# Getting Started with Gulp.js

Learn how to set up and use Gulp.js, a task runner for Node.js based on Node streams.

**Introduction**

[Gulp](http://gulpjs.com/) is a command line task runner utilizing **Node.js** platform. It runs custom defined repetitious tasks and manages process automation.

What makes Gulp different from other task runners is that it uses Node streams, piping output from one task as an input to the next. It reads a file once, processes it through multiple tasks, and then writes the output file. This results in faster builds because there is no need to create and read intermediary files on hard drive.

The goal of this tutorial is to introduce main concepts of Gulp and see it in action. We will make a quick project to demonstrate some of the most frequent front-end development tasks you'll encounter, and create an integrated workflow.

You will learn:

* How to set up an automated workflow,
* Do Sass and CoffeeScript processing,
* Do asset compilation (concatenation and minification),
* Watch for change in your files and act on it,
* Automatically reload the page in the browser after change.

By the end of the tutorial you will be able to apply Gulp to your own project, customize it and be more efficient.

A brief overview of the steps to your first task:

1. Install Node.js and Gulp.
2. Create package.json and list dependencies (Gulp and plugins).
3. Install NPM modules.
4. Create gulpfile.js.
5. Load plugin and create tasks.
6. Run those tasks in the command line.

**Prerequisites**

Gulp is a command line tool, so you should be familiar working in terminal. In order to use Gulp you need to have [Node.js](http://nodejs.org/) installed on your system.

We will use [Sass](https://semaphoreci.com/community/tutorials/sass-lang.com) and [CoffeeScript](http://coffeescript.org/) processing, but having deep knowledge of them isn't required.

**Setting up the Environment**

Before we can create and run our first task, we need to have something to work on and apply Gulp to.

First we will define structure of our project, and what we want from Gulp. After that we will install Gulp and start working.

To follow along you should clone [this repository](https://github.com/malizmaj/gulp-project.git) so you can reuse the project files that we will rely on for the rest of the tutorial.

git clone https://github.com/malizmaj/gulp-project.git

If you are interested in the finished code, checkout the finished branch.

**Setting up a New Project**

Our project has the following structure:

gulp-project

├── assets

├── gulpfile.js

├── index.html

├── package.json

├── scripts

│   ├── hello.cofee

│   └── main.js

└── styles

└── main.scss

The goal is be to create an automated workflow for working with Sass and CoffeeScript. We want to make tasks that will:

* Process styles/main.scss to assets/main.css, and scripts/hello.cofee to scripts/hello.js,
* Concatenate and minify scripts/main.js and scripts/hello.js to assets/script.js,
* React when we change scripts/hello.cofee, styles/main.scss, scripts/main.js, andindex.html,
* Create server and load index.html so we can see the change on each save

**Gulp Installation**

To use Gulp, you need to install it as a global module first throught NPM (if you're new to NPM, you can get to know it better in our [Getting Started with NPM tutorial](https://semaphoreci.com/community/tutorials/LINK.TO.GETTING-STARTED-WITH-NPM)):

sudo npm install --global gulp

Now we need to download Gulp and its plugins to our project. We will specify the plugins we're about to use in package.json:

{

**"devDependencies"**: {

**"gulp"**: "latest",

**"gulp-util"**: "latest",

**"gulp-sass"**: "latest",

**"gulp-coffee"**: "latest",

**"gulp-uglify"**: "latest",

**"gulp-concat"**: "latest",

**"gulp-connect"**: "latest"

}

}

We list them in devDependencies because you'll typically run Gulp during the development stage.

Now install the dependencies by running:

npm install

This creates node\_modules directory with all the plugins in the root of your project.

**Writing the First Gulp Task**

All Gulp configuration goes in gulpfile.js in the root of your project. A pattern for writing tasks is that you first load a plugin you're about to use and then define a task which is based on that plugin.

Gulp is very plugin driven. If you want to accompish something, you need to know which plugin to use. Usually a single plugin has a single purpose, and all the plugins are just regular Node.js code.

First we load the plugins using Node.js and its require function:

**var** gulp = require('gulp');

Now we can write our first task. A basic form of a task looks like this:

gulp.task('name', **function**() {

*//implementation of the task*

});

We call gulp object and its task method. It takes two arguments: name of the task, and a function literal (or an array of other tasks, which you will see later).

We will create a simple task to get familiar with basic methods. We will take index.html and copy it to the folder assets. The code for that is:

gulp.task('copy', **function**() {

gulp.src('index.html')

.pipe(gulp.dest('assets'))

});

Most important methods of gulp object are:

* src where we put the name of the file we want to work with and use as an input,
* pipe will take output of the previous command as pipe it as an input for the next,
* dest writes the output of previous commands to the folder we specify.

To run the task open the terminal, navigate to the root of the project and run gulp command and task name as a parameter, like this:

gulp copy

You should now have a copy of index.html in your assets folder.

You can also pass arrays, or use [globs](http://en.wikipedia.org/wiki/Glob_%28programming%29) with src and dest:

* folder/\*.html - will match all the HTML files in folder
* root/\*\*/\*.html - will match all the HTML files in all the folders from root to its children

**Using the First Plugin**

Gulp relies on the plugins to do almost everything, so we need to learn how to use them.

Now we will use a plug-in called gulp-util. Its purpose is to log custom messages to the terminal.

First we need to load the plug-in:

**var** gutil = require('gulp-util');

After that we write task as in the previous case:

gulp.task('log', **function**() {

gutil.log('== My Log Task ==')

});

Here we called plug-in we defined as gutil, and called its log method with our custom message. Now we can run log task in the terminal:

$ gulp log

[19:41:37] Using gulpfile ~/gulp-project/gulpfile.js

[19:41:37] Starting 'log'...

[19:41:37] == My First Task ==

[19:41:37] Finished 'log' after 763 μs

You can [learn more about gulp-util here](https://www.npmjs.com/package/gulp-util).

**Using Preprocessors**

Now we will take care of the Sass and CoffeeScript files so that the main page can use them.

**Sass**

Now we will create a task to process the styles/main.scss which can be then used in our HTML file.

To do this just follow the established pattern: load the plug-in and create the task:

**var** sass = require('gulp-sass');

gulp.task('sass', **function**() {

gulp.src('styles/main.scss')

.pipe(sass({style: 'expanded'}))

.on('error', gutil.log)

.pipe(gulp.dest('assets'))

});

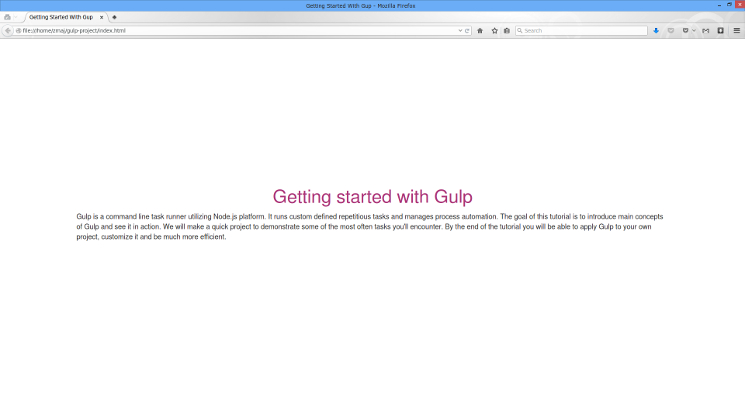
Here we piped main.scss file to sass object and as an argument we passed an object with options we want to use. Visit [official Sass reference](http://sass-lang.com/documentation/file.SASS_REFERENCE.html) for more information about output styles, and [gulp-sass plug-in page](https://www.npmjs.com/package/gulp-sass) for some other options.

We also handled a possible error message and used gulp-util to log it.

Now we can run our sass task:

gulp sass

We should now have assets/main.css waiting for us. You can open index.html to see the result.



**CoffeeScript**

To process CoffeeScript file we will do the same as previously:

**var** coffee = require('gulp-coffee');

gulp.task('coffee', **function**() {

gulp.src('scripts/hello.coffee')

.pipe(coffee({bare: **true**})

.on('error', gutil.log))

.pipe(gulp.dest('scripts'))

});

[gulp-coffee plug-in](https://www.npmjs.com/package/gulp-coffee) supports the same options as the standard compiler, so you can reference the [official CoffeeScript website for more information](http://coffeescript.org/).

Now run the coffee task and there should be a new scripts/hello.js file.

gulp coffee

**Asset Compilation**

Asset compilation is an act of minifying and concatenating scripts together so that the server loads the page faster.

In this task we will minify all the JavaScript files using [gulp-uglify plug-in](https://www.npmjs.com/package/gulp-uglify), and then merge them using [gulp-concat](https://www.npmjs.com/package/gulp-concat).

**var** uglify = require('gulp-uglify'),

concat = require('gulp-concat');

gulp.task('js', **function**() {

gulp.src('sripts/\*.js')

.pipe(uglify())

.pipe(concat('script.js'))

.pipe(gulp.dest('assets'))

});

uglify works fine with default options, and concat as an argument takes a name for our newly combined file.

Now call the task and asset/script.js will be ready for index.html to use.

gulp js

Note that coffee and js tasks are closely related. We can combine them in a new task that will do both tasks in the order we specify. That task first needs to execute coffee so we have scripts to compile. We can even make it a default task - a task that runs when we run just gulpin the terminal.

gulp.task('default', ['coffee', 'js']);

We combined two tasks in an array, in the order we want them executed. Now run default task and both tasks will execute one after the other:

gulp

**Watching for File Changes**

Our next goal is to automatically do all the processing tasks when a change happens in the code. We accomplish this with watch method of the gulp object; it comes as a standard part of the Gulp so there is no need for loading new module.

watch method takes as arguments: source to be watched, and a task to be triggered after change. So we can define task like this:

gulp.task('watch', **function**() {

gulp.watch('scripts/hello.coffee', ['coffee']);

gulp.watch('scripts/\*.js', ['js']);

gulp.watch('styles/main.scss', ['sass']);

});

When you run the watch task it will listen for a change and it will keep running until stopped. You can try running it now:

gulp watch

Now try making a change to the styles/main.scss and saving it; sass task will execute andwatch will still keep running.

**Creating Server for Live Reload**

Our final task is to see a change we make in the code reflected in the browser without manual reload. After that we can integrate all the tasks into an efficient workflow.

First we will load gulp-connect and create a new task:

**var** connect = require('gulp-connect');

gulp.task('connect', **function**() {

connect.server({

root: '.',

livereload: **true**

})

});

We created a task which will launch a server with some options: we set root of our server to the root of the project, and value of livereload to true. You can visit [gulp-connect plug-in page](https://www.npmjs.com/package/gulp-connect) for more information.

Now run the task:

gulp connect

Navigate to localhost:8080 in your browser and you should see index.html open, as the server runs in the background.

**Complete Integration**

To be able to see the changes we make to source files, we need to reload the server after there is a change. It means that we need to establish a relation between watch and connect, and we need to pipe the end of some tasks to connect.reload().

The final gulpfile.js looks like this:

**var** gulp = require('gulp'),

gutil = require('gulp-util'),

sass = require('gulp-sass'),

coffee = require('gulp-coffee'),

connect = require('gulp-connect'),

uglify = require('gulp-uglify'),

concat = require('gulp-concat');

**var** coffeeSources = ['scripts/hello.coffee'],

jsSources = ['scripts/\*.js'],

sassSources = ['styles/\*.scss'],

htmlSources = ['\*\*/\*.html'],

outputDir = 'assets';

gulp.task('log', **function**() {

gutil.log('== My First Task ==')

});

gulp.task('copy', **function**() {

gulp.src('index.html')

.pipe(gulp.dest(outputDir))

});

gulp.task('sass', **function**() {

gulp.src(sassSources)

.pipe(sass({style: 'expanded'}))

.on('error', gutil.log)

.pipe(gulp.dest('assets'))

.pipe(connect.reload())

});

gulp.task('coffee', **function**() {

gulp.src(coffeeSources)

.pipe(coffee({bare: **true**})

.on('error', gutil.log))

.pipe(gulp.dest('scripts'))

});

gulp.task('js', **function**() {

gulp.src(jsSources)

.pipe(uglify())

.pipe(concat('script.js'))

.pipe(gulp.dest(outputDir))

.pipe(connect.reload())

});

gulp.task('watch', **function**() {

gulp.watch(coffeeSources, ['coffee']);

gulp.watch(jsSources, ['js']);

gulp.watch(sassSources, ['sass']);

gulp.watch(htmlSources, ['html']);

});

gulp.task('connect', **function**() {

connect.server({

root: '.',

livereload: **true**

})

});

gulp.task('html', **function**() {

gulp.src(htmlSources)

.pipe(connect.reload())

});

gulp.task('default', ['html', 'coffee', 'js', 'sass', 'connect', 'watch']);

Now to explain all the changes:

1. We did some refactoring and created variables to hold file paths.
2. Created a new html task so we can watch for the changes in HTML files which we then pipe to reload(), and made the task part of the watch task.
3. Added .pipe(connect.reload()) at the end of js and sass tasks so that the server reloads every time they execute.
4. Made a new default task that will first do all the processing, create a server and keep a watch

Now we have an integrated workflow. To try it out run:

gulp

Visit localhost:8080. Now you can try making some change to styles/main.scss, hello.coffee orindex.html, and save it. In the terminal you should see that a task had run and the browser reflected that change.

**Final Word**

Gulp can be used to automatize a great number of menial tasks that are common during development. You should be able to use the final gulpfile.js as a basis for your project, with some slight modification, and you can find a plug-in for your particular need.

What makes Gulp different from [Grunt](http://gruntjs.com/) for example, is the approach of piping of input and output which may result in faster execution depending on your code, and preference of code over convention, which can make configuration files easier to read and understand. Grunt on the other hand is currently more mature, has a larger community, and more plugins. You can learn more about Grunt in our [Getting Started with Grunt tutorial](https://semaphoreci.com/community/tutorials/LINK.TO.GRUNT.TUTORIAL). As time goes on, the performance of both tools and number of plugins will probably level out so the choice will come down to personal preference.

You can visit these links to:

* [Browse through the available Gulp plugins](http://gulpjs.com/plugins/),
* [Consult Gulp API](https://github.com/gulpjs/gulp/blob/master/docs/API.md),
* [Learn more about Node.js streams](https://github.com/substack/stream-handbook).