1.3

**Install or upgrade Node.js**

Check if you have node installed and which version you have in the terminal – run:

**`node –version`**

If you're running 0.10 or prior, you'll need to upgrade. If you see a node command not found error, then you don't have Node.js installed.

If you need to, go to nodejs.org and click the button for the most recent stable version. That will download the package to your computer, then you can double click that, and run through the installer. Once the installation is complete, you can close out of it, open up terminal again. You'll need to open up a new window, so close this window out and then hit Command N to open a new window, and then type node --version and you should see the version of Node that you installed.

If you're using Windows, be sure to restart your computer before running the node version command from the command line, to ensure the Node.js installation completed. Now that you have an appropriate version of Node.js installed, you can leverage it to run libSass.

1.4

**Install Grunt-Sass**

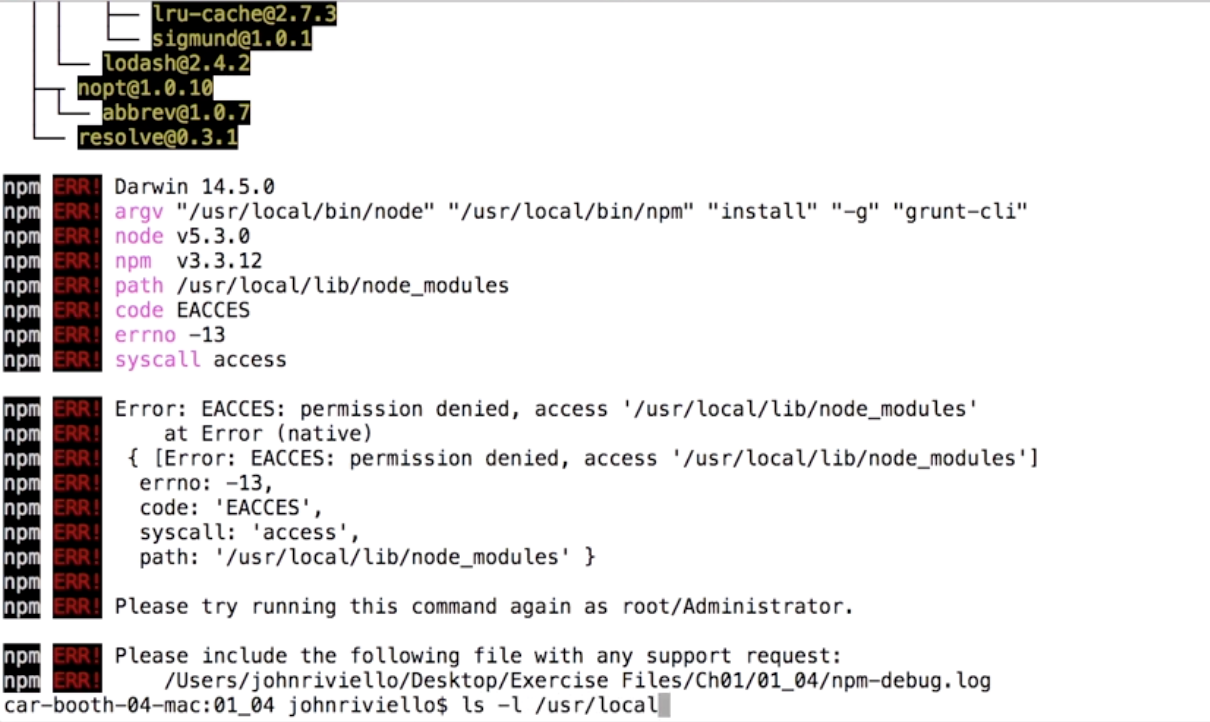
To make it easier to interact with and leverage other NPM modules, use the Grunt task runner.

If you're not familiar with Grunt (uses libSass), it is a node.js based tool to help you execute various NPM modules. There are other task runners, such as Gulp, and Broccoli (they use libSass instead of Ruby Sass) that also offer node-sass tools.

To use Grunt, you first have to install the Grunt CLI, or command line interface tool, globally, which will provide you with a Grunt command. To do that, open up the command line and then run:

`**npm install -g for global, grunt-cli**`

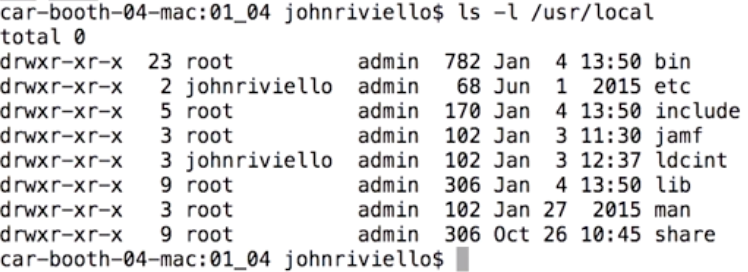
That will then attempt to install the latest version of Grunt CLI. If the install failed with a permissions error then it's probably the first time you've tried to globally install a node package, and therefore you need to override your permissions in user local to have yourself as the owner instead of root, which is the default.



You can verify that by running:

`**ls -l/ usr/local**`

Which will output something like this:



As noted from the error from NPM, we need permission to write to /usr/local/lib/node\_modules. And we see here that lib is currently owned by root. We're going to address that by granting ourselves access to all of user local by running:

**sudo chown** for change owner, **- R** for recursive, so that the folder and all the folders inside of it will have this change applied to them, **$USER** which refers to our user, then /usr/local.

**`sudo chown -R $USER /usr/local`**

Press enter, enter your password, press enter again.

Now, if you press the up arrow twice, and rerun `**ls -l /usr/local`** command, you'll see that you're now the owner of all the subfolders of user local.

Now we can safely run:

**`npm install -g grunt-cli`**

without the need to prefix that with sudo, and npm successfully goes out, fetches Grunt CLI, and installs it globally.

You can verify that worked by running:

`**grunt –version**`

You should see a version number for Grunt CLI.

Now create a basic package.json file to keep track of all the node packages we'll be using locally in this project.

I'm going to name this h-plus-sport since that's the name of the website we'll be creating. And I'll set private to true since we're creating a website and not an actual node package, so that will help silence some warnings. It should look like this:

**{**

**"name": "project-name",**

**"private": true,**

**}**

**}**

Let's save that file now. Now we can start installing Node.js packages from the command line and save them to our package.json file.

Start off by installing Grunt in your local project folder.

In the Terminal, make sure you’re in the correct directory, and run:

**`npm install grunt --save-dev`**

That will install Grunt locally in the project, and once that completes, if you open our package.json file, you see the save dev option added Grunt as a dev dependency which will allow other developers to install Grunt in other dependencies easily by running npm installed from the root of the project.

**"devDependencies": {**

**"grunt": "^1.0.3"**

**}**

Now, let's install the node-sass Grunt plugin via the same method. In the Terminal, and I will run:

**`npm install grunt-sass --save-dev`**

And you'll see there it installs node-sass as well as grunt-sass since node-sass is a dependency of grunt-sass. Now that we've installed a couple of node modules, we should add the node modules directory to our source control ignore file, so those are not checked into the project since they should be added to the project via an npm install command.

Since I'm using git, I'll create a git ignore file. So, back to your text editor and create a new file called `**.gitignore`** and just add `**node\_modules`** to the top of the file.

1.5

### Configure Grunt-Sass

With Grunt-Sass installed, now you can create our Grunt file and configure it to compile Sass.

Create a file called: `**GruntFile.js`**

Copy this code into the file:

**const sass = require("node-sass");**

**module.exports = function(grunt) {**

**grunt.initConfig({**

**sass: {**

**options: {**

**implementation: sass,**

**outputStyle: "expanded"**

**},**

**dist: {**

**files: [**

**{**

**expand: true,**

**cwd: "assets/sass/",**

**src: "\*\*/\*.scss",**

**dest: "assets/css/",**

**ext: ".css"**

**}**

**]**

**}**

**}**

**});**

**grunt.loadNpmTasks("grunt-sass");**

**grunt.loadNpmTasks("grunt-contrib-watch");**

**};**

The `**options`** part of the object can take any option that's listed in the GitHub Read Me for Node Sass except for a few.

The output style is set to expanded. You want to configure this to whatever style most closely matches your existing CSS to add base, so that when you start the process it'll actually.

There’s a target defined which is called dist, short for distribution. And here you specify the files this target should run against, and what it should create. This is an array, here if it contains only one object, it is defined dynamically so it can process multiple files. To do that, enable Dynamic Processing, which is done by setting expand to true. That allows access to all these options listed on the Grunt JS website for building the file object dynamically.

Current Working Directory, **cwd** is set to **assets/sass**

Source files are set to all .**scss** files in the current folder and sub-folders

Destination directory is set to the **assets/css**, and lastly the file extension is be **.css**

To compile a sass file into a css one, go to the Terminal and, in the root folder of the project, run:

**`grunt sass`**

In the CSS folder, you’ll see that grunt has compiles CSS from our Sass file

1.6

**Speed up development with Grunt Watch**

How to watch SCSS files and compile them to CSS whenever you edit them.

Go to the Terminal, and run:

**`npm install grunt-contrib-watch --save-dev`**

As you can tell from the name having contrib, this plugin is maintained by the Grunt team.

Once that is installed go to Text editor and edit the Grunt file. You should add lines to make it look like this:

**```const sass = require("node-sass");**

**module.exports = function(grunt) {**

**grunt.initConfig({**

**sass: {**

**options: {**

**implementation: sass,**

**outputStyle: "expanded"**

**},**

**dist: {**

**files: [**

**{**

**expand: true,**

**cwd: "assets/sass/",**

**src: "\*\*/\*.scss",**

**dest: "assets/css/",**

**ext: ".css"**

**}**

**]**

**}**

**},**

**watch: {**

**files: "assets/sass/\*\*/\*.scss",**

**tasks: "sass"**

**}**

**});**

**grunt.loadNpmTasks("grunt-sass");**

**grunt.loadNpmTasks("grunt-contrib-watch");**

**grunt.registerTask("default", ["sass", "watch"]);**

**};```**

For the properties for this you can view them on the **grunt-contrib-watch** page on github.com.

And we see that watch takes a couple of options: **files** and **tasks**, and those take either a string or an array.

**`files**: “**assets/sass/\*\*/\*.scss”** `

You want to have all folders and subfolders, and then the .scss files.

**`tasks:** **“sass”`**

You want it to run, in your case for now, it'll just be the sass task.

This says whenever any of the SCSS files in this folder or subfolders are changed, to run the **“sass”** task.

To check it’s working, go to the terminal and run:

**`grunt watch`**

You can see that grunt is running the watch task and it's waiting for a file change. Go back to your editor and modify your Sass file to see if it picks it up.

Make a change in the scss file and save it. Now look at your styles.css file to see the if it changed there as well. If you go back to the Terminal, you should see that it recognized that the styles.scss file changed and it then ran the sass task.

To stop the command from watching the files just press: **control C** and you'll see it exit.

Define a default Grunt task:

The most common thing you and the other developers on your team will be doing is running Grunt Watch, so let's make that the default Grunt task. To do that, go back to your text editor, and in our Grunt file we're going to add a line at the end:

**grunt.registerTask("default", ["sass", "watch"]);**

This will set up an alias for a default task and tell it to run the sass task first, followed by the watch task, which will run the sass task again when we edit a Sass file.

The default alias name is special for Grunt in that it means you can simply run `**grunt**` from the command line and the default task will run.

Save this file, then we'll test it in the Terminal – run:

**`grunt`**

This time, it runs the sass task and starts up the watcher. You can then press control C again to stop that process.

Your development setup with Grunt Watch will now make it easy for you and other developers to quickly get started developing Sass.

1.7

The original syntax of Sass includes no curly braces, no semicolons and uses indentation to delimit block of styles. It's more concise, but it does not look like CSS.

SCSS looks exactly like CSS. You can take your CSS files, move them to your Sass folder, change the extension to SCSS and you have valid SCSS. So, for your first commit of an SCSS file, you can do just that.

One feature of Sass that is pretty powerful is the ability to nest your rules. The Ruby version of Sass actually comes with a commandline tool to convert your CSS to SCSS and automatically add nesting to the SCSS files. *(I’m using the lib version of sass with node-sass).* That does not exist in LibSass.

This online tool provides the same functionality: css2sass.herokuapp.com.

Just don't blindly use the tool though as although it will generate the same styles for you, it may alter your selectors a bit. For example, if you have multiple selectors of the same block of styles, and then one of those selectors is used separately, as we do in our CSS file with the sub and sup selectors here, it will duplicate each selector in the multiselector block style and add the additional styles instead of maintaining the multiple selectors.

After you run the tool, it may require some manual adjusting to get it where you like it. If you have a large amount of CSS you're converting, you may want to skip this automated step entirely and just add nesting as you wish later on manually.

*e. g. Another spot this happens is down below here. So we have this webkit appearance button and cursor pointer and a couple of spots here that are all in the SCSS covered by this one line of selector. So there's four selectors there. We can just copy this from the CSS and paste it in the SCSS. Just below that, we see the border and padding are set in multiple spots here, and that selector is right here so I'll go ahead and I will copy that.*

Also, there's the potential for comments to end up out of placeIf we take a look back at your original CSS and search, you can see where it is meant to be.

When you're writing new Sass code, it's easy to get carried away with nesting and go many levels deep. This typically results in bloated, unnecessary CSS. So it's best to follow what's known as the inception rule, which was coined on this article on the Sass way. **The rule states that you should never nest more than four levels deep.**

Additionally, now that you're using nesting to create selectors, you're generating selectors for browsers to use that don't exist in your actual Sass source code. So if you're used to inspecting the element with the browser's developer tools, copying the selector and searching for it in your code base, your searches will come up empty if the selector was generated due to nesting. This sways some developers away from using nesting at all in Sass. I personally still enjoy the organization and reduced typing nesting affords, and with the debugging techniques that Sass offers, you can be working just as fast as you were with vanilla CSS.