**Links, Resources, and things mentioned**

* [Node JS](https://nodejs.org/)
* [Grunt JS1](http://gruntjs.com/)
  + [grunt-cli](https://github.com/gruntjs/grunt-cli)
  + [grunt-sass1](https://github.com/sindresorhus/grunt-sass)
  + [grunt-contrib-concat](https://github.com/gruntjs/grunt-contrib-concat)
  + [load-grunt-tasks](https://github.com/sindresorhus/load-grunt-tasks)
  + [grunt-contrib-jshint](https://github.com/gruntjs/grunt-contrib-jshint)
  + [grunt-contrib-watch](https://github.com/gruntjs/grunt-contrib-watch)
  + [grunt-contrib-csslint](https://github.com/gruntjs/grunt-contrib-csslint)
* [Gulp JS1](http://gulpjs.com/)
* [Grunt Plugin Directory](http://gruntjs.com/plugins)
* [npm](https://www.npmjs.com/)
* [JSHint Documentation](http://jshint.com/docs/)
* [Broccoli JS1](http://broccolijs.com/)

# Grunt Walkthrough

In this walkthrough I am going to show you how to install and set up grunt to use in all of your projects. If you want to follow along I highly suggest you fork, clone, or download the [project repo I will be working from here3](https://github.com/JohnUdacity/grunt-workflow-guide). During this walkthrough I will also include Git Commit hashes that you can use follow along or in case you make mistakes and want to see what went wrong.

Note you can also[*view the commit history here2*](https://github.com/JohnUdacity/grunt-workflow-guide/commits/master)and see what changes have been made

After you get the repository I suggest you open your terminal or git client and checkout commit 6719c2c550c14b0c11cff5d69e45aa23ab0f5188, this is where I will be starting from.

In the repo we have a src folder. The src folder is where we will be doing all of our work, and we will be using grunt to create a dist folder. The dist folder will contain all of our minified, concatenated, and improved code!

## Getting Grunt Setup

Getting setup with grunt is pretty easy but there is a dependency you need to install before you get start. First, you need to make sure you have Node on your system.

1) Install NodeJS

Next we need the grunt command line interface.

2) Install the gruntCLI

npm install -g grunt-cli

Now that we have that setup let's go back to our project, I have some sass code and some javascript files in here that we are going to minify and concat.

First things first we should create a package.json file. A package.json file is a special file that node uses to track dependencies on a project. We can begin creating a package.json file by using the command

npm init

Next we should add grunt as one of our developer dependencies.

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

Using this command will install grunt to a new node\_modules folder as well as add it as a developer dependency in our package.json file. Developer dependencies are dependencies that are only needed for development and are not necesarilly needed for our application to run.

Grunt relies on having a gruntfile.js so lets create one and do our work there.

(This is completed in [this commit1](https://github.com/JohnUdacity/grunt-workflow-guide/commit/cf2ab88d90c8aa9506d554dc0e4e10871ad0e4f8))

## nstalling and configuring a Grunt plugin

Lets use grunt to convert our sass code into usable css. Grunt plugins allow us to keep our grunt files simple and allow us really powerful operations. Grunt has a ton of plugins so be sure to check out the Grunt plugin directory. To install a plugin its as simple as

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

Now if we examine our node\_modules folder we should see that we have a grunt-sass module in there.

Now lets write some code. Pop open our gruntfile.js and first thing, we need to write our grunt wrapper function. This basically just lets node know we are using our grunt module.

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

}

Now we need to load our grunt plugin aka our grunt task

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

grunt.loadNpmTasks('grunt-sass');

}

Now our task is loaded, we're almost ready to run our gruntfile next we need to configure our task

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

grunt.loadNpmTasks('grunt-sass');

grunt.initConfig({

sass: {

dist: {

src: 'src/sass/style.scss',

dest: 'dist/css/style.css'

}

}

});

};

## Running a Grunt task

So now we can go back to our terminal and run the command grunt this will execute our default task and in this case will run our sass task. Now we can see our compiled css in our dist directory.

Let's go through this setup one more time to concatenate our javascript files.

1) Install our concat plugin, in this case grunt-contrib-concat  
2) load the task  
3) configure the task  
4) register the task  
5) run grunt

When we run grunt now we will see that 2 tasks complete. We can run our tasks individually too by using the commands grunt sass or grunt concat.

(You can see the completed code for this step at [this commit](https://github.com/JohnUdacity/grunt-workflow-guide/commit/29b7a1a668da4c880f6da979817b78bdbedc0f61))

## Loading tasks more easily

So you can easily use grunt with this pattern, but if you start using a lot of tasks loading all these tasks is going to take up a lot of lines!

To simplify this we can use the load-grunt-tasks plugin to handle the loading of all our tasks

We first need to install the plugin.

npm **install** *--save-dev load-grunt-tasks*

now we can replace our loadNpmTasks lines with a single command

require('load-grunt-tasks')(grunt);

Yay! Now any plugin we install is loaded and ready to use.

(You can see the changes in [this commit](https://github.com/JohnUdacity/grunt-workflow-guide/commit/a524127d9dea1c5b18e40bfee933eefb74b6dc14))

## Handling directories with a YAML file

Additionally it might be worthwhile to keep a configuration file of our directories. Trust me when I say it's easier to change a directory in one place than it is in 50.

we can create a .yml or .yaml file and put our paths in it.

Gruntconfig.yml

scssDir: src/sass/

jsSrcDir: src/js/

jsConcatDir: dist/js/

cssDir: dist/css/

Now we need to connect our config file with our grunt file. We can do this with the following line

**var** config = grunt.file.readYAML('Gruntconfig.yml');

Now we can use this as our path in case things change

(You can view [this commit here](https://github.com/JohnUdacity/grunt-workflow-guide/commit/225b2e2090b919798366977509569d0e245f9bfc))

## Adding options to our plugins

One common operation we may want to do is lint our code. So let's install a jshint plugin and create a jshit task using our new setup.

npm **install** *--save-dev grunt-contrib-jshint*

Excellent now we just need to define our task in our task configuration and register it.

jshint: {

all: [

'Gruntfile.js',

config.jsSrcDir + "\*.js"

]

}

This will lint both our gruntfile and all of our js files in our source directory!

Now lets run grunt. Our file is linted and error free!

(You can check out the linter setup at this [commit here](https://github.com/JohnUdacity/grunt-workflow-guide/commit/b031819fc4777ccf8bde67ba5846a9c2acd7785f))

Let's take a look and see if we can construct an error though.

If we open up script1.js in our src directory lets change it to

**function** equal(a, b) {

**return** a == b;

}

Now if we want to prohibit the use of == we'll have to add a rule for that. We can add an option property to our JShint task configuration

So if we want to enforce the use of triple equals we can add to our configuration

jshint: {

options: {

"eqeqeq" : true

},

all: [

'Gruntfile.js',

config.jsSrcDir + "\*.js"

]

}

Now if we run our grunt task our linter will return an error!

(You can see our updated code at [this commit](https://github.com/JohnUdacity/grunt-workflow-guide/commit/398c8b86eb1a117ec90fa59e19511eb743d4adb2))

## Watching for Changes and Organizing our code

One of the coolest things we can do with a task runner is set up watch tasks. Watch tasks tell grunt to "watch" files for changes and when a change is detected to execute a specific series of tasks.

To watch tasks we're going to install another plugin into our dependencies

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

Now we have our watch plugin lets setup a watch task that will watch for changes in our sass files and compile them into css when a change is made.

In our grunt config lets add

watch: {

sass: {

files: config.scssDir + "\*\*/\*.scss",

tasks: ["sass"]

}

}

Don't forget to register your new Watch task too!

Now lets run grunt and we can see in our terminal now that grunt didn't just run our code and finish it is "waiting....", lets make a change to our sass file and save and when we go back to our terminal we will see that or sass task executes and our code recompiles.

Let's take out that !important in our sass file and if we switch back to our terminal ta-da! Our sass task ran and our code compiled into our new css without the !important property

Watching will usually break if we have linting errors, but we can also manually stop watching by going to our terminal and using the keyboard shortcut control+C this will kill our task

(You can see how the watch task was [restructured at this commit](https://github.com/JohnUdacity/grunt-workflow-guide/commit/2caabf403edfbd75c93051a9e86b1b5dbd6863e5))

## Additional improvements!

So now we're watching for changes, but it might be worthwhile to modularize our gruntfile a bit. It is easy to get really unorganized when trying to handle JS, CSS, and other things in a single gruntfile. We can separate out and modularize this code a little.

To start we are going to make a new folder called grunt\_tasks. This folder will contain the code that will hold the bulk of our grunt work (hehe pun intended)

in our grunt\_tasks folder we are going to create 2 files one to hold all of our JavaScript related grunt tasks and one to hold all of our Sass related grunt tasks.

I'm going to call them javascript.js and sass.js

So first we are going to get rid of our initConfig setup. Most of this code will be moved into our JavaScript.js and our Sass.js

Instead we are going to use a require statement specifically

require('./grunt\_tasks/sass.js')(grunt, config);

require('./grunt\_tasks/javascript.js')(grunt, config);

These commands basically make sure our gruntfile has access to the contents of these files and makes sure these files have access to the config file we setup.

So what goes inside of javascript.js and sass.js?

Well first we are going to start with a similar wrapper as there is in our main gruntfile except we are going to pass it the config parameter as well

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

};

Next we are going to use grunts merge operation to ensure these files merge properly in our grunt file

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

grunt.config.merge({

});

};

Now we treate the rest of this file exactly as we would our gruntfile!

(You can see our new directory structure at [this commit](https://github.com/JohnUdacity/grunt-workflow-guide/commit/8ff6d929ca52369092aaca483d1623416eaa4b46))

So Javascript.js is going to contain all of our js grunt tasks

javascript.js (I've additionally added a watch task here to watch our js code

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

grunt.config.merge({

concat: {

dist: {

src: config.jsSrcDir + '\*.js',

dest: config.jsConcatDir + 'scripts.js'

},

},

jshint: {

options: {

"eqeqeq": true,

},

all: [

'Gruntfile.js',

config.jsSrcDir + "\*.js"

]

},

watch: {

js: {

files: "<%= concat.dist.src %>",

tasks: [

"jshint",

"concat"

]

}

}

});

};

Sass.js (I've added a csslinter (grunt-contrib-csslint) as well to insure our css is compiled correctly)

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

grunt.config.merge({

sass: {

dist: {

src: config.scssDir + 'style.scss',

dest: config.cssDir + 'style.css'

}

},

csslint: {

strict: {

options: {

important: 2

},

src: "<%= sass.dist.dest %>"

}

},

watch: {

sass: {

files: config.scssDir + "\*\*/\*.scss",

tasks: [

"sass",

"csslint"

]

}

}

});

};

So this is a really awesome way to do things. We keep our code really organized and our gruntfile doesn't become extraordinarily large and difficult to read. If we are more interested in what is happening to our javascript, sass, etc.. we can easily inspect a single shorter file in our grunt\_tasks folder.