

RIVERSIDE JS WORKSHOP, NOVEMBER 2015

- Install Node.js: <https://nodejs.org/en/download/>
- Install Ruby: <http://rubyinstaller.org/> (be sure to add the ruby executable path to environment in Windows)
- Install Sass: `gem install sass` to install Sass.
 - Documentation: <http://sass-lang.com/>
 - Cheat sheet: <http://sass-cheatsheet.brunoscopelliti.com/>
- Install grunt cli

GETTING STARTED WITH GRUNT

Grunt and Grunt plugins are installed and managed via *npm*, the Node.js package manager. Grunt 0.4.x requires stable Node.js versions $\geq 0.8.0$. Odd version numbers of Node.js are considered unstable development versions.

Before setting up Grunt ensure that your npm is up-to-date by running `npm update -g npm` (this might require sudo on certain systems).

INSTALLING THE CLI

Using Grunt 0.3? Please see Grunt 0.3 Notes

In order to get started, you'll want to install Grunt's command line interface (CLI) globally. You may need to use `sudo` (for OSX, *nix, BSD etc) or run your command shell as Administrator (for Windows) to do this.

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

This will put the `grunt` command in your system path, allowing it to be run from any directory.

Note that installing `grunt-cli` does not install the Grunt task runner! The job of the Grunt CLI is simple: run the version of Grunt which has been installed next to a **Gruntfile**. This allows multiple versions of Grunt to be installed on the same machine simultaneously.

HOW THE CLI WORKS

Each time **grunt** is run, it looks for a locally installed Grunt using node's **require()** system. Because of this, you can run **grunt** from any subfolder in your project.

If a locally installed Grunt is found, the CLI loads the local installation of the Grunt library, applies the configuration from your **Gruntfile**, and executes any tasks you've requested for it to run. To really understand what is happening, read the code.

WORKING WITH AN EXISTING GRUNT PROJECT

Assuming that the Grunt CLI has been installed and that the project has already been configured with a **package.json** and a **Gruntfile**, it's very easy to start working with Grunt:

1. Change to the project's root directory.
2. Install project dependencies with **npm install**.
3. Run Grunt with **grunt**.

That's really all there is to it. Installed Grunt tasks can be listed by running **grunt -help**, but it's usually a good idea to start with the project's documentation.

PREPARING A NEW GRUNT PROJECT

A typical setup will involve adding two files to your project: **package.json** and the **Gruntfile**.

package.json: This file is used by npm to store metadata for projects published as npm modules. You will list grunt and the Grunt plugins your project needs as **devDependencies** in this file.

Gruntfile: This file is named **Gruntfile.js** or **Gruntfile.coffee** and is used to configure or define tasks and load Grunt plugins. *When this documentation mentions a **Gruntfile** it is talking about a file, which is either a **Gruntfile.js** or a **Gruntfile.coffee**.*

PACKAGE.JSON

The **package.json** file belongs in the root directory of your project, next to the **Gruntfile**, and should be committed with your project source. Running **npm install** in the same folder as a **package.json** file will install the correct version of each dependency listed therein.

There are a few ways to create a **package.json** file for your project:

Most **grunt-init** templates will automatically create a project-specific **package.json** file. The **npm init** command will create a basic **package.json** file. Start with the example below, and expand as needed, following this specification.

```
{
  "name": "my-project-name",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-connect": "0.5.0",
    "grunt-contrib-jshint": "~0.10.0",
    "grunt-contrib-watch": "~0.4.1",
    "grunt-contrib-uglify": "~0.5.0"
  }
}
```

INSTALLING GRUNT AND GRUNTPLUGINS

The easiest way to add Grunt and gruntplugins to an existing **package.json** is with the command **npm install <module> --save-dev**. Not only will this install **<module>** locally, but it will automatically be added to the **devDependencies** section, using a tilde version range.

For example, this will install the latest version of Grunt in your project folder, adding it to your **devDependencies**:

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

The same can be done for gruntplugins and other node modules. As seen in the following example installing the **JSHint** task module:

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

Checkout the current available gruntplugins to be installed and used on your project at the [plugins page](#). Be sure to commit the updated package.json file with your project when you're done!

THE GRUNTFILE

The **Gruntfile.js** or **Gruntfile.coffee** file is a valid JavaScript or CoffeeScript file that belongs in the root directory of your project, next to the **package.json** file, and should be committed with your project source.

A **Gruntfile** is comprised of the following parts:

- The "wrapper" function
- Project and task configuration
- Loading Grunt plugins and tasks
- Custom tasks

AN EXAMPLE GRUNTFILE

In the following **Gruntfile**, project metadata is imported into the Grunt config from the project's package.json file and the **grunt-contrib-uglify** plugin's **uglify** task is configured to minify a source file and generate a banner comment dynamically using that metadata. When grunt is run on the command line, the **uglify** task will be run by default.

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

  // Project configuration.
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
      options: {
        banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
      },
      build: {
        src: 'src/<%= pkg.name %>.js',
        dest: 'build/<%= pkg.name %>.min.js'
      }
    }
  });

  // Load the plugin that provides the "uglify" task.
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task(s).
  grunt.registerTask('default', ['uglify']);

};
```

Now that you've seen the whole Gruntfile, let's look at its component parts.

THE "WRAPPER" FUNCTION

Every Gruntfile (and gruntplugin) uses this basic format, and all of your Grunt code must be specified inside this function:

```
module.exports = function(grunt) {  
  // Do grunt-related things in here  
};
```

PROJECT AND TASK CONFIGURATION

Most Grunt tasks rely on configuration data defined in an object passed to the `grunt.initConfig` method.

In this example, `grunt.file.readJSON('package.json')` imports the JSON metadata stored in `package.json` into the grunt config. Because `<% %>` template strings may reference any config property, configuration data like file paths and file lists may be specified this way to reduce repetition.

You may store any arbitrary data inside of the configuration object, and as long as it doesn't conflict with properties your tasks require, it will be otherwise ignored. Also, because this is JavaScript, you're not limited to JSON; you may use any valid JS here. You can even programmatically generate the configuration if necessary.

Like most tasks, the `grunt-contrib-uglify` plugin's `uglify` task expects its configuration to be specified in a property of the same name. Here, the `banner` option is specified, along with a single uglify target named `build` that minifies a single source file to a single destination file.

```
// Project configuration.  
grunt.initConfig({  
  pkg: grunt.file.readJSON( 'package.json' ),  
  uglify: {  
    options: {  
      banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'  
    },  
    build: {  
      src: 'src/<%= pkg.name %>.js',  
      dest: 'build/<%= pkg.name %>.min.js'  
    }  
  }  
});
```

LOADING GRUNT PLUGINS AND TASKS

Many commonly used tasks like concatenation, minification and linting are available as grunt plugins. As long as a plugin is specified in package.json as a dependency, and has been installed via npm install, it may be enabled inside your Gruntfile with a simple command:

```
// Load the plugin that provides the "uglify" task.
grunt.loadNpmTasks('grunt-contrib-uglify');
```

Note: the `grunt --help` command will list all available tasks.

CUSTOM TASKS

You can configure Grunt to run one or more tasks by default by defining a default task. In the following example, running grunt at the command line without specifying a task will run the uglify task. This is functionally the same as explicitly running `grunt uglify` or even `grunt default`. Any number of tasks (with or without arguments) may be specified in the array.

```
// Default task(s).
grunt.registerTask('default', ['uglify']);
```

If your project requires tasks not provided by a Grunt plugin, you may define custom tasks right inside the **Gruntfile**. For example, this **Gruntfile** defines a completely custom default task that doesn't even utilize task configuration:

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

  // A very basic default task.
  grunt.registerTask('default', 'Log some stuff.', function() {
    grunt.log.write('Logging some stuff...').ok();
  });

};
```

Custom project-specific tasks don't need to be defined in the **Gruntfile**; they may be defined in **external .js** files and loaded via the `grunt.loadTasks` method.

FURTHER READING

- The [Installing grunt](#) guide has detailed information about installing specific, production or in-development, versions of Grunt and grunt-cli.

- The [Configuring Tasks](#) guide has an in-depth explanation on how to configure tasks, targets, options and files inside the Gruntfile, along with an explanation of templates, globbing patterns and importing external data.
- The [Creating Tasks](#) guide lists the differences between the types of Grunt tasks and shows a number of sample tasks and configurations.
- For more information about writing custom tasks or Grunt plugins, check out the [developer documentation](#).