# Getting started with Grunt.js

Learn how to install and use Grunt.js, a task-runner for JavaScript and Node.js projects.

[Grunt](http://gruntjs.com/) is a command line Javascript task runner utilizing **Node.js** platform. It runs custom defined repetitious tasks and manages process automation. The [project's homepage](http://gruntjs.com/who-uses-grunt) lists many big players in software development that use Grunt in their development as part of continuous integration workflow.

**Use case of Grunt.js**

Let's say you're a front-end developer and you use Sass, some CoffeeScript, need to minimize your stylesheets and scripts (asset compilation), and on top of that you want to see changes in real time in your browser whenever you change a source file. Doing this manually soon becomes tiring as you spend a lot of time on repetitive, menial tasks, AKA grunt work.

A solution to that problem is to automate all menial tasks into automated Grunt tasks. The configuration takes some time to set up, needs some project and asset planning but it will save you a lot of time in the long-run. If you work in a team environment that means you can work with a set of unified commands and everyone will share a common workflow.

Grunt.js uses power of Node.js to work and it runs from your command line. All the configuration is done in regular Javascript (or CoffeeScript).

Grunt's functionality depends on plug-ins and those depend on your needs. There are close to[4,000 Grunt plugins available](http://gruntjs.com/plugins) for different tasks such as:

* Javascript validation,
* test running,
* Sass, Less and CoffeeScript processing,
* concatenation and minification of assets,
* watching for changes,
* live reloading,
* cleaning and copying files and folders,
* image minification,
* auto-prefixing...

Let's first breakdown a typical workflow to get a big picture:

1. Install Node.js and Grunt.
2. Create package.json and list dependencies (Grunt and plugins).
3. Install NPM modules.
4. Create Gruntfile.js.
5. Configure tasks you need to run.
6. Run those tasks in the command line while you work.

**Setting up the environment**

Having [Node.js](http://nodejs.org/download/) is a prerequisite. First you need to install Grunt for the command line as a global module, after which you have grunt command available globally.

sudo npm install -g grunt-cli

In this tutorial we will build this workflow: use [**Sass**](http://sass-lang.com/) to write the stylesheets, **[CoffeeScript](http://coffeescript.org/)** for scripts, introduce asset pipeline (concatenation and minification) and live reload of the page whenever there is a change in source files.

The initial project folder structure should look like this:

project

├── Gruntfile.js

├── index.html

├── package.json

├── scripts

│ ├── hello.coffee

│ └── main.js

└── styles

└── main.scss

mkdir -p project1/{scripts,styles}

cd project1

touch Gruntfile.js index.html package.json scripts/coffee.cofee scripts/main.js styles/main.scss

First we need Grunt for project (which is separate from the grunt-cli used to run gruntcommand). We install it via npm. So we need to fill package.json and list **grunt** as dependency and some plugins which will perform the tasks:

{

"devDependencies": {

"grunt": "^0.4.5", *// Core Grunt module*

"grunt-contrib-concat": "^0.5.0", *// Javascript concatenation*

"grunt-contrib-cssmin": "~0.10.0", *// CSS minification*

"grunt-contrib-sass": "^0.7.3", *// Sass processing*

"grunt-contrib-uglify": "^0.5.1", *// Javascript minification*

"grunt-contrib-watch": "^0.6.1", *// Watch for change in file and reload*

"grunt-contrib-coffee": "~0.12.0" *// CoffeeScript processing*

}

}

Now we run local installation of packages which should put node\_modules directory in the project folder along with the listed modules.

npm install

Note: you can run npm install grunt-some-plugin --save-dev to install the plugin and automatically save it in package.json as a dependency.

**Grunt configuration**

Next step is to create configurations in your Gruntfile.js file and define how the commandgrunt will behave. Every Gruntfile has four distinct regions:

* The "wrapper" function,
* Project and task configuration,
* Loading Grunt plugins and tasks,
* Custom tasks and aliases.

At the start we need to wrap everything with grunt global object:

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

};

Inside the above function we define our configuration for various tasks, load them and create custom tasks like this:

grunt.initConfig({

task: { }

});

grunt.registerTask(taskName, [optional description, ] taskFunction);

grunt.loadNpmTasks('yourplugin');

Note that since grunt.initConfig() is a regular function, you can use any valid Javascript to make configuration programmatic.

**First task - Sass processing**

First write some style rules in your main.scss. After that we can configure Grunt to process the Sass file with one command. To proccess the files we need [grunt-contrib-sass](https://github.com/gruntjs/grunt-contrib-sass) plugin which should be already installed with npm in the project folder.

We first need to configure the task itself inside grunt.initConfig() function (you can check out the [finished Gruntfile here](https://github.com/malizmaj/node-example-app/blob/master/Gruntfile.js)):

sass: {

dev: { *// indicates that it will be used only during development*

files: {

*// destination // source file*

'styles/main.css': 'styles/main.scss'

}

}

}

As you can see we name the task we wish to configure, then we define options, in this case which file to convert. First we define destination and name of the new processed file, then the source file.

You can add comma and define more files following the same format, use an array to hold several sources, or you can use globals and format input like this: styles/\*.scss or \*\*/\*.scss. We can even make variable which will hold the values and use them instead for better programming practice.

Next we need to load the task after the grunt.initConfig() function:

grunt.loadNpmTasks('grunt-contrib-sass');

Now we are ready to go. Fire up your terminal and run the command to convert the file(s) and there should be a regular CSS file waiting for you.

grunt sass

**CoffeeScript processing**

Very similar to how we used the Sass plugin. Inside the grunt.initConfig() function add configuration for the coffee task. Then mark the task to load and run in the command line.

coffee: {

compile: {

files: {

'scripts/hello.js': 'scripts/hello.coffee'

}

}

}

grunt.loadNpmTasks('grunt-contrib-coffee');

grunt coffee

**Asset compilation**

Next we want our styles and scripts to be concatenated and minified (so the server needs to make fewer requests and load the page faster). Again, we configure some tasks and load them. First we define tasks inside grunt.initConfig() function:

cssmin: {

build: {

src: 'styles/main.css',

dest: 'styles/main.min.css'

}

},

concat: {

options: {

separator: '\n/\*next file\*/\n\n' *//this will be put between conc. files*

},

dist: {

src: ['scripts/hello.js', 'scripts/main.js'],

dest: 'scripts/built.js'

}

},

uglify: {

build: {

files: {

'scripts/built.min.js': ['scripts/built.js']

}

}

}

\* **cssmin** is responsible for minification of CSS files, **concat** for Javascript concatenation and**uglify** for Javascript minification.

grunt.loadNpmTasks('grunt-contrib-uglify');

grunt.loadNpmTasks('grunt-contrib-cssmin');

grunt.loadNpmTasks('grunt-contrib-concat');

At this point we can run individual tasks like grunt concat, grunt uglify etc. But since we would always run them one after another, it would be useful to define a custom task that will run a set of commands: one for *sass+cssmin*, and the another for *coffee+concat+uglify*.

**Custom tasks**

We can create our own custom tasks that run when we call them from command line. It is done by creating aliases before or after loadNpmTasks(). We can create what task(s) to run by default when we just run grunt, or custom tasks with custom name (alias) command where we put various tasks we wish to run in an array, in order in which we want them executed.

*// grunt.registerTask(taskName, [optional description, ] taskFunction)*

grunt.registerTask('default', ['sass']);

grunt.registerTask('css', ['sass', 'cssmin']);

grunt.registerTask('js', ['coffee', 'concat', 'uglify']);

grunt

grunt css

grunt js

As we use Javascript, we can create custom functions and programmatically define what the tasks will do. You can check out the [official documentation](http://gruntjs.com/creating-tasks) for more information on how to do this. Some of the things you can do this way include:

* Tasks can be run inside other tasks
* Tasks can be asynchronous
* Tasks can access their own name and arguments.
* Tasks can fail if any errors were logged, or continued if used --force flag
* Tasks can fail if required configuration properties don't exist.
* Tasks can be dependent on the successful execution of other tasks.

**Watching for file changes**

After all the setup so far, it would be nice if those tasks would trigger automatically when we make a change to some files. Just for that kind of job there is a **watch** plugin.

**Watch** task runs a unique set of tasks when the file has been saved. We first need to configurewatch, then run grunt watch which will continue to be ran until you cancel it as it listens for the save trigger.

Configure watch inside grunt.initConfig() as usual and load the task:

watch: {

sass: {

files: '\*\*/\*.scss', *// \*\* any directory; \* any file*

tasks: ['css']

},

coffee: {

files: 'scripts/\*.coffee',

tasks: ['coffee']

},

concat: {

files: ['scripts/hello.js','scripts/main.js'],

tasks: ['concat']

},

uglify: {

files: 'scripts/built.js',

tasks: ['uglify']

}

}

grunt.loadNpmTasks('grunt-contrib-watch');

As you can see we can segregate tasks in subtasks that are called only when we need them. From the example above, we can call grunt watch:sass and it will only evaluate files that end in*.scss* anywhere in the project folder and if we change them, css task will run. If we just rungrunt watch all the subtasks will be evaluated concurrently.

You can now try making some change to main.scss file and upon save you should see automatic conversion to main.css. Same with CoffeeScript files.

Note the order at which tasks run, especially when you have multiple files. Be careful not to fall into infinite-loop trap when one change triggers another and another... Also note that grunt watch will run until you cancel the process, and with that in mind it should be put as the last task in the array.

**Live reload**

If you make a change to some source file, you can see that change reflected live in your browser because **watch** comes with **live-reload** option available. It is made possible by harnessing **Node.js** and it's back-end power.

First make modification in the Gruntfile.js to the whole watch task inside grunt.initConfig():

watch: {

sass: {

files: '\*\*/\*.scss',

tasks: ['css'],

options: {

livereload: 35729 *// 35729 is the default port === true*

}

},

coffee: {

files: 'scripts/\*.coffee',

tasks: ['coffee']

},

concat: {

files: ['scripts/hello.js','scripts/main.js'],

tasks: ['concat']

},

uglify: {

files: 'scripts/built.js',

tasks: ['uglify'],

options: {

livereload: **true**

}

},

all: {

files: ['\*\*/\*.html'],

options: {

livereload: **true**

}

}

},

Next add <script src="http://localhost:35729/livereload.js"></script> to the index.html (or other files as needed) before closing body tag.

Open the index.html file and run

grunt watch

As port 35729 is the default port for the live-reload, you can go to http://localhost:35729/ to see if it works. You should get a welcome json file.

Now try making some changes to the index.html and watch as the browser reflects that change. Same works for scss and coffee changes if everything is configured correctly.

You can learn more about live reloading and options on these repository homepages:

* [**watch** plugin](https://github.com/gruntjs/grunt-contrib-watch).
* [**connect** plugin](https://github.com/gruntjs/grunt-contrib-connect).

**Project sample**

You can head to this [sample GitHub project using Grunt.js](https://github.com/malizmaj/node-example-app) to see concrete example of Grunt, or[here to see just the Gruntfile](https://github.com/malizmaj/node-example-app/blob/master/Gruntfile.js).

You can use this serie of commands to quickly set it up on your machine (assuming you have*git*, *node.js*, and *grunt-cli* installed):

git clone https://github.com/malizmaj/node-example-app.git

cd node-example-app

npm install

grunt

After the last command, grunt watch will be running in the background. Now try openingindex.html page, changing styles/main.scss file and saving. Try something simple, like the*$bgcolor*. You should see the page automagically refresh with that new background color. You can also try the same with scripts/hello.coffee.