"dir": "$packages"} }
]
```

16.3 Editing Macros

If the macro recording didn't turn out exactly as you wished, you can always edit them by hand. If you open any `.sublime-macro` file and see the array of action objects. Using the Sublime Text commands referencee you are able to build out some pretty complex macros.

Running, Testing and Deploying with Build Systems

Build systems allow you to run your command line tools right from Sublime Text - these are commonly commands such as `compass`, `make`, `rake` or `grunt`, but can really be anything that is available from the command line.

It's important to note here that build systems aren't a replacement for tools like Gulp, Grunt, Rake, Ant or Make, but rather a way to run these tools from the editor and see the response without leaving the editor. Depending on the workflow, you may find you do not wish to replace having a terminal window open with a Sublime Text build system. If you heavily use the terminal to work with your apps, you may wish to forgo build systems and use your terminal alongside Sublime Text.

17.1 Creating a build File

Build tasks are housed in a `.sublime-build` file which is a JSON file that holds a number of options for running

Your own build files can live anywhere in your `User` folder. I recommend creating a `build` folder inside of the your `User` folder that will hold all of your build tasks.

At it's simplest, a build file takes the `⌘` argument. This is the command that will be executed when the build is run.

Let's build something to run "ls -l" (listing the directory files)

```
{
    "cmd" : "ls -l"
}
```

You can then select the build system from `Tools` → `Build System` and hit `⌘` + `B` to run it.

You may or may not see build results. If you don't, open the build results tab (`Tools` → `Build Results` → `Show Build Results`). If you rather hide the results, set `"show_panel_on_build": false` in your `Preferences.sublime-settings` file.

Selectors

Just as we can with snippets, we can limit the scope of a build system to a specific language with the `selector` attribute.

Let's say we only want the above build to work in PHP files:

```
{
    "cmd" : "ls -l",
    "selector" : "source.php"
}
```

You can also pass an array of multiple sources:

```
{
    "cmd" : "ls -l",
    "selector" : ["source.scss","source.sass"]
}
```

Then set the build system to `Automatic` under `Tools` → `Build System` and this build system should be selected automatically.

Variables

When running commands, chances are you need some information about the current file or folder to run that command. This is where variables come in .

Let's say I'm working on `style.scss` and I want to compile it to `style.css`.

We can easy make a build system that looks like this:

```
{
  "cmd": ["sass","style.scss", "style.css"],
  "selector" : "source.scss",
}
```

Note how above instead of specifying a single cmd string, I've passed an array. It is recommended that you use an array of commands rather than spaces in a string.

Now, make sure the build system is set, and run it with `⌘` + `B`.

The above is great, but there are a few questions to ask yourself here:

1. what if our file isn't called style.scss?
2. what if we want to rename the output file to something else? like output-style.css
3. What if we want to move the directory?

With variables, we can pass along information about the current file and directory. For instance, the above build system is better written as:

```
{
  "cmd": ["sass","${file_name}", "${file_base_name}.css"],
  "selector" : "source.scss"
}
```

What about different folders? We can use the project folder variable

```
{
  "cmd": ["sass","${file_name}", "${project_path}/css/${file_base_name}.css"],
  "selector" : "source.scss"
}
```

For a list of all variables, check out the documentation

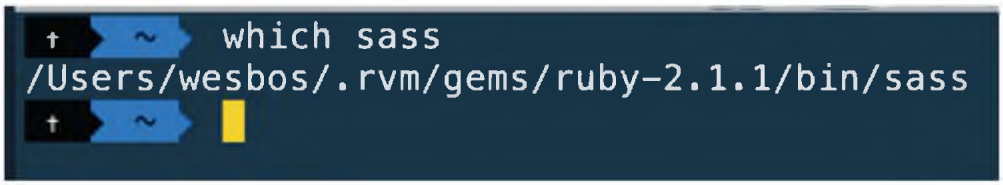
Capturing Errors

If you run the command and there is an error, you want to know where the error occurred. Rather than spew everything into the console, build systems have two settings - `file_regex` and `line_regex` which allow you to find the exact file and line where the error occurred.

Path Issues

Sometimes you might run into issues where Sublime Text can't find the command you are trying to run. This often happens when you are running multiple versions of ruby, or the executable is located somewhere other than your $PATH variable.

For this we need to specifically tell sublime where it is. For instance, I had trouble running the sass gem because I use RVM for multiple versions of ruby. To fix, open up a terminal window and type `which sass` to see where the gem is installed. We can then use the `path` option in our build file like so:




```
{
  "cmd": ["sass","style.scss", "style.css"],
  "selector" : "source.scss",
  "path" : "/Users/wesbos/.rvm/gems/ruby-2.1.1/bin/sass"
}
```

Cross Platform

You can even set different options based on your operating system. Handy if you switch between mac/linux/pc often:

```
{
    "selector" : "source.scss",
    "osx" : {
        "cmd" : ["rm", "$file_name"]
    },
    "windows" : {
        "cmd" : ["rd", "$file_name"]
    }
}
```

17.2 Build Resources

Build systems may seem simple, but there are many variables that go into creating a build system depending on what language, OS or method you choose. Here are a few fantastic resources and further reading on the subject:

http://docs.sublimetext.info/en/latest/reference/build_systems.html

http://addyosmani.com/blog/custom-sublime-text-build-systems-for-popular-tools-and-languages/

Bookmarks

Bookmarks are a great way to reference points in you code base making them easier to jump to. I find I use bookmarks for temporary reference points inside a document while I use the Goto Anything palette for all other jumping around.

With Bookmarks, you are bookmarking that line of code, not the line number. Bookmarks stay with the line you initially intended, so adding or removing code before the bookmark will not affect it. They do not, however, stay with you when you copy/cut/paste lines of code. If you delete a line of code, the bookmark will be assigned to the line of code below the deleted line.

> **Note:** Bookmarks also remember the columns (or character) that you bookmarked it at, so it is possible to have multiple bookmarks per line.

You can access the bookmark menu under `Goto` → `Bookmarks` but I suggest getting comfortable with the keyboard shortcuts.

Set / toggle a bookmark with `⌘` + `F2`. If you are using an Apple keyboard, make sure you also are pressing `FN`. You'll notice that a tiny little chevron shows up in the gutter marking the bookmark.

Windows and Linux users should use `Ctrl` in the place of `⌘`.

You can cycle through your document's bookmarks with `F2`, or reverse the cycle `Shift` + `F2`. Clear all bookmarks with `⌘` + `Shift` + `F2`.

Sublime offers an option to select all bookmarked positions via the menu system. If you would like to create a keyboard shortcut for this, use the command `select_all_bookmarks` and reference the section on creating your own keyboard shortcuts.

Working with Git

If you work on a team or collaborate on any sort of open source project, you probably use git and GitHub for version control. While GitHub offers a GUI application for working with git, many developers still prefer the speed and simplicity of the command line. Using git inside of Sublime Text allows you to have many of the command line benefits without ever having to leave the editor.

I have also found that the menu system and language used in Sublime Text Git is really nice for beginners who aren't entirely sure what all the git commands are - simply typing **git** into the commands palette will give you all available git commands for that current file and repo. This is great way for beginners to learn and git comfortable with git.

Sublime doesn't have git integration out of the box so we need to install a package. There are a number of packages for using git within Sublime Text and the one you pick is really up to your own personal preference. I prefer to use one simply called **git** (https://github.com/kemayo/sublime-text-2-git/) as it allows me to access every single command from the Sublime command palette - this is very much integrated with how you should be using Sublime Text. Later in this chapter we will review a few more packages that you may find helpful when working with git.

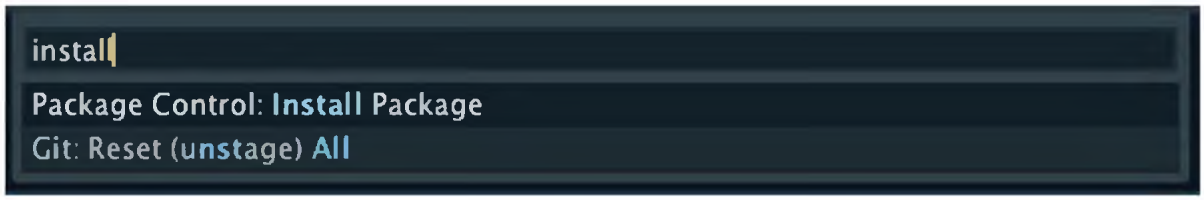
19.1 Sublime + Git Tutorial

Gittin' Ready

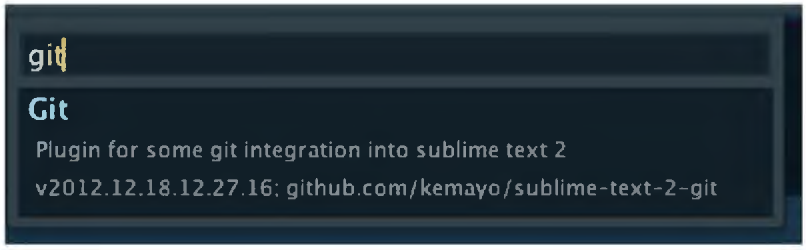
This tutorial assumes you at least have an idea of what Git is and why you would use it for version control. If you are completely new to git, follow along, you will learn how to got from a blank slate to adding files to github in this basic tutorial. If you are already a git power user, you may want to just skim this tutorial to see how everything lines up with your terminal/cmd line experience.

Before we go ahead, make sure you have git installed. If you have never setup git on your computer before, GitHub has a great guide on how to do so.

So, to install Sublime Git, open up package control with `⌘` + `Shift` + `P` and type install package.

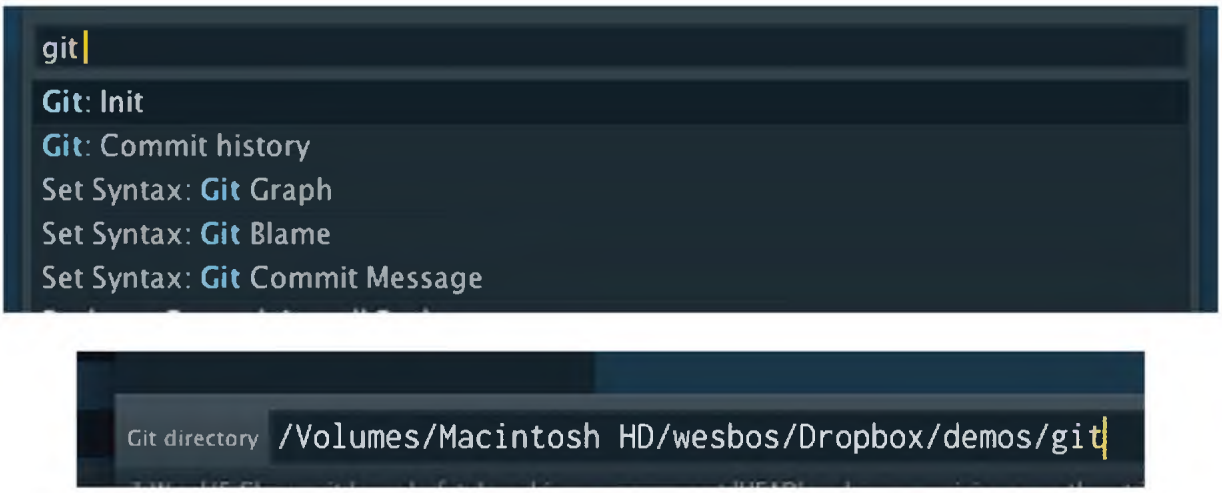


Type **git** and install hit enter. Wait a few seconds and you are good to go!



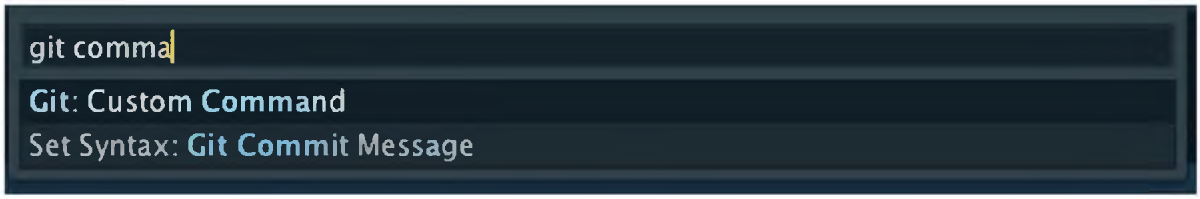
Gittin' Goin'

First thing we need to do is initialize a blank git repository. Simply bring up the command palette (`⌘` + `Shift` + `P`) and type `git init` . You will be prompted to double check the directory path in the bottom of the editor - make sure your are initializing the repo in the folder that you really want it in. We now have an empty git repository on our local computer.

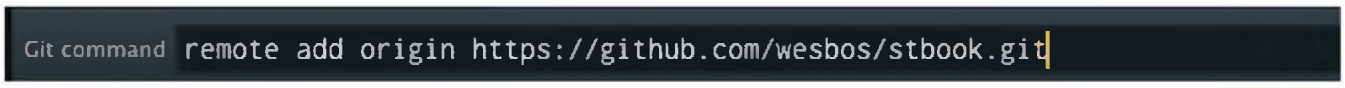


The next step is to link things up with our github account. If you use bitbucket, gitlab or some other remote repo, simply sub this url out for yours.

Now, this package provides commands for most common git use cases, but sometimes you will need to run a more specific git command. One time this is true is when you are adding a remote github repo to your local repo. For this, bring up the command palette and type `git custom`. This will bring up an input in the bottom of your editor.

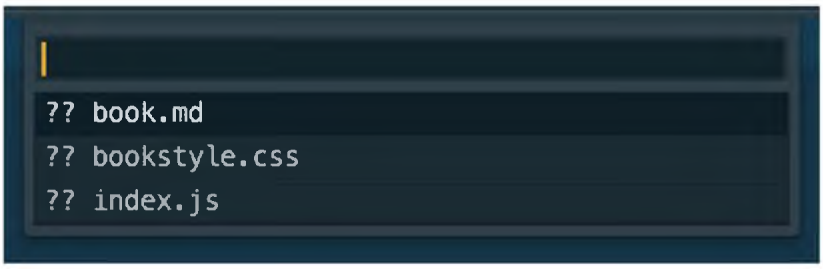


Go grab your github url and type `remote add origin <your github repo url>` into the command palette. Make sure to leave off the **git** part as sublime will run your command as `git [whatever you type]`.



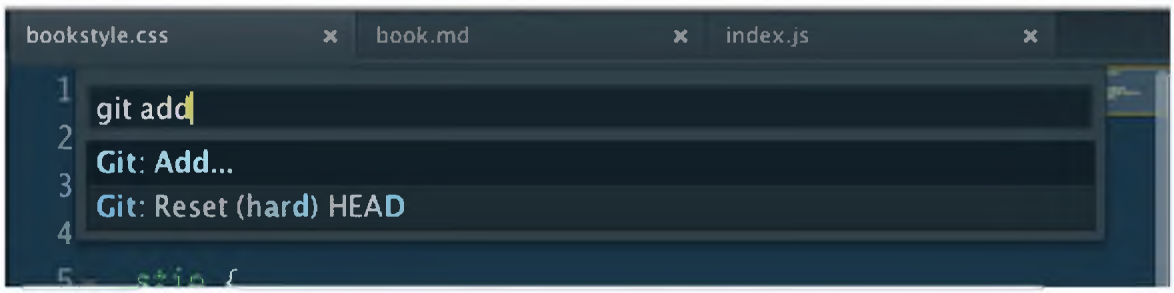
Adds and Commits

At any time you can take a look at the status of your git repo by simply running the `git status` command. Here is what mine looks like after initializing the repo and creating 3 files in the directory:

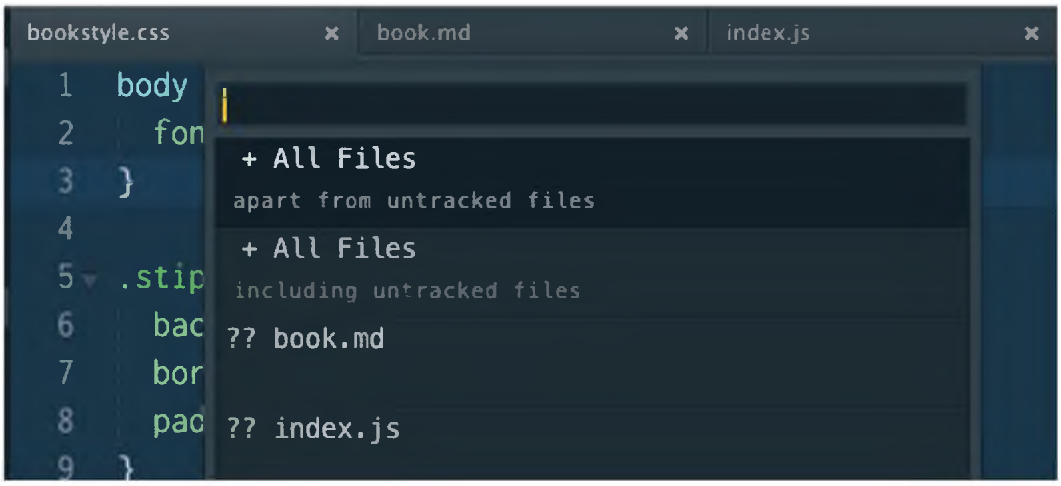


Three untracked files

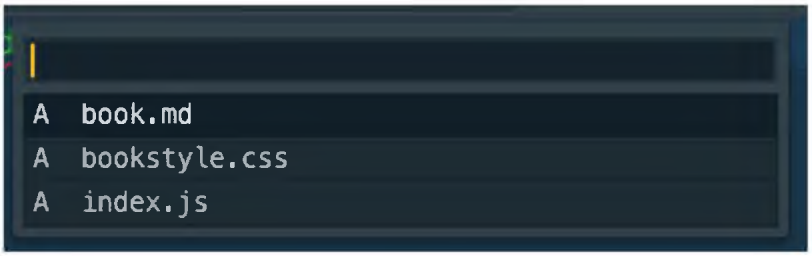
The `??` denotes that those files are in the folder, but are untracked -- they haven't been added to the repo yet. Add files to the repo by bringing up the command palette and typing `git add`



This will bring up a second screen with the option to add tracked files, all files including untracked, or a single file from the git repo. Since we haven't added anything to the repo yet, let's go ahead and `+ All Files including untracked files`

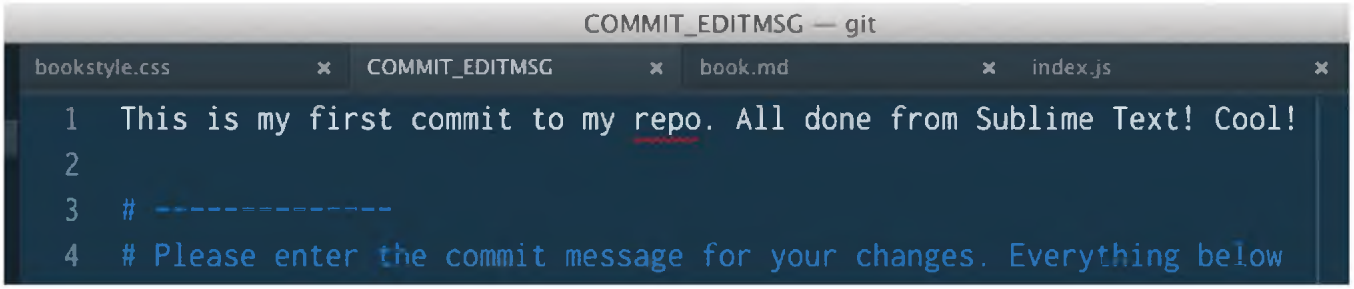


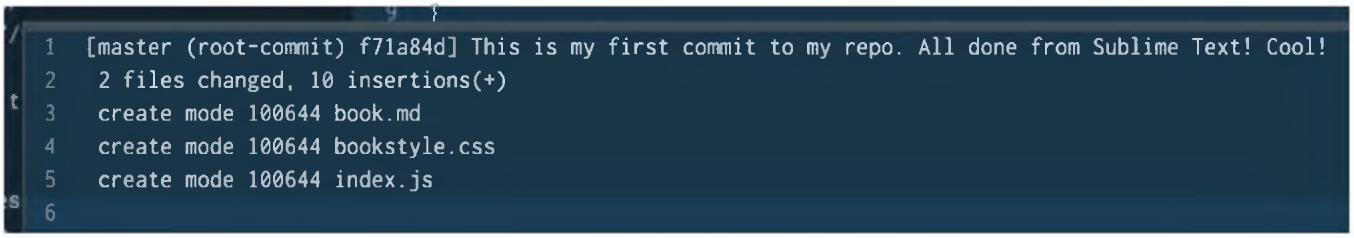
Now when we do a quick `git status` we can see all three files changed from `??` to `A` -- this means they have been **A**dded to the repo.



You are also able to quickly add a single file by bringing up the command palette and running `git add current file`

Next we want to **commit** the files and then **push** them to our github repo. You probably have the hang of this by now - open up the command palette and type git commit. This will bring up a file called **COMMIT_EDITMSG**. Simply just type your commit message at the top of the file and close it (`⌘` + `W`) - no need to save this file.

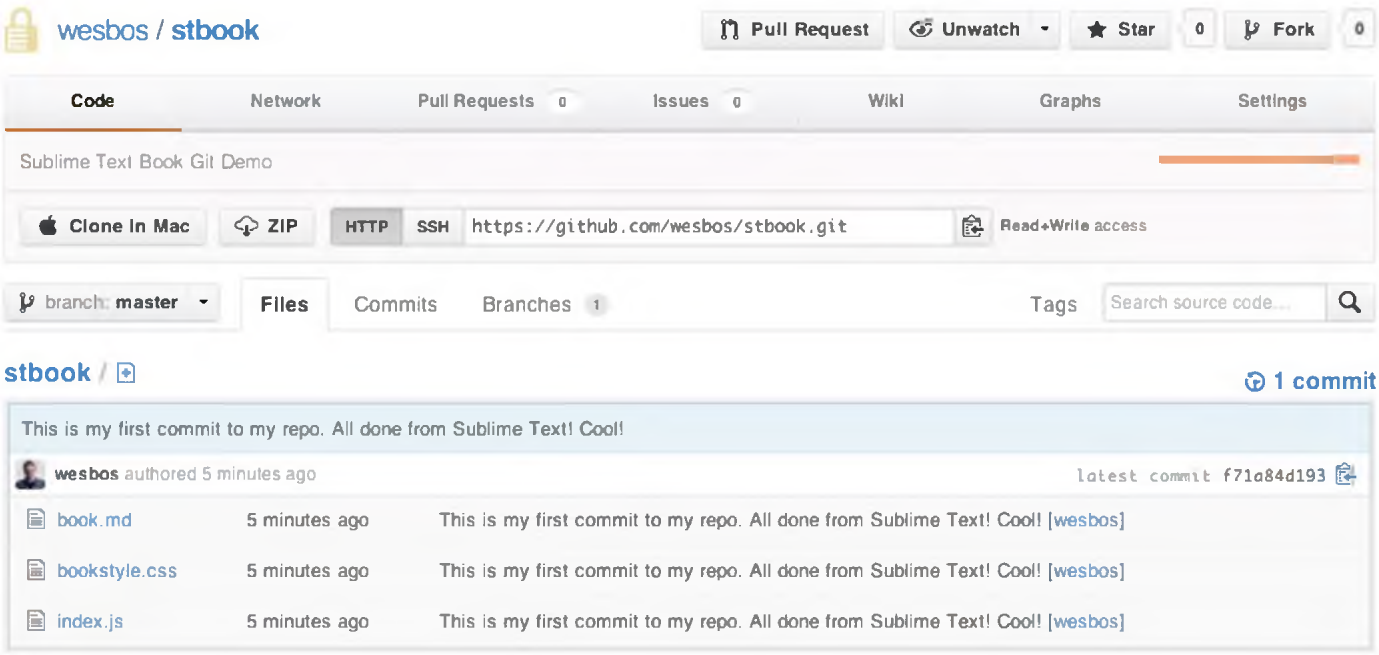




So, we have added our files to the repo and committed those changes - now it's time to push them to the github repo. Open the command palette and type git push. All your currently commited files will be pushed to github and you should receive a success message in the lower bottom pane of sublime text.

```
1  To https://github.com/wesbos/stbook.git
2   * [new branch]      master -> master
3
```

Result of git push current branch

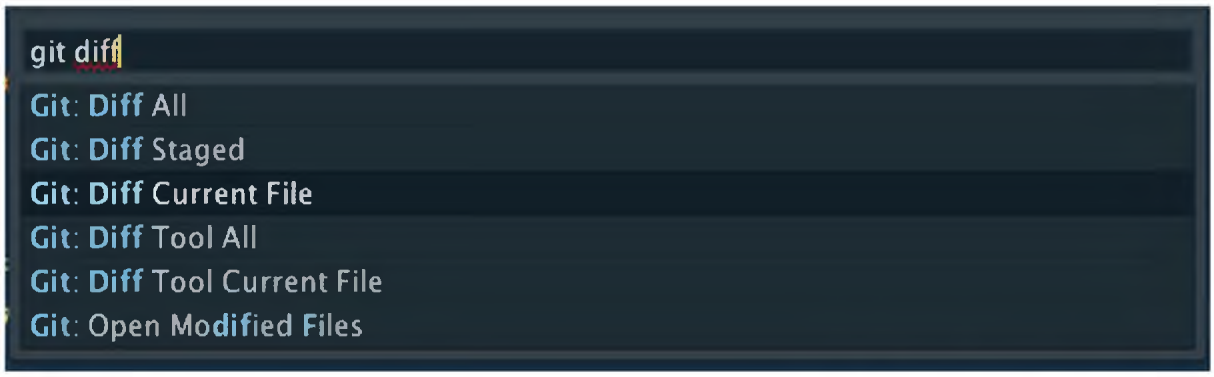


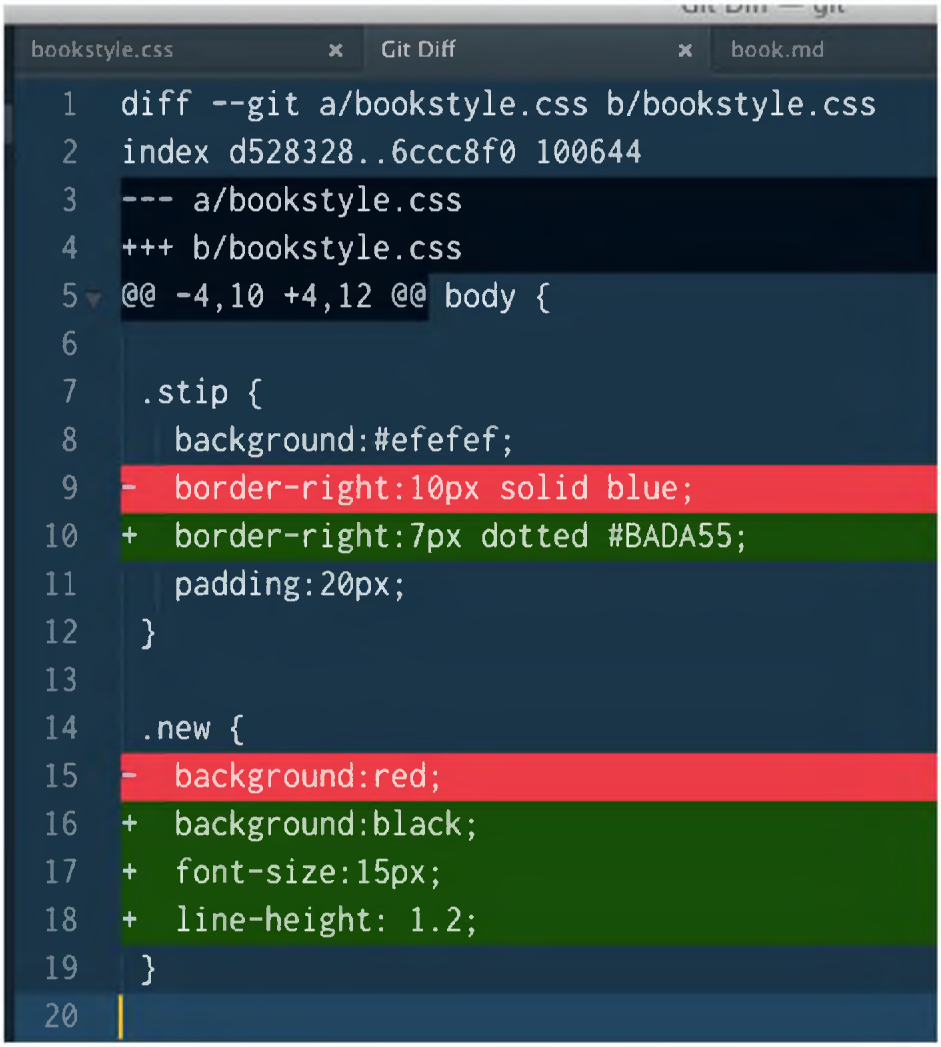
Ta-da! It's on github!

Diffing

One of my favorite parts about having git baked right into sublime text is the ability to diff files right inside the text editor.

I'm going to make a few changes to my `bookstyle.css` file and then run `Git diff current file` when I have it open. Git will compare the file against the one we just added, committed and pushed.





If you have never seen a diff file before, the black (called headers) give you a summary of whats going on in this diff, the green are added lines of code and the red are deleted lines of code.

Gitting everything else

I could write an entire book on how to use git, so for the sake of keeping on track I'm not going to go through every single git command available. Here are a few common commands as well as a few handy Sublime Text nuggets.

Pulling - `git pull` pulls the latests changes from the repo. Pretty simple but crucial when you work on open source or on a team.

Branching - Sublime Git offers a great menu of options for working with branches - you can easily create, switch, pull, push, delete and merge branches without having to remember all those CLI arguments.

Open modified files - Use this to quickly open all files that have been modified. Great for reviewing your code before you add and commit to a repo

Quick Commit - Busy person? Don't have to to add **and** commit your file? Add and commit the current file in just one swoop by choosing **git quick commit** from the command palette

19.2 Additional Git Packages

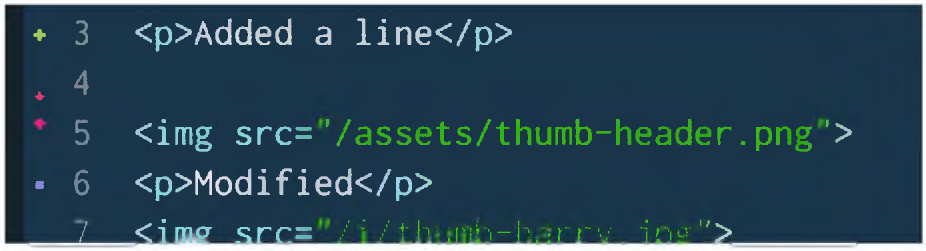
Sidebar Git

There is an alternative package called **SideBar Git** (https://github.com/SublimeText/SideBarGit) which will allow you to manage your files in the sidebar by right clicking files/folders and triggering commands that way. I feel that the command palette in the above plugin is far superior than clicking around - but to each his/her own. This package also does not currently support git tagging.

19.3 GitGutter

When working on project that involve Git, it's helpful to know what lines have be added, edited or deleted.

Git gutter is a simple yet powerful package that will show you just that, with icons in the gutter.



If your gutter icons are just white, you need to make sure you are using a theme that supports GitGutter. The package's readme has a list of supported colour schemes as well as instructions on how to make your theme ready.

Sublimerge

This plugin lets you diff and merge files from a git or svn repo right inside the sublime text GUI. I prefer to use something a little more graphical to do merges, but it looks to work great!

Comparing and merging two files

At it's simplest, Sublimerge is great at comparing two files in your project - no version control involved.

To use, simply select two files from the sidebar, right click and select `Compare selected files`.

This will open the files side and side and highlight the differences.

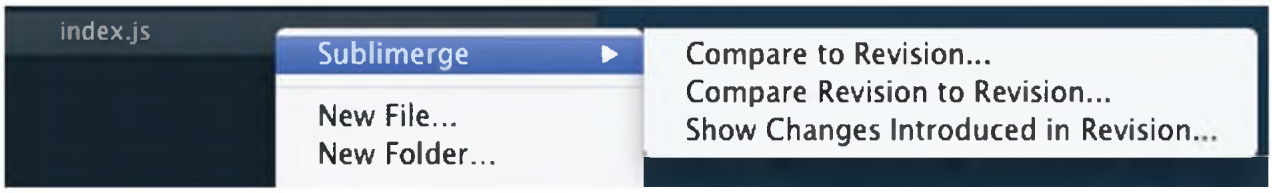
From there, we can step through each change and either move it to the left, right or ignore it.

To start the merge process, right click anywhere in the editor and select `Go to Next Change` or type `control` + `Option` + `=` and then use `Ctrl` + `Option` + `,` /+ `Ctrl` + `Option` + `.` to merge to the right and left accordingly.

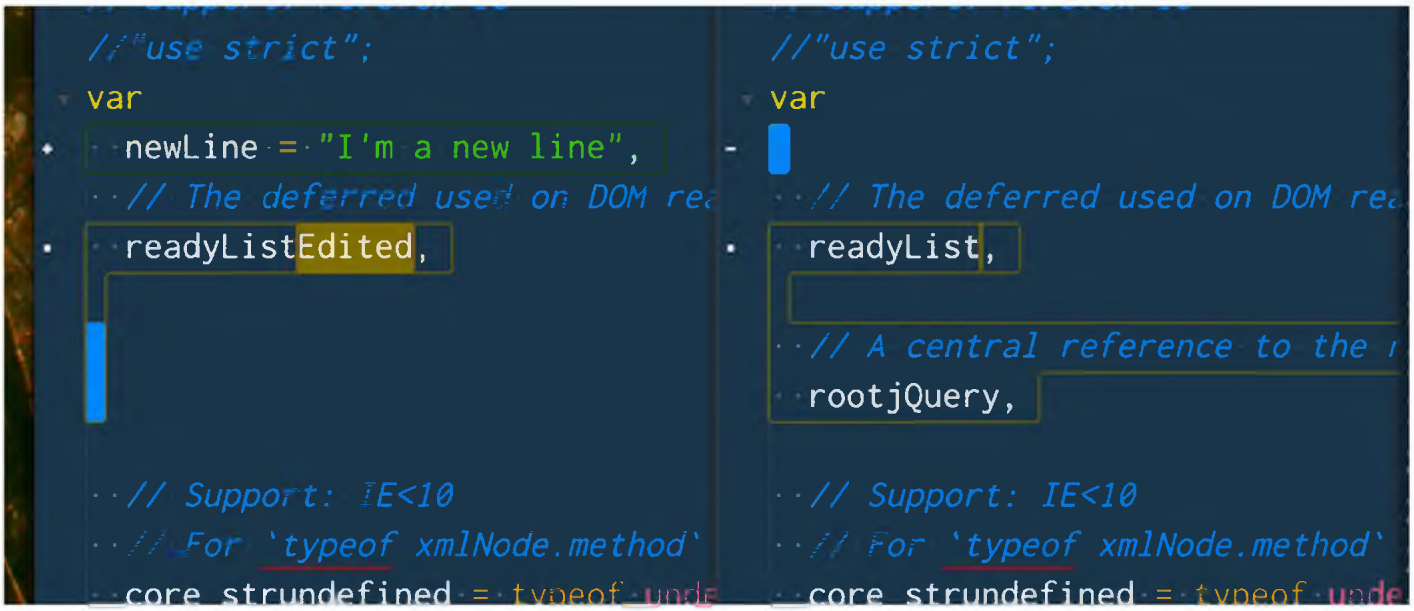
Comparing Git Revisions

Sublimerge is aware of your git repo and is able to compare against different commits.

To use this, open your git repo in Sublime and select a file from the sidebar with a right click and you will see three options:



1. **Compare To Revision** – This will compare the current file to any previous git commit. When you select this, you will be asked to choose a previous commit to compare it to.
2. **Compare Revision to Revision** – Similar to above, but rather than comparing the current file to a commit, you will be asked to select two commits to compare against each other.
3. **Show Changes introduced in Revision** - select any revision and view what new changes were introduce with that commit.



Git, SVN and Mercurial Integration

For deeper integration into your version control system, check out the docs for Sublimerge available at http://www.sublimerge.com/docs/vcs-integration.html.

Mastering Emmet

Emmet is a package for Sublime Text that helps with writing of CSS and HTML. To say that it speeds you up is an understatement, you would be silly to code HTML or CSS without this package installed.

If you are already familiar with Zen Coding don't skip this chapter just yet, Emmet is its successor that does so much more than the original Zen Coding. While you can get the Emmet package for pretty much any editor, we are going to cover it extensively as is so well aligned with the philosophy of Sublime Text.

20.1 Emmet and HTML

Emmet is most famous for it's HTML shortcuts. If you write Jade, Slim or HAML, everything here translates 100% with the latest version of Emmet.

With almost everything in Sublime Text, you type your shortcut and then hit the `tab` key.

Elements

Start off simple, type the element name you want and hit tab. Notice how Emmet puts your cursor right inside the tags? Handy!

`p + tab` → `<p>|</p>`

`span + tab` → `<span>|</span>`

`img + tab` → `<img src="|" alt="">`

Classes and IDs

Then we can step it up and add classes.

`span.warning` → `<span class="warning"></span>`

If you leave out the element before the class, it assumes a div:

`.wrapper` → `<div class="wrapper"></div>`

Notice how these are just CSS selectors + tab?

`ul.nav` → `<ul class="nav"></ul>`

Works with IDs too:

`h2#post23` → `<h2 id="post23"></h2>`

You can also do both at the same time:

`.post#post23` → `<div class="post" id="post23"></div>`

Attributes

Just like CSS, you can also use square brackets to denote element attributes

`input[placeholder="username"]` → `<input type="text" placeholder="username">`

There is no limit to how many attributes you can set at once:

`img[src="dog.jpg"][alt="Cute puppy"]` → `<img src="dog.jpg" alt="Cute puppy">`

You can even mix and match with classes/IDs:

`label.name[for="first"]` → `<label for="first" class="name"></label>`

Text

New to Emmet is the ability to insert text inside the elements:

`p{hello world}` → `<p>hello world</p>`

As always this can be used with any of the existing things we have learned:

`span.warning{Watch out man!}` → `<span class="warning">Watch out man!</span>`

Multiple Elements and $ placeholder

We have all needed to quickly generate a bunch of elements before. Rather than copy/paste, we can use the asterisk to multiply the number of elements:

`li*3` →

```
<li></li>
<li></li>
<li></li>
```

As always, everything we learned before can be applied. Just make sure to put the `*n` at the end of your recipe:

`label.display{Yeah!}*2` → `<label for="" class="display">Yeah!</label><label for="" class="display">Yeah!</label>`

A common use case for this is when your create a number of elements, each with a class of `box`, but they also need a unique class such as `box1`, `box2`, etc..

This is where the amazing `$` increment placeholder comes in. Just use $ where you want 1/2/3 to be, and Emmet will replace it for you.

`.box.box$*5` →

```
<div class="box box1"></div>
<div class="box box2"></div>
<div class="box box3"></div>
```

```
<div class="box box4"></div>
<div class="box box5"></div>
```

Awesome! The dollar sign works anywhere in an Emmet string.

`li{Item $}*3` →

```
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
```

If you uses more than one `$`, a leading zero will be placed for numbers under that many digits.

`img[src="dog$$.jpg"]*10` →

```
<img src="dog01.jpg" alt="">
<img src="dog02.jpg" alt="">
...
<img src="dog09.jpg" alt="">
<img src="dog10.jpg" alt="">
```

You can start the numbering at certain number:

`li.item$@5*3` →

```
<li class="item5"></li>
<li class="item6"></li>
<li class="item7"></li>
```

And even count backwards!

`p.countdown$@-*10` →

```
<p class="countdown10"></p>
<p class="countdown9"></p>
...
<p class="countdown2"></p>
<p class="countdown1"></p>
```

Nesting Elements

Finally, one of the most powerful features in Emmet is to nest elements.

We can use the `>` to specify children:

`div.wrapper>p.warning{Watch Out}` →

```
<div class="wrapper">
    <p class="warning">Watch Out</p>
</div>
```

And yes it works with everything else!

`ul>li.book${Book $}*5` →

```
<ul>
    <li class="book1">Book 1</li>
    <li class="book2">Book 2</li>
    <li class="book3">Book 3</li>
    <li class="book4">Book 4</li>
    <li class="book5">Book 5</li>
</ul>
```

We can do siblings with the `+`

`header+section.post+footer.bottom` →

```
<header></header>
<section class="post"></section>
<footer class="bottom"></footer>
```

Emmet Filters

A lesser-known part of Emmet is the ability to pipe any Emmet selector into a filter.

Closing Element Comments

For example, many developers like to add comments to the end of their elements so there know when they start and end. For this we simply append `|c` to the end of the selector like so:

`.wrapper>.inner|c`

And that gives us:

```
<div class="wrapper">
    <div class="inner"></div>
    <!-- /.inner -->
</div>
<!-- /.wrapper -->
```

Escaping HTML

Pipe to `e` to have escaped HTML returned:

`span.hello|e` → `<span class="hello"></span>`

Pipe to HAML or Jade

When you are in a HAML or Jade file, this is the default functionality, however if you wish to get HAML or Jade in another HTML file - pipe in `|haml` or `|jade`

`ul.drinks>li{Drink $}*3|jade` →

```
ul.drinks
    li Drink 1
    li Drink 2
    li Drink 3
```

Expand in a single Line

Emmet does a great job of putting block elements on their own line and inline ones beside each other. If, for whatever reason, you wish to put everything on a single line, pipe `|s`.

`.wrapper>p.warning|S` → `<div class="wrapper"><p class="warning"></p></div>`

Even More

There are handful more HTML helpers including grouping, climbing, inputs and document setup. I recommended you check the Emmet cheat sheet http://docs.Emmet.io/cheat-sheet/

20.2 Emmet and CSS

The other major half of Emmet is useful when writing CSS (or Sass, LESS or Stylus). Many think that Emmet is just a huge snippets library, when in reality it is much more than that. The power of Emmet with CSS comes from the *fuzz searching*, which means that we can just type a few characters of what we want, and Emmet will make a best guess for what we wanted.

For example, if we want to type `overflow:hidden;` , just type `oh` , `o-h` , `ovhi` , `ovh` or any number of combinations + `tab` and it will be replace with `overflow:hidden;` .

This is huge because we no longer have to memorize or guess what the snippet was, we can just type as much as we think will get us there and 95% of the time, it works out!

My preferred way of writing Emmet snippets is just to type the first 1-3 letters of both the property and value.

`posrel` → `position: relative;`

`posab` → `position: absolute;`

`fl` → `float:`

`fll` → `float: left;`

`fr` → `float: right;`

`db` → `display: block;`

`dib` → `display: inline-block;`

`tdn` → `text-decoration: none;`

`brad` → `border-radius:`

`c` → `color: #`

`w` → `width:`

`p` → `padding:`

You can also put a dash `-` in between the property/value. This is helpful when things aren't going the way you want.

For example if you want `position:absoute;`, you might try `pa`. However, that completes to `padding:`. Putting a dash between each letter will fix this: `p-a`

Emmet also allows the use of a colon `:` in place of the above mentioned dash. However, it doesn't play nice with Sublime Text because sublime text auto completes the semicolon `;` when you type a colon in CSS. So, best to just avoid using the colon and stick with dashes or nothing at all.

Numbers and Units

So, Emmet works great for property/value key pairs that are words, but what about number units?

We already know that `w` and `h` give us `width:` and `height:`, so let's step it up a notch.

Adding a value after any Emmet snippet will set it in pixels:

`w100` → `width: 100px;`

`h200` → `height: 200px;`

`p10` → `padding: 10px;`

`mr3` → `margin-right: 3px;`

adding a `p` or `%` after the value will turn them to percentages:

`w100p` → `width: 100%;`

`h200p` → `height: 200%;`

`p10p` → `padding: 10%;`

`mr3p` → `margin-right: 3%;`

Similiarily `e` will set ems and `r` will set rems:

`w100e` → `width: 100em;`

`h200r` → `height: 200rem;`

`p10e` → `padding: 10em;`

`mr3r` → `margin-right: 3rem;`

We can set multiple sides at once with dashes:

`p10-20-5-50` → `padding: 10px 20px 5px 50px;`

And use `i` for inherit and `a` for auto.

`m0-a-i-a` → `margin: 0 auto inherit auto;`

It knows about borders too!

`b1-s-#000` → `bottom: 1px solid #000;`

`bbw2r` → `border-bottom-width: 2rem;`

Colors

Colors in Emmet are pretty standard:

`c` → `color:#000;` `c:r` → `color:rgb(0, 0, 0);` `c:ra` → `color:rgba(0, 0, 0, .5);`

As of Emmet 1.1, you can also covert hexcodes to RGBA values.

`c#badda55.3` → `color: rgba(186, 221, 165, 0.3);`

`bg#1d1.45` → `background: rgba(17, 221, 17, 0.45);`

`bdc#fff.5` → `border-color: rgba(255, 255, 255, 0.5);`

More CSS

There are almost a thousand possible snippets you could expand to with Emmet, most of which you can figure out by just typing the first few letters.

For more, make sure to visit the cheat sheet at http://docs.Emmet.io/cheat-sheet/

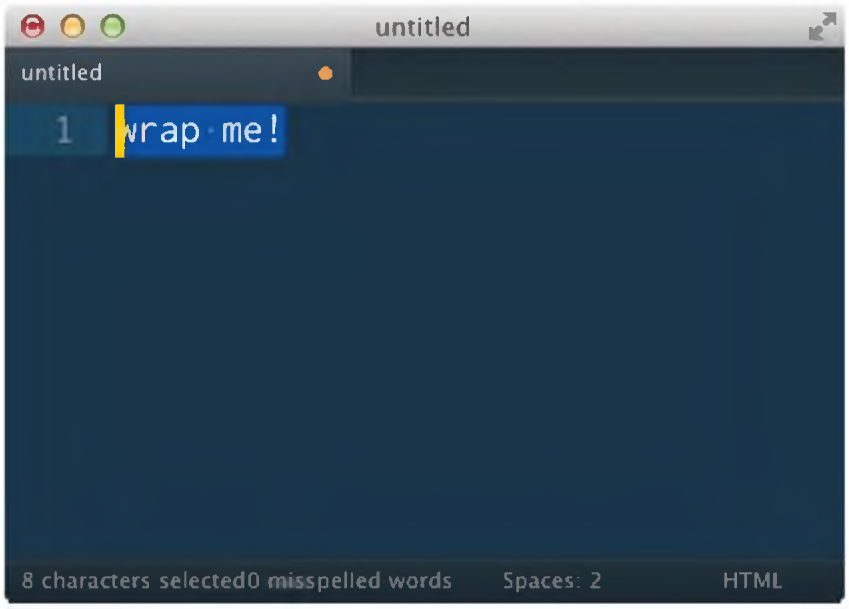
20.3 Other Emmet Hot Tips

The third part of Emmet is a handful of workflow/utility functions that it builds into Sublime Text.

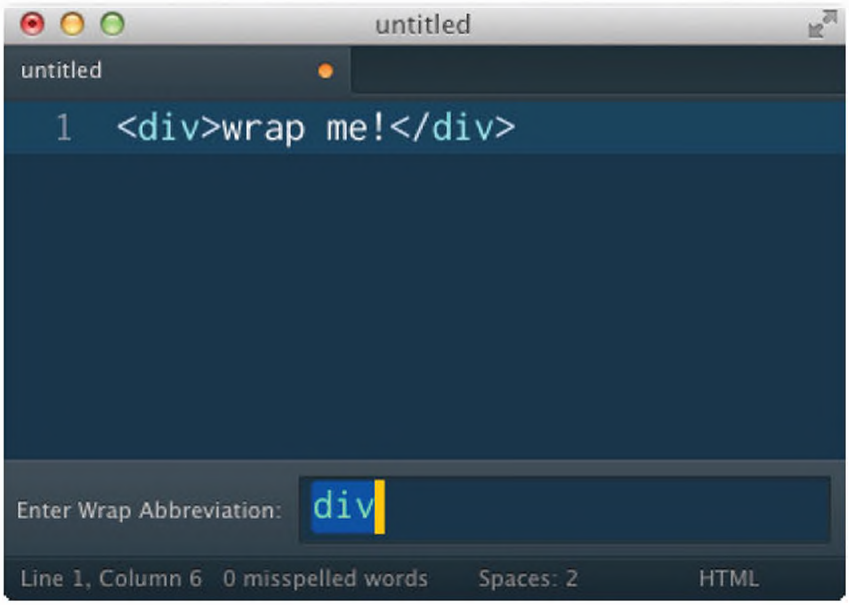
Wrap with Emmet Snippet

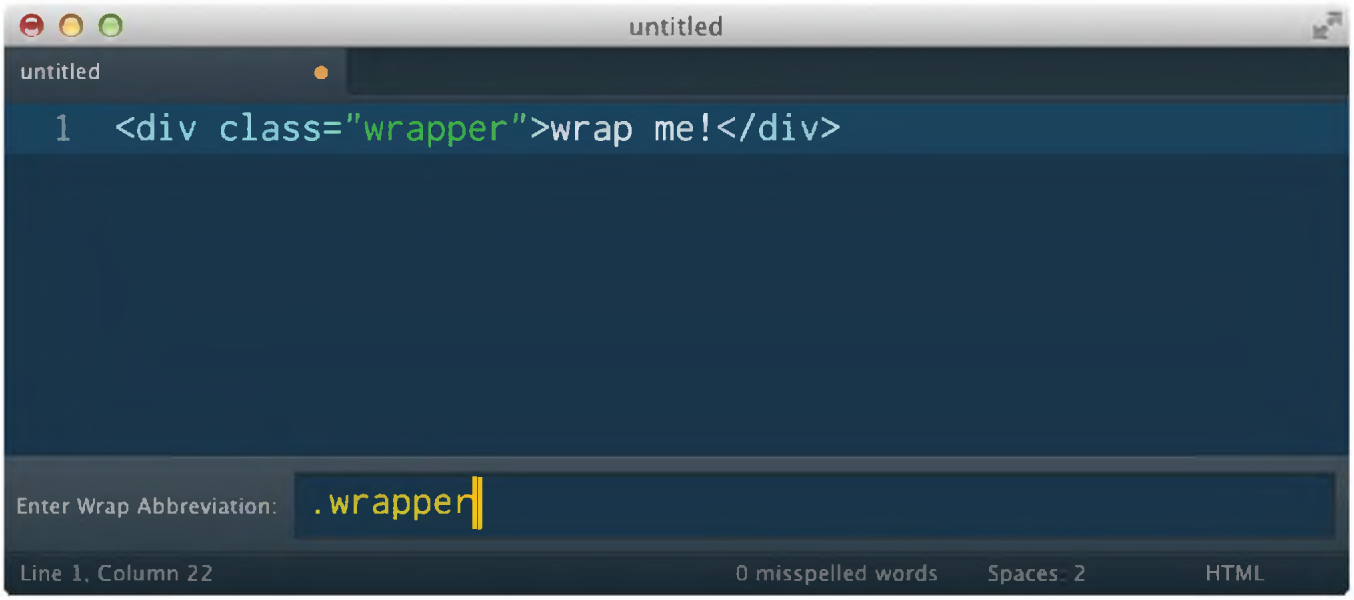
While Sublime text already has a *wrap selection with tag*, it's not nearly as nice or flexible as Emmet. Sublime text only allows to you specify the tag name - no classes/ids/attributes/nesting or any of the great things Emmet has to offer.

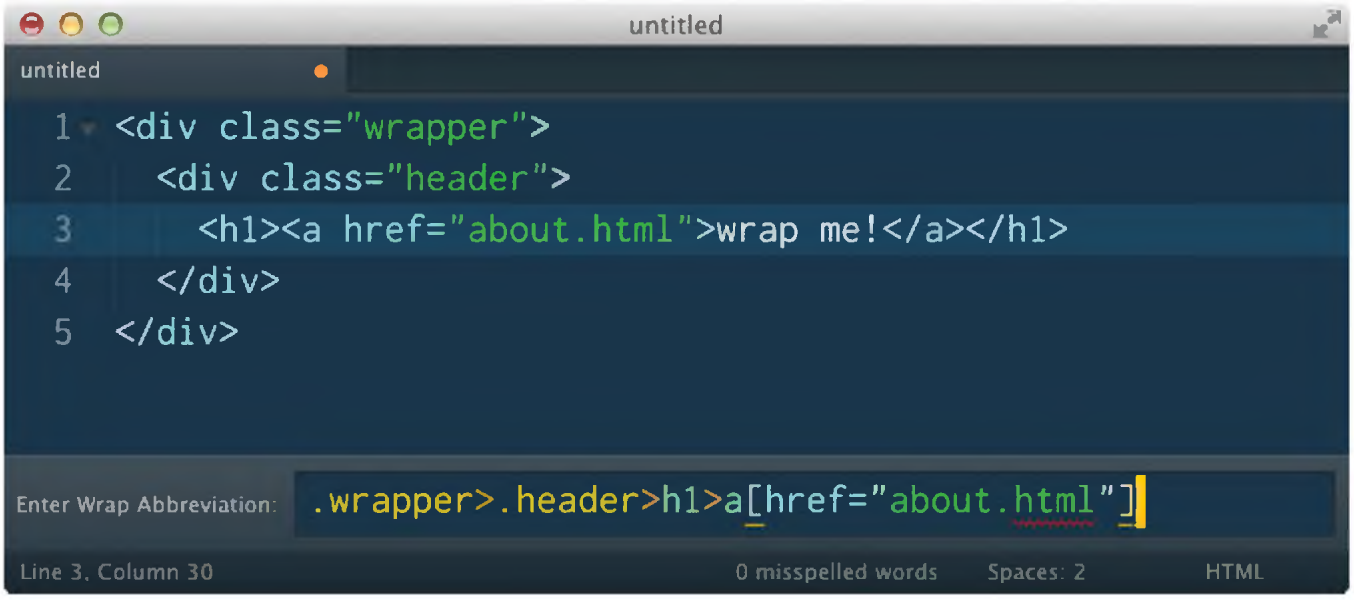
To use this, you simply need to be in any HTML/Slim/Jade/HAML document, select the text you want and hit `Ctrl` + `w` or select **Emmet: Wrap With Abbreviation** from the command palette.



This will open an input box at the bottom of the editor, where you can type any previously learned Emmet snippet. The default is a div, but you can put *anything* in here.


untitled
untitled
1 wrap me!
Enter Wrap Abbreviation:
div
Line 1, Column 6 0 misspelled words Spaces: 2 HTML


untitled
untitled
1 wrap me!
Enter Wrap Abbreviation:
.wrapper
Line 1, Column 22 0 misspelled words Spaces: 2 HTML


untitled
untitled
1
2
3 <h1>wrap me!</h1>
4
5
Enter Wrap Abbreviation:
.wrapper>.header>h1>a[href="about.html"]
Line 3, Column 30 0 misspelled words Spaces: 2 HTML

When finished, hit enter to return to editing you code!

Encoding / Decoding Data URI

Data URIs are a way to convert binary files (usually PNGs and JPGs) into a single string of text. This allows you to include images into your HTML or CSS without having any external files.

We can use them in both HTML and CSS.

```
<img src="dot.png">
```

becomes

```
<img src="
```

and in CSS

```
    li {
        background-image:url('check.png');
    }
```

becomes

```
li {
    background-image:url('
58ZDrAz3D/McH8yw83NDDeNGe4Ug9C9zwz3gVLMDA/A6P9/AFGGFyjOXZtQAAAAAElFTkSuQmCC');
}
```

I won't get into all about DataURIs right now, but for more info, read up on the Data URI Wikipedia Article.

The cool thing about Emmet is that rather than using some generator the convert them, you can do it right from the editor itself! When in either CSS or HTML, put your cursor anywhere in the file path and hit `Ctrl` + `Shift` + `D` or select `Emmet: Encode/Decode Image to data:URI` from the command palette.

Data URIs are usually *huge*, so if you are using them, I recommend turning line wrapping off.

Increment/Decrement

When writing CSS, you often need to bump values up or down by 0.1/1/10. Rather than backspacing your current value and re-writing the new one, you can use keyboard shortcuts to increment/decrement them!

For example, given this CSS. I want to increment each one 3px, 0.3rem and 30% accordingly. You are probably used to this functionality from Chrome/Firefox developer tools.

```
.wrapper {
    padding:10px;
    font-size:1.3rem;
    width:50%;
}
```

Simply put your cursor anywhere within the number and use:

`Ctrl` + `↕` → up/down by `1`

`Alt` + `↕` → up/down by `0.1`

`Alt` + `⌘` + `↕` → up/down by `10`

Lorem Ipsum

When building a proof of concept app/web page, placeholder text is something that can really slow you down. Thankfully Emmet can solve all of this for you.

To get placeholder text, simply type `lorem` and hit tab.

`lorem` → `Lorem ipsum dolor sit amet, consectetur adipisicing elit. Repellendus voluptatibus voluptas, ipsa quod officia quaerat at possimus veritatis eveniet nemo dolores fuga quibusdam explicabo. Hic quae facere eius aut consectetur.`

Only want a bit, or a lot? Append the number of words you wish to have at the end of the word lorem:

`lorem5` → `Lorem ipsum dolor sit amet.`

You can also use this with Emmet snippets:

`p>lorem` → `<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Expedita inventore, error odit beatae, eveniet ratione modi ipsum nulla ex obcaecati cupiditate quam quae nam consequuntur tenetur fugiat harum, dolore repudiandae.</p>`

even multiples!

`ul>li*3>lorem4` →

```
<ul>
    <li>Lorem ipsum dolor sit.</li>
    <li>Totam tenetur dicta incidunt.</li>
    <li>Blanditiis illo laborum reiciendis.</li>
</ul>
```

Matching Pair

Long nested HTML? Sometimes it is hard to find which closing element belongs to what. With matching pair you can jump to the associated open/close tag.

Open this via the command palette or use the `Ctrl` + `Shift` + `T` shortcut.

20.4 Other Emmet Treats

Be sure to reference the documentation at http://docs.Emmet.io/ and the cheat sheet http://docs.Emmet.io/cheat-sheet/

Workflow & Code Quality

As a developer, how you work should be very important to you. By buying this book, you are obviously very aware that your workflow can really impact the speed at which you work as well as the quality of code you end up producing.

This section will cover various tools that you can use to both speed up how you work as well as maintain and improve code quality.

21.1 Live reload

Live reload is an package that communicates with your browser to reload your page whenever a change to your website is make. It is one of my absolutely favorite tools in my web development workflow because it speeds things up so quickly.

You may not realize it, but moving away from your editor to refresh your browser can take a few seconds, and if you are doing this a few times a minute (and even more if you are working on tweaking your CSS) it can quickly add up. Saving a few seconds, 3 times a minute can add up to well over 3 hours over the course of your week!

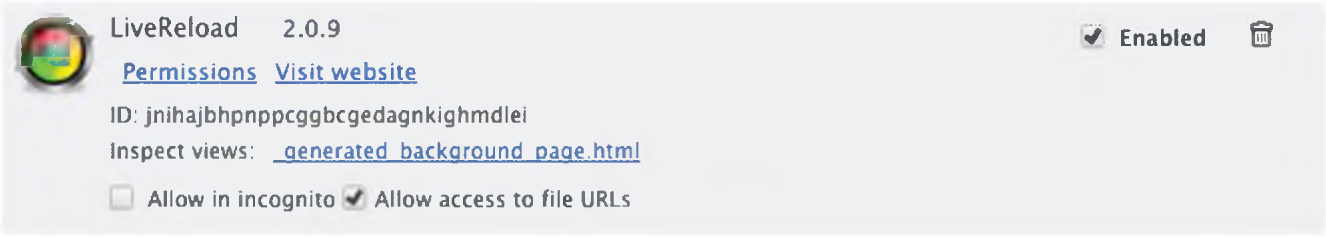
Installing

There are two steps to install live reload. The first is the Sublime Text package. As we always do, go ahead and search for Live Reload in package control. Go ahead and install it. When done, Sublime will need a full restart.

Search for Live Reload

The second step is to get Sublime working with your web browser. Live Reload uses web sockets to communicate to the browser when changes are made. If you are using Chrome, Safari or Firefox on the desktop, there are great packages that get everything running for you. Visit Live Reload website for links on each of those.

If you are using the chrome plugin, make sure you check the box that says "Allow access to file URLs" otherwise you will be limited to local servers.



Live reload on mobile devices

If you are running on another browser that doesn't support extensions, like a mobile browser, you can still use live reload by placing the following JavaScript snippet into your web page or application:

```
<script src="http://localhost:35729/livereload.js?snipver=1"></script>
```

To break that down, `localhost` is the local name for the server running on your computer and `35729` is the port which live reload runs on.

If you are running live reload on your computer but testing on another computer or mobile device, you will want to switch `localhost` with the IP address or local name of your computer. My computer is named `w`, has the IP address of `192.168.124.112` and I have a local domain of `test.com` setup on my router (more info here) so any of the following will work as long as the devices are on the same WIFI network.

```
<script src="http://w:35729/livereload.js?snipver=1"></script>
<script src="http://192.168.124.112:35729/livereload.js?snipver=1"></script>
<script src="http://test.com:35729/livereload.js?snipver=1"></script>
```

21.2 Sublime Server

If you are a front end developer working with HTML, CSS and JavaScript, chances are that you working locally. Being able to spin up a local server quickly is an extremely helpful. There are no databases or programming langauges involved here, simply just serving up static files as you would with Python's simpleHTTPserver or a LAMP install.

Sublime Server is a package for Sublime Text that allows you start a localhost server in a matter of seconds. The really nice thing about this package is that it will serve up any directory or project you currently have open in sublime text, you aren't limited to a single parent directory – go ahead and add directories from anywhere on your computer.

It also replaces the native "Open in browser" functionality by opening that file on your locahost instead of the file system. This is key for when you are working with JavaScript AJAX requests that must be running in a server environment.

Installation and Usage

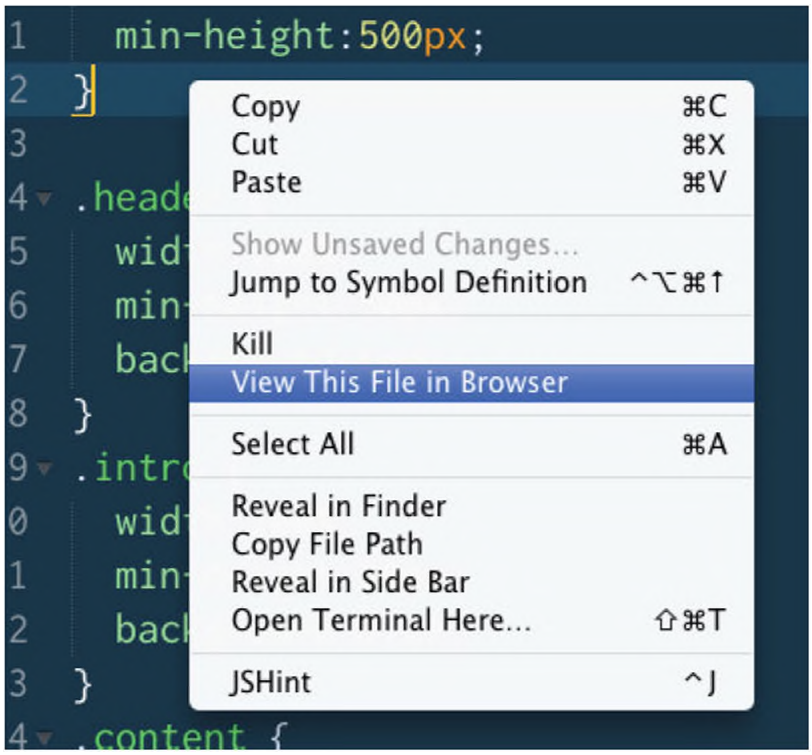
To install, open your command palette and type "install package". When the list comes up, search for "Sublime Server" and hit enter. That is it, as simple as it gets.

Usage of Sublime Server is also pretty straight forward. Make sure you have a project or a few folders open in Sublime and then go ahead and type "start" into your command palette giving you the option to start the server.

Once you have done that, nothing will happen but the server will be running in the background. By default the server runs on port 8080 which means that you can view all your files by going to **http://localhost:8080/** in the browser.

If you are already running something on port 8080 you can change this value and a few others in the settings available at `tools` → `SublimeServer` → `Settings...`

You also have the option of viewing your files straight from Sublime Text. To do this, right click any file and select "View This File in the Browser". This will open your file in your preferred browser under the localhost server.



If you add folders to Sublime that you want served up, simply use the **Restart SublimeServer** command. Once you are finished developing you can use the **Stop SublimeServer** command. It's recommended that you turn this off when on a public network because anyone with your IP address would be able to view your served up files.

21.3 Live Linting with SublimeLinter

In the programming world, each language has a tool called a linter that analyses your code, checking and flagging for potential errors, bad habits and consistency mistakes.

A linting tool goes a long way in improving your coding style and a live linter is like a slap on the hand every time you something as little as forget a semi colon or use a restricted name for a method. A linting tool also lets you know when your code is broken, saving you a trip to the browser/compiler/server to realize that you made a mistake.

The holy grail of linting packages for Sublime is called SublimeLinter. It provides linting for 16 langauges and growing, this includes C/C++, CoffeeScript, CSS, Git Commit Messages, Haml, HTML, Java, JavaScript, Lua, Objective-J, Perl, PHP, Puppet, Python, Ruby and XML.

Refer to the package installation section of this book to install SublimeLinter. For a few languages, the package kicks off the linting task to the system installation of that language so if you are trying to lint PHP, make sure you have php installed. If you are trying to lint ruby, make sure you have ruby installed. You get the point :)

Additionally, if you are a JavasScript developer, you must also have Node.js installed. If you do not, head on over to http://nodejs.org/ and download one of the easy installers for mac, windows or linux.

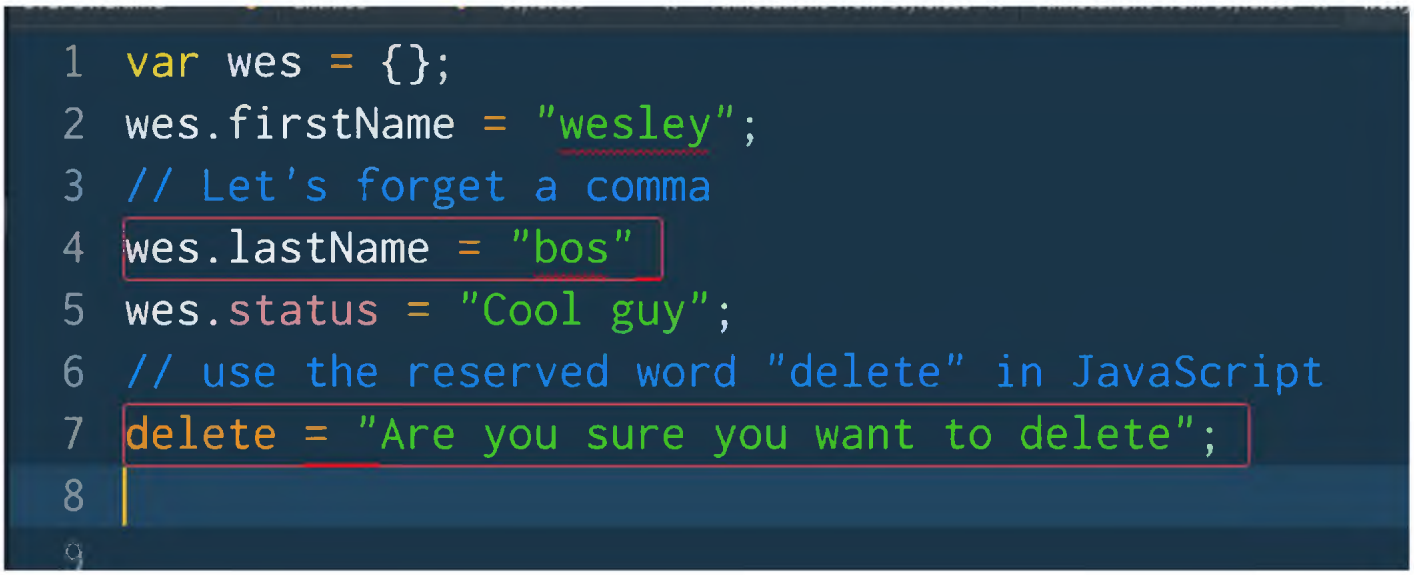
Linting your code

Since most developers reading this book will touch on JavaScript at some point, I will be using JavaScript as an example here, but the process for every langauge is the same.

By default the linting kicks in after 2 seconds or on file save. If there is an error in your code, you will automatically see lines surround the code that is affected. What color something shows up as depends on if your theme supports Sublime Linter or not. My Cobalt2 theme supports it so you will see warnings in orange and illegal statements in red. Other themes wills simply show you white lines around the effected code.

JavaScript Linting Example

Here is an example of some JavaScript I wrote. At first glance, it may seem harmless and will even run just fine. The linter is throwing flags on two items as they may cause trouble down the road in our development.



CSS Linting Example

Here is another example in CSS. While most haven't heard of linting CSS, it's a great practice to get out of bad habits. In this example, I have "using ids" set to warning so I'm seeing orange while using the color 'weswhite' is illegal red because that is not a real color.

```
1 footer#bottom { /* Warning dont use IDs */
2   color:red;
3 }
4
5 .wrapper {
6   color:weswhite; /* Illegal color */
7 }
```

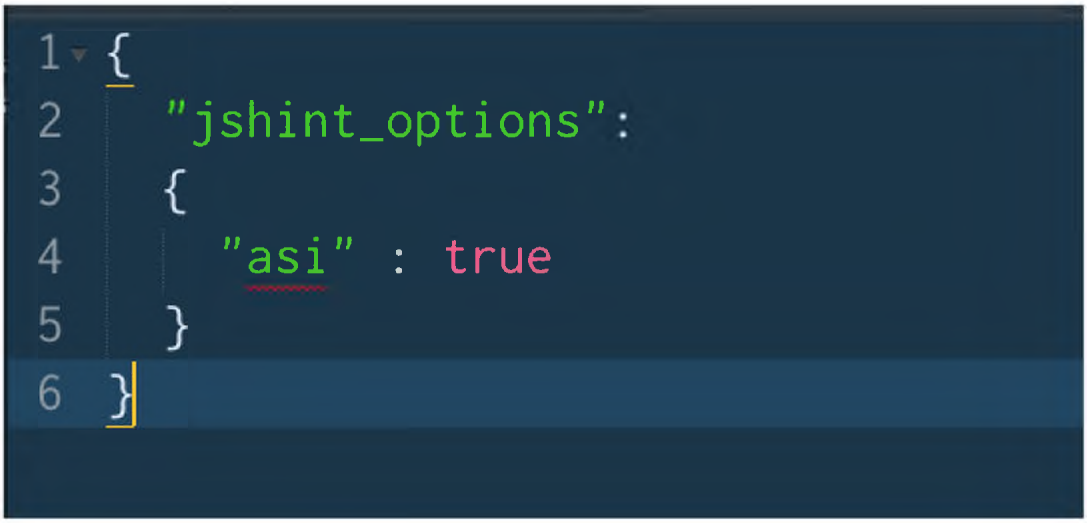
Linting Settings

Every developer has their own style and they might not agree with all the rules that are set by default in the linter settings. That is totally okay and is why linters have the ability to turn on and off different rules. A big part of the linter is to make sure that you are *consistent* with you coding style.

All of your **user specific** linting settings should go to into a file located at `Packages/User/SublimeLinter.sublime-settings`. If you do not have that file, go ahead and create it manually or by using the menu `Preferences` → `Package Settings` → `SublimeLinter` → `Settings - User`.

A reference of all the available settings can be found at `Packages/SublimeLinter/SublimeLinter.sublime-settings`. Event more options can be found by referencing that langauge's linting documentation. Make sure to not edit these in place as they will be overwritten next time the package is setup. Instead copy the settings out and change them in our file inside the `User` directory.

For example, if I was a total boss that knew exactly how JavaScript automatic semicolon insertion **(ASI)** worked, I could opt to turn off that rule in my own settings document like so:



21.4 Working with FTP / SFTP

Like it or not, FTP is still a part of many developer's workflow. While I encourage you to embrace git version control and deployment systems such as grunt or capistrano, there are still times when SFTP/FTP is needed. For this, there are two good solutions which we will now dive into.

SFTP Package

The SFTP package by Will Bond is a fantastic cross platform, stand alone solution. The trial is fully featured and untimed, continued use runs $16 for the package and it is worth every penny. Will Bond is the developer behind Sublime Package Control so I feel $16 is a drop in the bucket considering all he has done for the Sublime Text community.

The SFTP supports FTP, FTPS and SFTP servers as well as SSH key authentication. As an added bonus, the package can detect changes via Git, Mercurial and SVN. As with all things in Sublime Text, if you are used a nice graphical interface, this will take a little getting used to.

To get started, install the package via package control, just as you would any other package. We then have two options: map a local directory to a remote server or to simply just tap into a remote server.

At the time of writing, the SFTP package is currently only supported in Sublime Text 2 although the author has noted intent to port it over to Sublime Text 3.

Remote only server

Again, I do not recommend simply FTPing into a remote server as that is a poor workflow, but there are situations where this will happen.

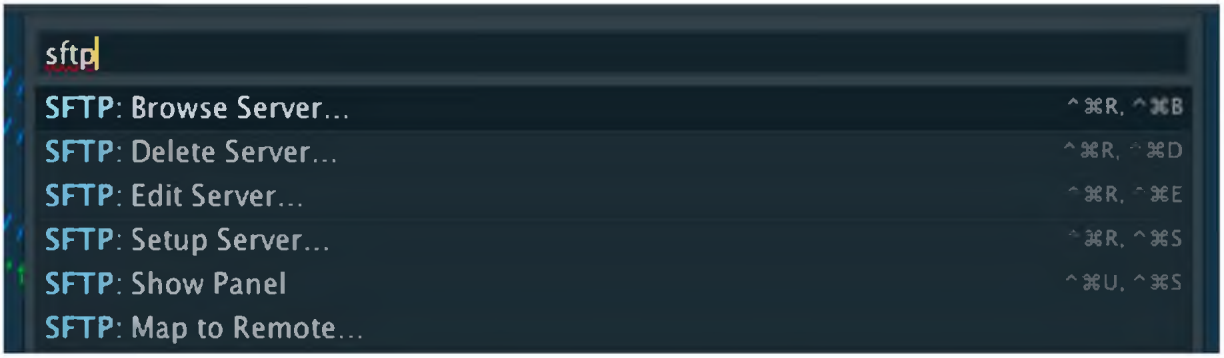
Open up a blank window of Sublime and Choose `File` → `SFTP/FTP` → `Setup Server`. This opens up the screen for configuring your server which in true Sublime Text style, is just a JSON file. Uncomment the options that you require and change the others - this is the same as any other FTP program.

This package comes with good defaults so if you are using a common hosting provider, chances are you can leave most of them as-is only changing `type`, `host`, `user` and either `password` or `ssh_key_file`.

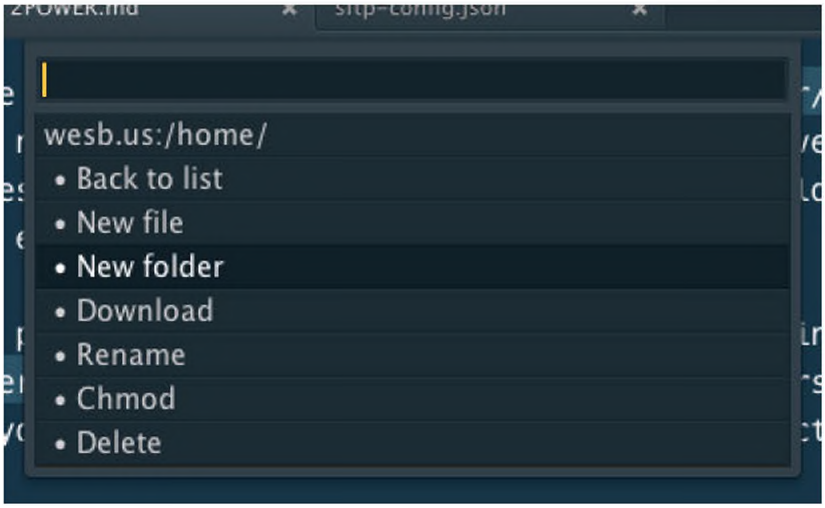
Because these files are stored as plain text on your hard drive, I recommend using SSH keys rather than a text password. That way if you get hacked or accidentally share one of these config files, there is no harm done.

When you are finished, `⌘S` to save it in `Packages/user/sftp_servers`. The file name will be used as the server name, so save it as something descriptive – spaces are allowed and you should *not* include any extension.

Once saved, pop open your command palette and type to find `SFTP Browse Server`. This will list all your available servers, just one if this is your first. Select the one you wish to connect to.



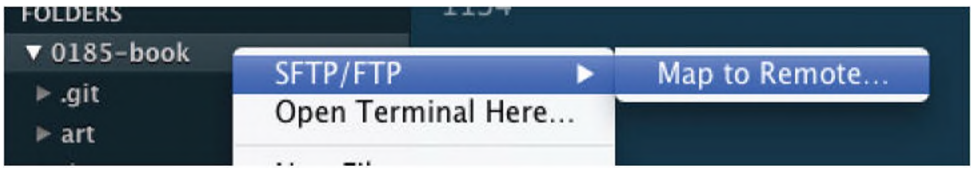
If all your credentials are correct, you will see a listing of all your server's files. Everything is done through the drop down palette, so resist the urge to use your mouse, use your arrow keys to traverse through your folders. At the top of each folder/file, you have a list of the usual FTP options such as creating, deleting and CHMODing the permissions.



Working with the drop down palette can be slow and a little disruptive of your workflow. At the time of writing there are a few solutions on the Will Bond's site that are in development to implement a sidebar with the remote files.

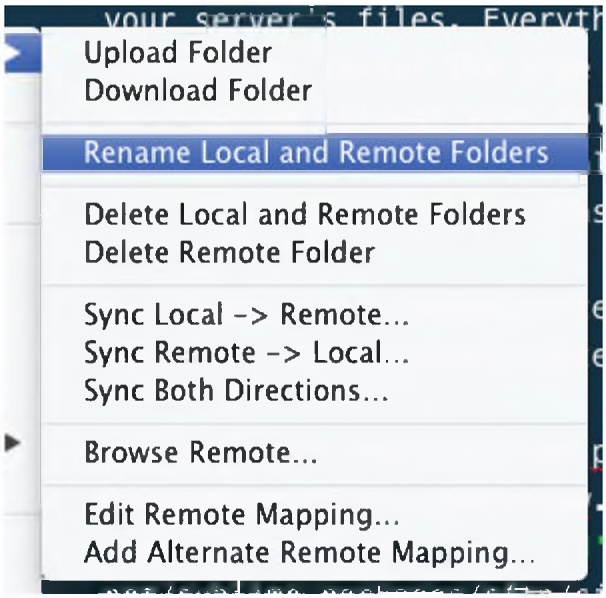
Mapping local to remote

The other (and safer) way to manage your FTP is to link a local directory to a remote one on your server. This way you make edits to your local copy and then sync your changes with the server. To set this up, open a project's folder in Sublime, right click the folder name and choose `SFTP/FTP` → `Map to Remote`. This opens up a JSON file similar to the one we filled out in the last section but with a few new options.



There are many options you can put in this file and I won't go into all of them here but one really great option is to set the `upload_on_save` to `true` which will automatically upload the file every time you save it. For a complete reference for all the options visit the SFTP Package Website.

Save the above file in the directory that you would like to map to the remote server, do not put in your users folder. Again, pay attention to whether or not you want to include this file in your version control as it may contain sensitive information.



SFTP → Filezilla

If you were a hardcore filezilla user before Sublime Text you may have years and years worth of FTP entries in FileZilla. Thankfully there is a nice little package called SublimeZilla which allows you to import all your old favorites into Sublime SFTP.

Transmit Doc Send

If you are coming from a text editor like Coda or used to using a visual interface for managing your websites, you may be used to saving a local copy of your file and then uploading it remotely with a key press. While the previously mentioned SFTP package does this, there is another useful package for users of Panic's Transmit.

To use this package, go ahead an install it from github or from package control. Once you have your local directory and remote FTP setup with Transmit, you can simply use the `Ctrl` + `u` keyboard shortcut to trigger a "dock send". Transmit will then figure out where the local file is, and which remote file it needs to replace. Voilà, the file has been uploaded.

Additionally, `Ctrl` + `Shift` + `u` will upload the file to the currently active transmit connection.

21.5 Tricky trick: Renaming and moving files

You may have already caught on that a huge part of this book is doing things without touching your mouse. The more you can do with you keyboard, the faster you will be. One part of Sublime that I was using my mouse for was renaming files. When this happens I would either right click the file in my sidebar and rename it, or do the same in my computer's file browser.

Just like almost everything in sublime, you can do this via the command palette. Just open it up and type rename. A text input will appear at the bottom of the screen for you to rename the file.

(rename style.css to /css/style.css)

So what? Well, the real trick here is something you can't otherwise do in Sublime Text, and that is move files from one folder to another – there is no dragging and dropping in the sidebar.

To move a file from one directory to another, simply append a directory structure before the file name. Here are a few examples with a file called style.css.

To move the file into a sub-directory called css:

rename `style.css` to `./css/style.css`

To move the file back to where it was:

rename `style.css` to `../style.css`

To create a directory called `base` inside of our `css` folder and then move the file into that directory:

rename `style.css` to `./css/base/style.css`

To get back up to our original spot, 2 directories up:

rename `style.css` to `../../style.css`

Note I've prepended everything with their `.` or `..`. In a file systems, theses mean **current directory** and **parent directory**.

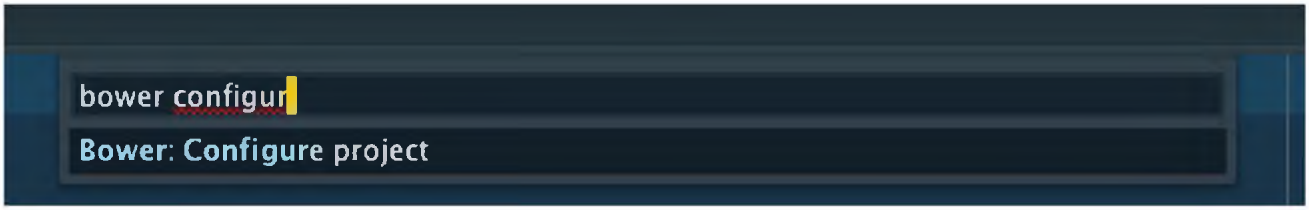
21.6 Bower Integration

If you haven't heard of it, Bower is a front-end package manager much like NPM is for Node, gem is for Ruby or PIP for Python. Rather than scouring the net and downloading .js files, you can install them right from your editor.

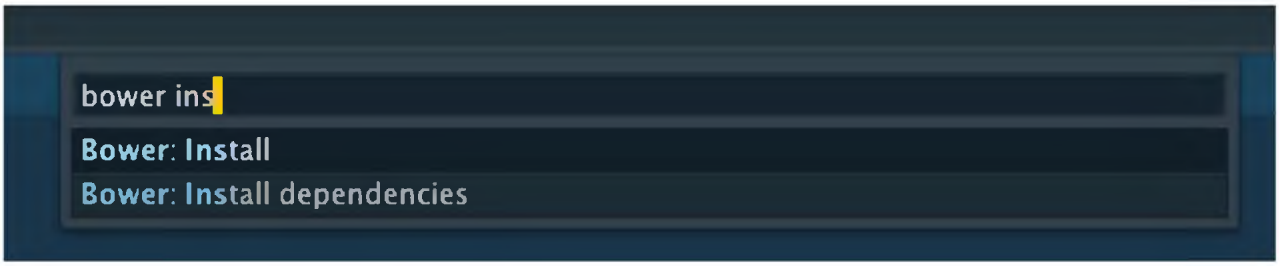
While it might not make sense to use NPM, gem or pip from your editor because you are in the terminal anyways, many front end developers can benefit from using bower right from Sublime Text.

First, make sure that you have Node installed, and then run `npm install bower -g` from your terminal.

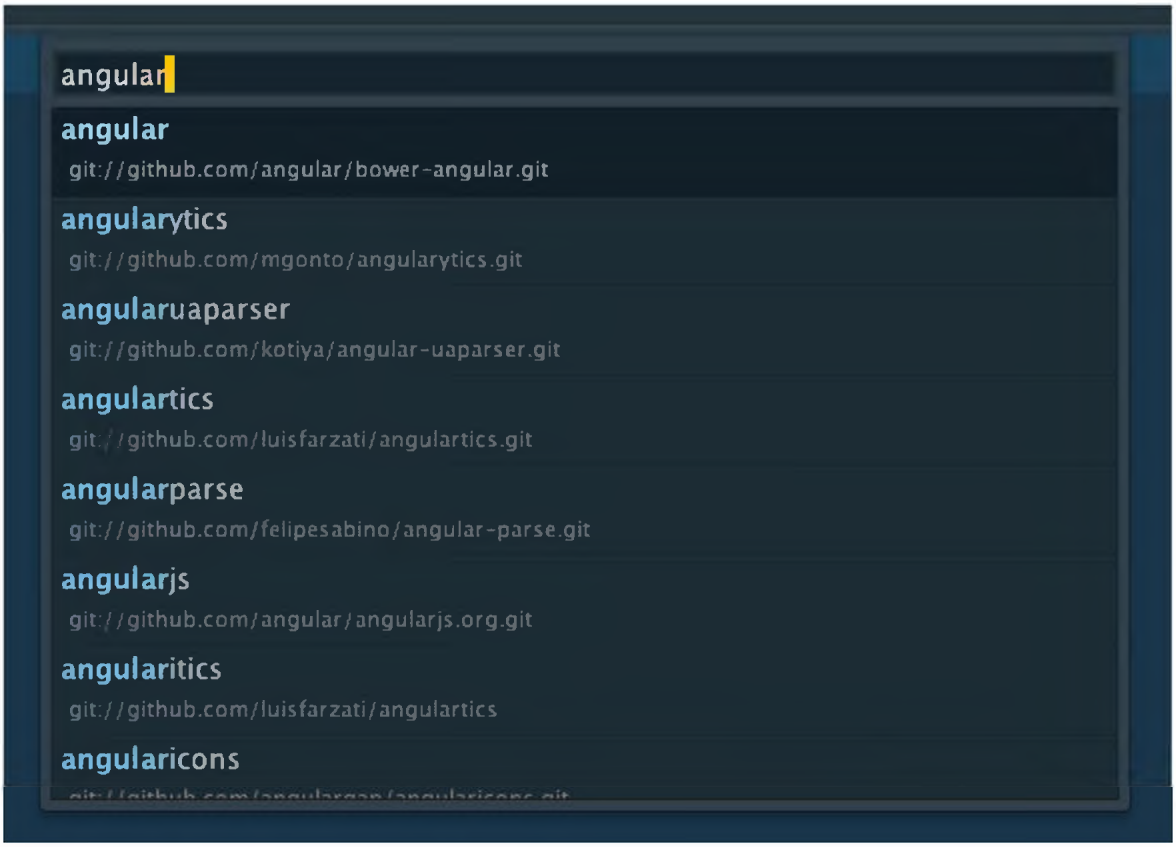
Next, install the bower package for Sublime Text.



First thing we need to do is do a bower config from the command palette. This will create a .bowerrc file inside your project which will house a few settings including what the folder is called.



Now from the command palette, type `bower install`



Then go ahead and search for the project you are looking for

Bower will create a folder called `components` and clone the entire repo into it's own folder

Bower Caveats

> This plugin, developed by Ben Schwarz, is meant to be extremely simple. There are a few things that you aren't able to do via the plugin. If you wish to save your dependences into a bower.json file or run a bower update, you will still have to do that from the command line.

Vim Mode

It's almost sacrilegious to mention Vim in a book about Sublime Text. Vim users *love* vim and even mentioning Sublime Text around a vim user can land you in a heated discussion. If you are passionate about *any* editor, you are in fantastic shape. I'm not here to debate which editor is better, but rather show you how you can harness the power of Vim right inside of Sublime Text. If you want to replicate 100% of vim inside Sublime Text, you should probably just use Vim, however there are many parts of Vim which are replicated in Sublime Text which are very welcome.

For those not familiar with vim, a quick overview: it's an editor that has been around for 20+ years. In fact, you probably already have vim installed, just open up your terminal and type `vim`. Vim a an extremely powerful tool that has a really stripped down interface. It's like sublime text in that you are able to move around quickly without the mouse and speed up your development with hundreds of possible packages.

A huge part of the power of vim comes from its non-standard way of interfacing with the text in the editor. Instead of using your mouse, you use a series of commands to move, select, cut, paste, delete, etc. Instead of being able to click around your file and type, select and move text at once, Vim introduces the concept of **modes** in which you can do different things in each mode.

22.1 Making Sublime Text act like Vim

Since this isn't a book about Vim, I'm not going go in depth about how to use vim, but rather how to get Sublime Text to behave like Vim.

22.2 Sublime Text 2

By Default, Sublime Text 2 ships with a package called "Vintage" that provides the vim functionality. The package is developed openly on GitHub and while fairly feature complete, it hasn't been updated in over a year. If you are using Sublime Text 2, this is the package you need to be using.

To get started, we simply need to enable it. Open your User Preferences with `⌘` + `,` and do a search for a setting called `ignored_packages`. You may have this setting already if you have used package control to ignore a package, if you have never ignored a package before, you will not see this setting. Since it's ignored by default, we simply need to overwrite the default setting by omitting it. Your setting should now look like this:

```
"ignored_packages": []
```

The above is just an empty array which overwrites the default of `"ignored_packages": ["Vintage"]`. If you have other ignored packages, your setting may look like this:

```
"ignored_packages":
[
    "SubliminalCollaborator",
    "MarkdownEditing",
    "Nettuts+ Fetch",
    "Goto-CSS-Declaration"
]
```

You can also turn on Vintage mode at a langauge level by adding `"ignored_packages": []` to your language specific settings file–more on this in the settings section.

22.3 Sublime Text 3

If you have upgraded to Sublime Text 3, using the Vintage package isn't the best option for you. While you can get it working after sifting through the forums for troubleshooting tips, there is a much better option called `Vintageous`.

Vintageous is a complete rewrite of the Vintage package made specifically for Sublime Text 3. To install, simply find it via package control and hit enter. Like all Sublime packages, the source is also available on GitHub.

22.4 Using Vintage Mode

Whether you are using ST2 Vintage or Sublime Text 3 Vintagous, the commands are basically the same.

- Press `esc` to enter normal/command mode.
- Press `i` to enter insert mode
- Press `v` to enter visual mode
- Most other motions from vim have had their keys mapped

You are also able to use the following Vim CTRL keys:

- `Ctrl+[` : Escape
- `Ctrl+R` : Redo
- `Ctrl+Y` : Scroll down one line
- `Ctrl+E` : Scroll up one line
- `Ctrl+F` : Page Down
- `Ctrl+B` : Page Up

To enable these control keys, use the following settings in your User settings file.

```
"vintage_ctrl_keys": true // ST2
"vintageous_use_ctrl_keys": true // ST3
```

OSX 10.7+

If you are using OSX Lion, Apple has implemented a press-and-hold to show character variations. This isn't ideal because often you will want to hold down the `hjkl` keys to move around your document. To disable this, open your computers terminal (Found in Utilities) and paste the following. A restart may be required.

```
defaults write com.sublimetext.2 ApplePressAndHoldEnabled -bool false
```

and for Sublime Text 3:

```
defaults write com.sublimetext ApplePressAndHoldEnabled -bool false
```

22.5 What's not included

Sounds too good to be true, right? Well, maybe. The Sublime implementation of vim is done entirely with key bindings, which means that everything was already possible with Sublime Text, but it has just been translated for those used to the vim keys.

Most importantly, Macros and commands haven't been ported as they are so tightly tied with Vim. Not to fear, Sublime has these things implemented as well, you will just need to learn or remap the keyboard shortcuts.

Language Specific Tweaks

So far, this book has been pretty agnostic towards which languages you work with - everything said so far is beneficial whether you are a frontend JS/CSS dev or a hardcore Python fanatic. Sublime has done a pretty good job shipping with defaults for most languages but they aren't perfect. As languages progress we are introduced to new syntaxes, design patterns and in the case of JavaScript and CSS, new languages!

If you work in any of the following languages, it will be helpful to take a look at what you can to do make your environment just that much better.

23.1 CSS

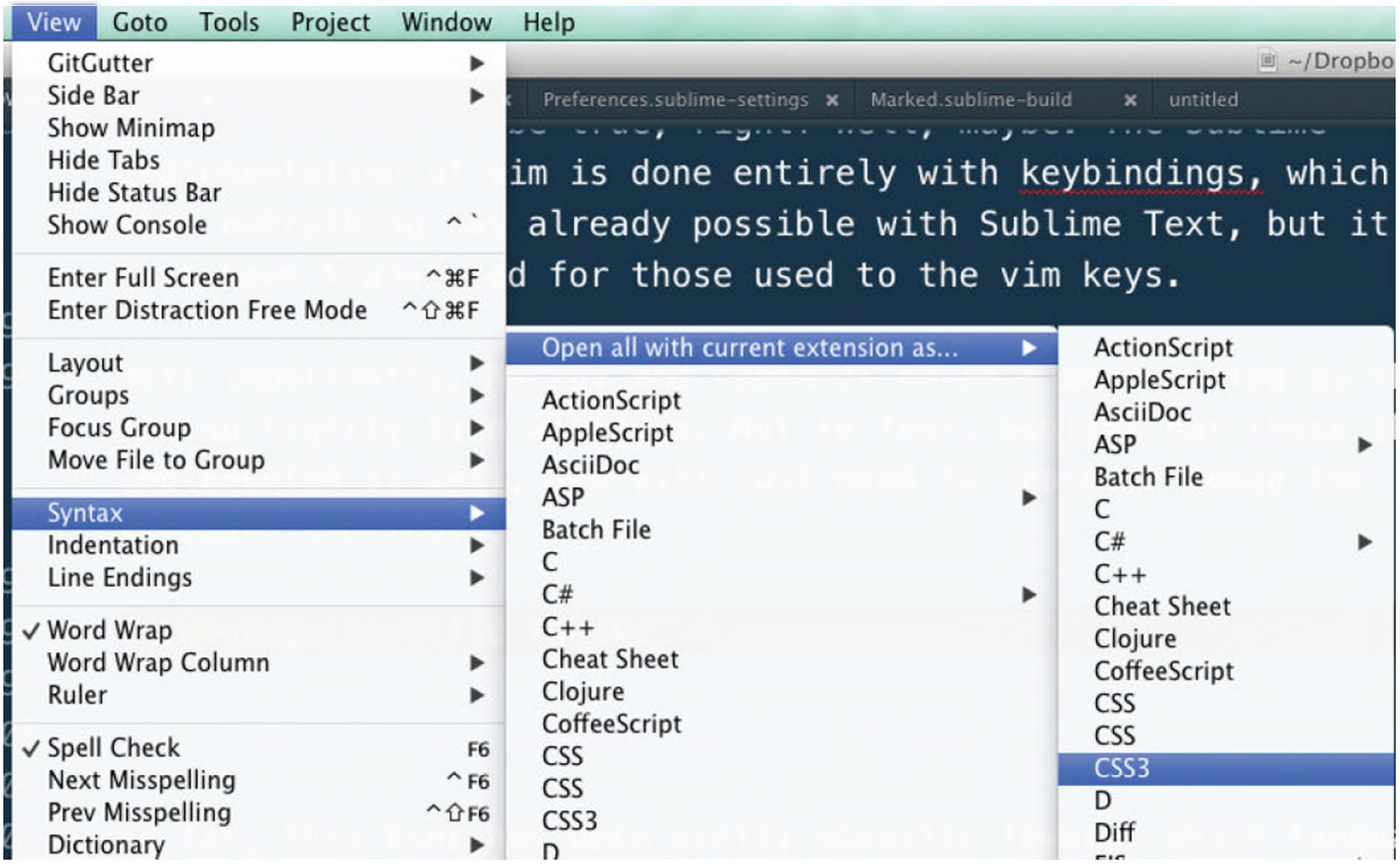
CSS3 has been around for some time now, but the syntax highlighting built into Sublime Text is still based on an old textmate plugin.

Update your syntax highlighting by installing the `CSS3 Syntax` package.

```
.wrapper {
  -webkit-transform:rotate(10deg);
  filter:blur(10px);
  transition:all 0.5s;
  animation:bounce;
  display: inline-block;
  color:gold;
}
```

```
.wrapper {
  -webkit-transform:rotate(10deg);
  filter:blur(10px);
  transition:all 0.5s;
  animation:bounce;
  display: inline-block;
  color:gold;
}
```

Once done, you can set all your .css files to be highlighted using this new syntax file rather than the old one. Open any .css file and go to `view` → `syntax` → `Open all with the current extension as...` → `css3`



23.2 LESS, SASS and Stylus

Chances are you are using some sort of CSS pre-processor to write your CSS. There are a number of tools that can make writing LESS,SASS,SCSS and Stylus even more fun.

Syntax Highlighting

The first is to get a proper syntax highlighter. Editing your files in CSS mode will work somewhat, but to take advantage of all the syntax highlighting features you will need to install the following for each language. All of these are available on github or via package control.

LESS: LESS-Sublime SASS: SASS-Textmate-bundle – also comes with a nice set of snippets Stylus: Stylus Syntax Highlighting

Helpful Tools

Variables are a huge part of CSS Preprocessors and being able to access them via a handy menu can really speed up your development time. The plugin sublime-list-stylesheet-vars does this for all three languages and is a recommended installation when working with one of these CSS preprocessors.

23.3 Coffeescript

Writing coffeescript in Sublime is a hoot with the CoffeeScript Sublime Plugin. This package provides a ton of functionality surrounding the language including syntax highlighting, snippets and building CoffeeScript.

You may be interested more in CoffeeScript compiling (see the build section) or CoffeeScript linting (see code quality section).

23.4 Templating: HAML, Slim, EJS, Jade

Just as with CSS preprocessors, the web development world is filled with templating languages.

HAML: Comes default with Sublime Text's Ruby package

Slim: A collection of snippets and a syntax highlighter via slim-template

EJS: EJS syntax highlighting via EJS.tmLanguage

When working with these templating languages, you may find yourself with HTML that you need to turn into HAML or Jade. For times like these, utilize the html-to-jade and html-do-haml packages.

23.5 JavaScript

I am primarily a JavaScript developer, so fine tuning my environment has been a ongoing pursuit. I've already talked about the live linting tools in this book but there are a few other things we can do to make developing JavaScript in Sublime a pleasure.

First off, the JavaScript syntax highlighter that comes with Sublime does a very poor job at properly highlighting you code. It also isn't updated for all the new goodies we are seeing in ES5/ES6.

To remedy this, I recommend that you install the JavaScriptNEXT language for Sublime Text which is available in package control under **JavaScriptNext - ES6 Syntax**. The author, Brandon Benvie, also has supplied a theme which is available on the github page

I also recommend installing the JavaScript snippets pack by JP Richardson. It is available on GitHub and package control.

Make sure you also read the section on JSHint Gutter below.

23.6 jQuery

Code faster with the jQuery snippets pack

23.7 Node.js

The Sublime Text Node.js package provides a set of snippets, code completion of the Node API, as well as a handful of useful commands for running Node from Sublime Text.

23.8 PHP

- PhpCS This plugin adds PHP CodeSniffer, PHP Coding Standards Fixer, the PHP Linter, PHP Mess Detector, Scheck support to Sublime Text.
- PhpDoc CodeDoc is a Sublime Text 2/3 plugin to speedup writing documenting comments.
- PhpBeautifier

23.9 Wordpress

Sublime Text WordPress Package provides autocompletes and snippets for all of the WordPress functions, hooks, constants and classes.

23.10 Python

- Python Auto-Complete auto-completes built-in functions of the python language.

23.11 Ruby

RuboCop is a static code analyzer that help your keep your coding style up to the standards of the Ruby Style Guide. The SublimeLinter-rubocop package will integrate this right into Sublime.

Must have Add-on Packages

In addition to all the packages talked about in this book, there are a handful of packages that almost all developer will find useful.

24.1 Emmet

I've already written in depth about Emmet and how it is an absolute must-have for anyone writing HTML and/or CSS. Read the Emmet section of this book for everything you need to know about it.

24.2 Autofilename

As you type paths to images/stylesheets/javascript, your editor should suggest the next folder or filename. Surprisingly, this isn't a native feature of Sublime Text.

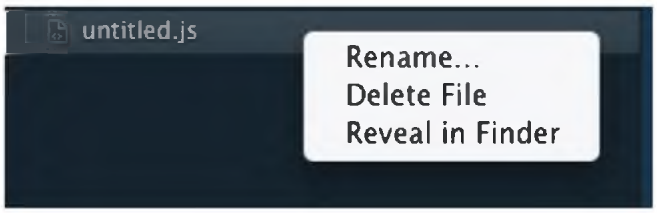
Installing Autofilename will add this functionality, so as you type, you can see a list of folders, files and even image width/heights.

24.3 HTML + CSS + JSON Prettifyer

Pretty JSON and HTML-CSS-JS Prettify are handy utilities that will take messy/poorly formatted HTML, CSS or JSON and format and indent it properly. Very handy when you receive a blob of generated code/json that needs to be readable.

24.4 Sidebar enhancements

The sidebar and the sidebar API in Sublime Text are pretty lacking in functionality, and this can be a burden to users who are coming from editors that are more GUI driven - some things just need to happen on a right click.



Lackluster Default Sidebar

SidebarEnhancements adds a *ton* of new functionality to the sidebar.

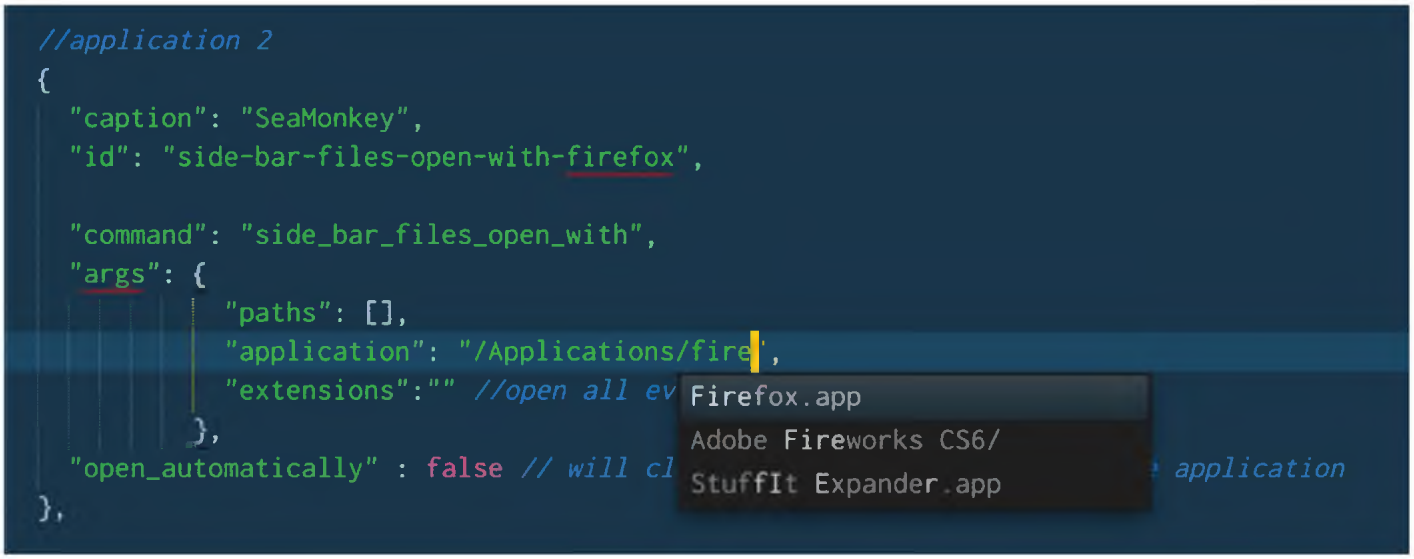
Open with...

Let's start with opening files in other programs from the sidebar.

Right click and thne `Open / Run` will open the file in the operating system's default program for that file type. If it is an HTML file, it will open in your default browser. For me, HTML files open in Chrome Canary.

But what if I want to open it in FireFox from the sidebar? We need to add another program. Right click the file, `Open With` → `edit applications...`. This will open up a `.sublime-menu` JSON file. Go ahead and copy one of the current applications and fill it out with the information for FireFox.

The two important lines to change here are `"ID"` and `"application"`. The application path will be easy to fill out if you have previously installed the AutoFileName extension.



Autofilename extension recognizes the path and suggests applications

The extension line will limit this application to certain file types. It makes sense that we only want to open `.HTML` and `.htm` files, so we use `"extensions":"html|htm"`. Leaving it blank (`"extensions":""`) will show up for everything including folders and you can also use wildcards such as `"extensions":".*"` for any file with extension.

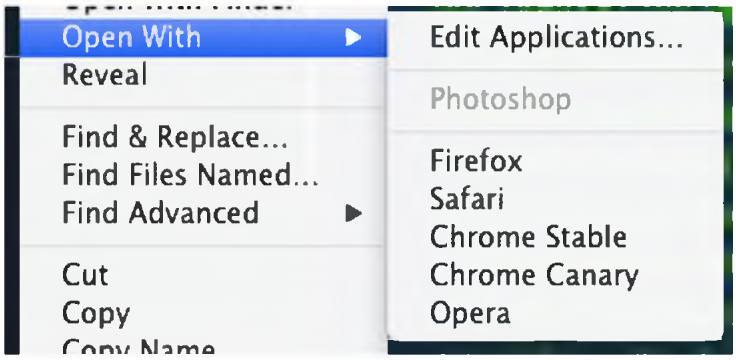
Once all is said and done, we have successfully added Firefox to our sidebar "*Open With*" using the following code snippet:

```
// Open with Firefox
{
  "caption": "Firefox",
  "id": "side-bar-files-open-with-firefox",

  "command": "side_bar_files_open_with",
  "args": {
            "paths": [],
            "application": "/Applications/Firefox.app",
            "extensions":"html|htm"
```

```
        },
  "open_automatically" : false // will close the view/tab and launch the application
},
```

If you are on Windows or Linux, make sure you use the appropriate path to your programs.

Want all of them? Use the following snippet to add all the browsers to your sidebar:



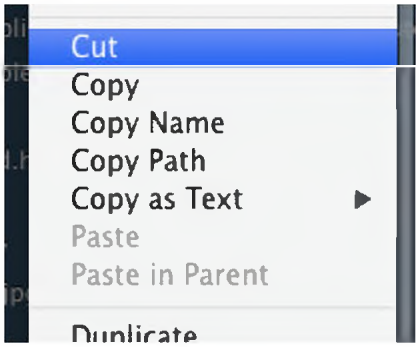
All Mac Browsers

```
// Open with Firefox
{
  "caption": "Firefox",
  "id": "side-bar-files-open-with-firefox",

  "command": "side_bar_files_open_with",
  "args": {
            "paths": [],
            "application": "/Applications/Firefox.app",
            "extensions":"html|htm"
          },
  "open_automatically" : false // will close the view/tab and launch the application
},

// Open with Safari
{
  "caption": "Safari",
  "id": "side-bar-files-open-with-safari",

  "command": "side_bar_files_open_with",
  "args": {
            "paths": [],
            "application": "/Applications/Safari.app",
            "extensions":"html|htm"
          },
  "open_automatically" : false // will close the view/tab and launch the application
},

// Open with Chrome Stable
{
  "caption": "Chrome Stable",
  "id": "side-bar-files-open-with-chrome-stable",
```

```
  "command": "side_bar_files_open_with",
  "args": {
            "paths": [],
            "application": "/Applications/Google Chrome.app",
            "extensions":"html|htm"
          },
  "open_automatically" : false // will close the view/tab and launch the application
},

// Open with Chrome Canary
{
  "caption": "Chrome Canary",
  "id": "side-bar-files-open-with-chrome-canary",

  "command": "side_bar_files_open_with",
  "args": {
            "paths": [],
            "application": "/Applications/Google Google Chrome Canary.app",
            "extensions":"html|htm"
          },
  "open_automatically" : false // will close the view/tab and launch the application
},

// Open with Opera
{
  "caption": "Opera",
  "id": "side-bar-files-open-with-opera",

  "command": "side_bar_files_open_with",
  "args": {
            "paths": [],
            "application": "/Applications/Opera.app",
            "extensions":"html|htm"
          },
  "open_automatically" : false // will close the view/tab and launch the application
},
```

More Features

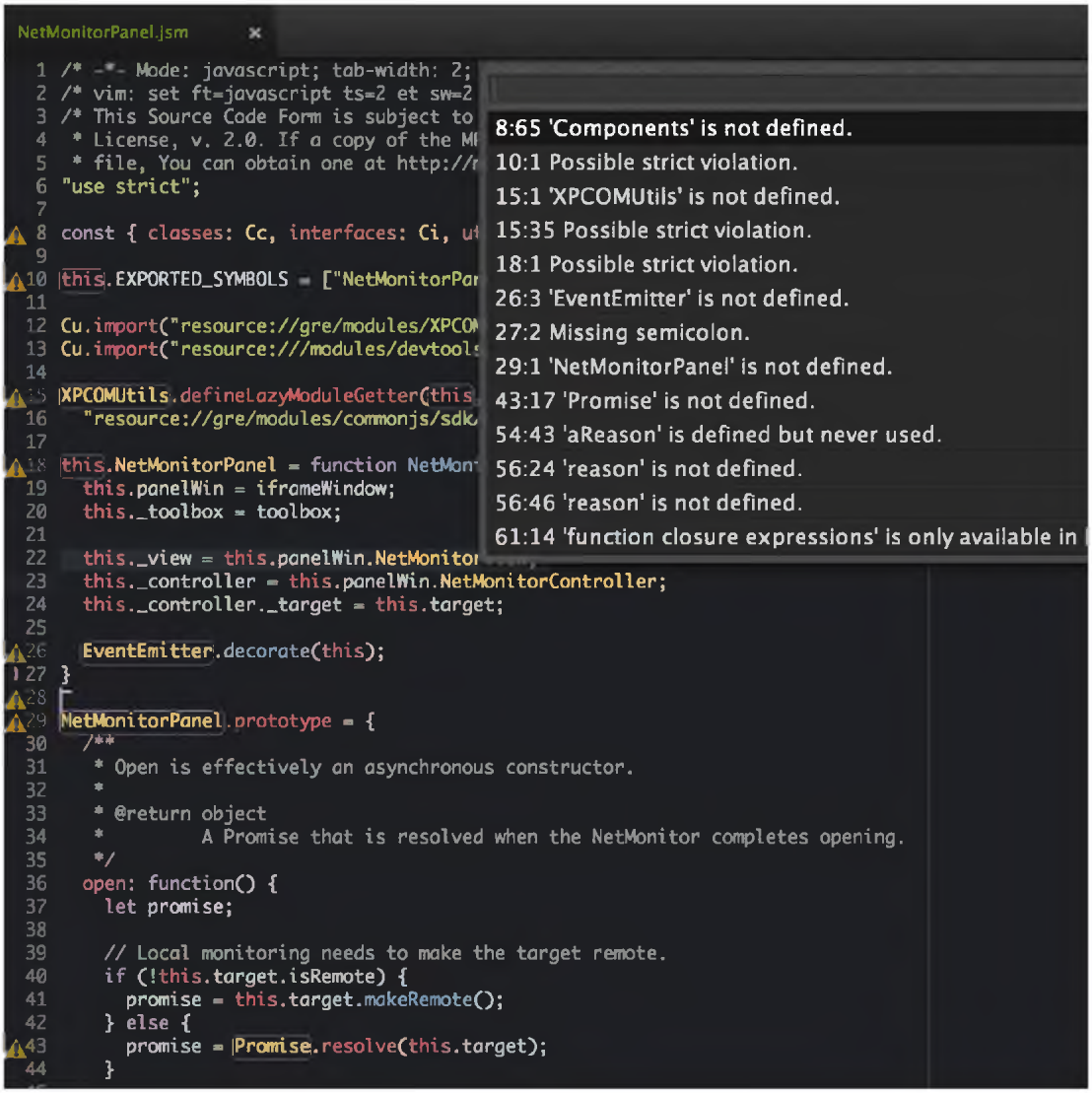
These ones are fairly straight forward and do not require much explanation. cut / copy / paste via the sidebar, something Sublime doesn't have by default.



As well as a few more advanced features around copying the file, copying the path, duplicating a file, copying as base64, paste in parent and a handful of edge case

24.5 JSHint Gutter

JSHint gutter is an alternative to Sublime Linter if you just need JavaScript linting. It can be easier to setup and provides live linting support, placing flags in the gutter when something is awry.



24.6 Alignment

The Alignment package is a simple yet very popular add on for Sublime Text.

Some coding style guides require that lines of text be aligned in a certain way. One of those could be having your `=` align up when doing variable declaration.

With this package, we can take code that look like this:

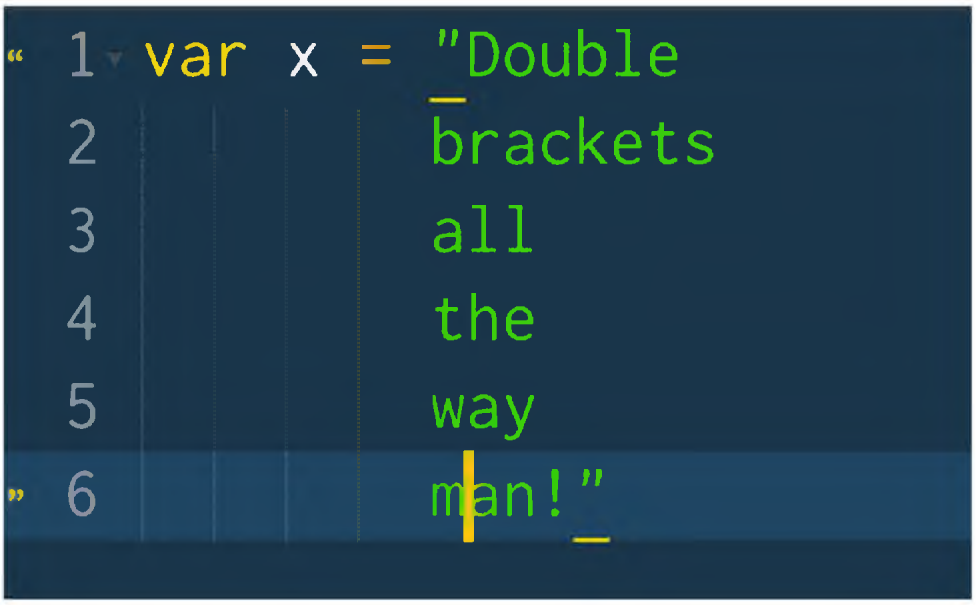
```
1 var x = 100;
2 var reallyLongVariab = "short value;";
3 var shortName = "this is a really long value";
4 var y = 200;
5
```

Highlight it all and hit `⌘` + `Ctrl` + `A` (`Alt` + `Ctrl` + `A` on windows) making this:

```
1 var x                 = 100;
2 var reallyLongVariab = "short value;";
3 var shortName         = "this is a really long value";
4 var y                 = 200;
5
```

24.7 Bracket Highlighter

Bracket highlighter is a very visual aid helping you see where yours brackets and tags start/stop:



Multiple lines

Matches all types of brackets

Works in CSS Too!

Getting colored brackets requires that you install a theme that supports it. Cobalt2 is ready to go.

24.8 Writing Markdown with Sublime Text

Markdown is by far the best way to write. Everything from your project's readme to blog posts to larger documents such as this book can all be written in markdown.

Using your code editor as a writing tool is far superior to other tools such as Word or Google Docs. That said, Sublime needs a few packages before you can get the most out of it as a document writer.

Syntax Highlighter

Sublime Text comes with a fairly good syntax highlighter, so without any packages you are already setup to start writing markdown.

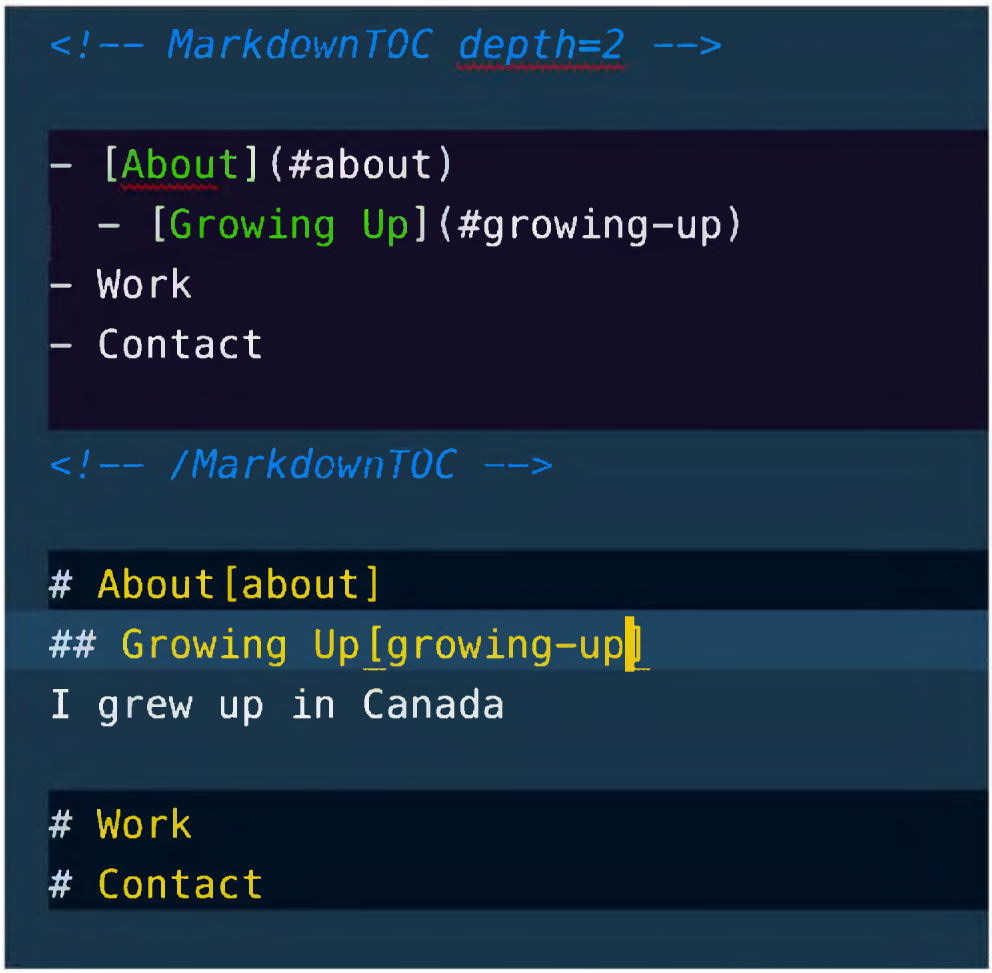
The package MarkdownEditing provides a suite of tools including two advanced themes, auto-pairing of markdown syntax such as _underscores _ and *asterisks*, as well as a handful of keyboard shortcuts.

Compiling

While I recommend using Sublime Text to edit and another program (such as Marked or Mau) to compile your markdown, there are options available if you wish to compile right from Sublime Text. Markdown Preview will allow you to compile to HTML and even preview it in the browser with a single keyboard shortcut.

Table of Contents

MarkdownTOC is a package that will generate a table of contents for your entire document. Simply run the command `MarkdownTOC: Insert TOC` once where you wish to have the table of contents, and it will update the TOC on every save.



Once the comments are in your code, you have the ability the specify how many title levels deep the TOC will go.

By default, the TOC won't be linked to the title, which is rather annoying, but you can remedy that by giving each title an ID. MarkdownTOC will notice the ID and auto-link them.

```
## Growing Up[growing-up]
```

I must caution that update the TOC for large documents can be fairly slow and it seems to block the editor while it is generating.

24.9 Maintaining State on a file

If you work on large documents, code folding is a great way to hide large chunks of code when you aren't working on them. The problem with code folding in Sublime Text is that the folds to not persist between sessions. This means that if you restart sublime text, all your folds are gone!

BufferScroll is a package that will remember a whole slew of things when you close a file, including:

- scroll position
- cursor positions
- selections
- marks
- bookmarks
- foldings
- selected syntax
- colour scheme

24.10 Expand to quotes

This package was referenced earlier in the Jumping, selecting and moving section. Sublime has expand selection to [line,word,paragraph,scope,brackets,indentation] but no "expand to quotes". So, install this one to extend the native functionality to expand to quotes. Helpful when working with strings or HTML attributes.

https://github.com/kek/sublime-expand-selection-to-quotes

24.11 TODO

If you are the type that adds TODO comments littered through your code, Sublime TODO is a handy little package that will gather all the references to TODO comments and compile them into a single, linkable screen.

https://github.com/robcowie/SublimeTODO

Tip + Tricks Grab Bag

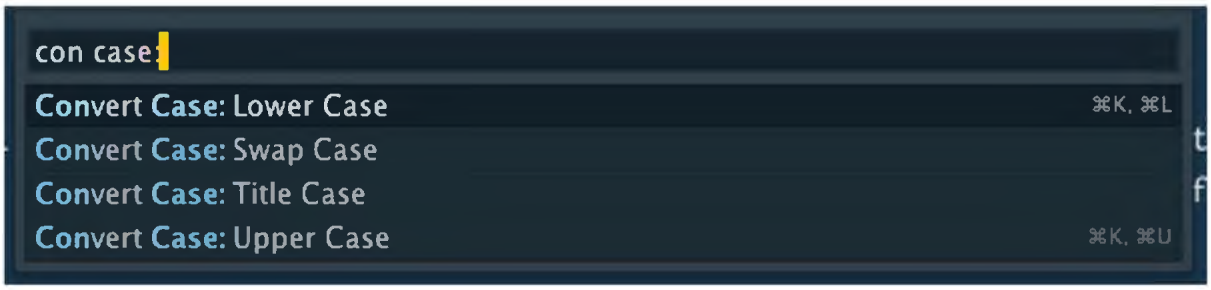
This second of the book is filled with tiny, yet extremely useful hot tips. While they might not fit into a chapter of this book, you will no doubt find many cases where they are useful in your development workflow.

25.1 Converting Case

Whether you are referencing a constant or just toning down some content from an overly zealous client, the case converting feature of sublime text allows you to convert from and to upper and lower case as well as swap and Title case.

Lower and Uppercase are bound to `⌘` + `K`, `⌘` + `U or L`. Swapping case and titling case do not have a keyboard shortcut but can be easily added with the `title_case` and `swap_case` commands and Chapter 15 of this book.

These commands are quickly available via the command palette and slowly available via the `Edit` menu item.



25.2 Code Comments

Every language has code comments. Toggle these commends with `⌘` + `/`. If nothing is selected the entire line will be commented out.

If the language supports block commenting, add in `option/alt` into the mix to block comment multiple lines of selected code. You can always uncomment a block comment when the cursor is anywhere block.

25.3 Sort, Reverse, Unique and Shuffle

Take the following code for example. 8 lines with 2 duplicates.

```
<img src="item05.jpg">
<img src="item02.jpg">
<img src="item01.jpg">
<img src="item02.jpg">
<img src="item04.jpg">
<img src="item06.jpg">
<img src="item03.jpg">
<img src="item04.jpg">
```

First thing we need to do is get rid of the duplicates. This is only available via the `Edit` → `Permute Lines` menu, however you can add a keyboard shortcut with the `permute_lines` command and the `operation` argument set to `unique`.

Selecting Unique will remove duplicate lines, leaving up with:

```
<img src="item05.jpg">
<img src="item02.jpg">
<img src="item01.jpg">
<img src="item04.jpg">
<img src="item06.jpg">
<img src="item03.jpg">
```

Now the lines are already mixed but, but if we wanted some sort of randomness to them, we could use the `shuffle` command.

Finally we want to order them, for this use use `Sort Lines`, available under the Edit menu or the command palette. That leaves us with:

```
<img src="item01.jpg">
<img src="item02.jpg">
<img src="item03.jpg">
<img src="item04.jpg">
<img src="item05.jpg">
<img src="item06.jpg">
```

By now you get the point - we could also reverse them again with the `reverse` command.

25.4 Distraction Free / Fullscreen Mode

Sublime Text isn't just for writing code - we reviewed how to use the editor as a fantastic markdown editor in Chapter 23. Whether you are writing prose or code, sometimes we just need to focus on what we are writing and nothing else.

For this, Sublime Text offers `Fullscreen Mode` mode which gives you a fullscreen editor. For Mac OSX users, this takes up it's own space.

One step further, Sublime offers a `Distraction Free` mode which hides all tabs, gutters, line numbers and other editor chrome. It's just you and your code.

You can customize the settings for Distraction Free mode in `Distraction Free.sublime-settings` available via `Preferences` → `Settings - More` → `Distraction Free`.

fin

That's all folks. Congrats on making it through the entire book. You have learned a lot and are well on your way to becoming a Sublime Text Power User! Keep practicing the shortcuts and referencing this book.

26.1 Updates

Keep an eye on your email for updates to the book - I will let you know when a newer version is released. If you have ideas or corrections you would like to suggest, please email me at wes@wesbos.com.

26.2 Have a question?

Have a question about Sublime Text? There are a number of ways to get help:

- Tweet at me on twitter @wesbos
- Ask a question on Stack Overflow and tag it with Sublime Text - <stackoverflow.com/questions/tagged/sublimetext>
- Register and visit the official Sublime Text forms - http://www.sublimetext.com/forum/
- Join the IRC chat on freenode - /j ##sublimetext