Gulp

14.02.2017

—

Mohnish.R.Sharma and Sonal Bajage.
Iksula
Mumbai

Overview

Gulp is a toolkit that helps you automate painful or time-consuming tasks in your development workflow. It's often used to do front end tasks like: .

1.Reloading the browser automatically whenever a file is saved.

2.Using preprocessors like Sass or LESS.

3.Optimizing assets like CSS, JavaScript, and images.

This is not a comprehensive list of things Gulp can do, Visit http://gulpjs.com/plugins/ to checkout more.

Why Gulp?

Tools like Gulp are often referred to as "build tools" because they are tools for running the tasks for building a website. The two most popular build tools out there right now are Gulp and Grunt.

The main difference is how you configure a workflow with them. Gulp configurations tend to be much shorter and simpler when compared with Grunt. Gulp also tends to run faster.

Installing Gulp

You need to have Node.js (Node) installed onto your computer before you can install Gulp

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

If you do not have Node installed already, you can get it by downloading the package installer from Node's website. Click here

When you're done with installing Node, you can install Gulp by using the following command in the command line:

”npm install gulp -g”

Only Mac users need the sudo keyword

```
$ sudo npm install gulp -g
```

The npm install command we've used here is a command that uses Node Package Manager (npm) to install Gulp onto your computer.

The -g flag in this command tells npm to install Gulp globally onto your computer, which allows you to use the gulp command anywhere on your system.

Now that you have Gulp installed, let's make a project that uses Gulp.

Creating a Gulp Project

I. Project Folder

First, we'll create a folder called project (project name) to server as our project root.

II. Run the npm init

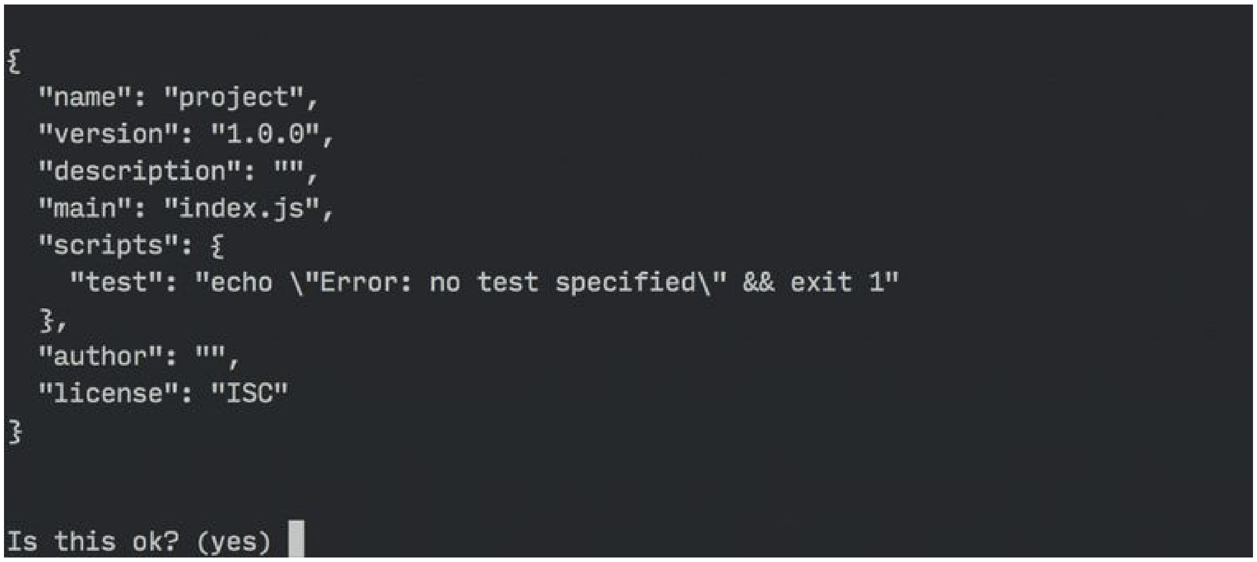
Run the npm init command from inside that directory

.

```
# ... from within our project folder
$ npm init
```

The npm init command creates a package.json file for your project which stores information about the project

npm init will prompt you:



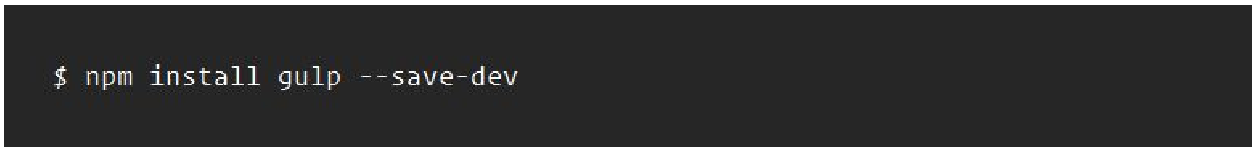
```
{
  "name": "project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this ok? (yes)
```

Once the package.json file is created, we can install Gulp into the project by using the following command:

III. Install Gulp Into The Project

npm install gulp --save-dev



```
$ npm install gulp --save-dev
```

This time, we're installing Gulp into project instead of installing it globally

We've added --save-dev, which tells the computer to add gulp as a dev dependency in package.json

If you check the project folder when the command has finished executing, you should see that Gulp has created a node_modules folder. You should also see a gulp folder within node_modules.

IV. Determining Folder Structure

Gulp is flexible enough to work with any folder structure. You'll just have to understand the inner workings before tweaking it for your project.

For this document, we will use the structure of a generic webapp

```
|- app/
    |- css/
    |- fonts/
    |- images/
    |- index.html
    |- js/
    |- scss/
|- dist/
|- gulpfile.js
|- node_modules/
|- package.json
```

In this structure, we'll use the app folder for development purposes, while the dist (as in "distribution") folder is used to contain optimized files for the production site.

Since app is used for development purposes, all our code will be placed in app.

We'll have to keep this folder structure in mind when we work on our Gulp configurations. Now, let's begin by creating your first Gulp task in gulpfile.js, which stores all Gulp configurations.

Writing Your First Gulp Task

In gulpfile.js file

The first step to using Gulp is to require it in the gulpfile.

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

The require statement tells Node to look into the node_modules folder for a package named gulp. Once the package is found, we assign its contents to the variable gulp.

We can now begin to write a gulp task with this gulp variable.

The basic syntax of a gulp task is:

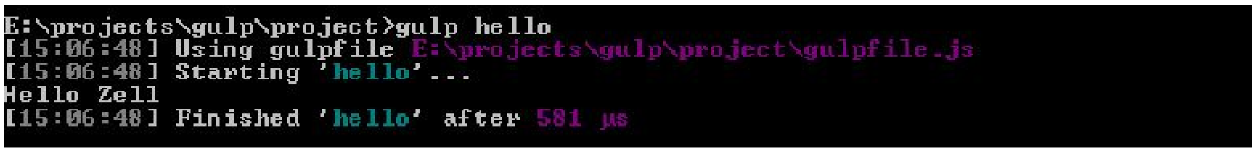
```
gulp.task('task-name', function() {
  // Stuff here
});
```

To test it out, let's create a hello task that says Hello Zell!

```
gulp.task('hello', function() {
  console.log('Hello Zell');
});
```

We can run this task with gulp hello in the command line

```
$ gulp hello
```



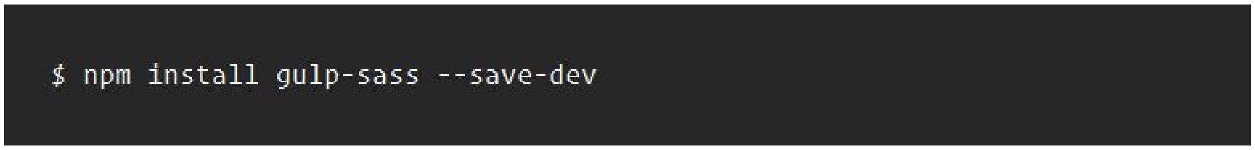
```
E:\projects\gulp\project>gulp hello
[15:06:48] Using gulpfile E:\projects\gulp\project\gulpfile.js
[15:06:48] Starting 'hello'...
Hello Zell
[15:06:48] Finished 'hello' after 581 μs
```

Preprocessing with Gulp

We can compile Sass to CSS in Gulp with the help of a plugin called gulp-sass.

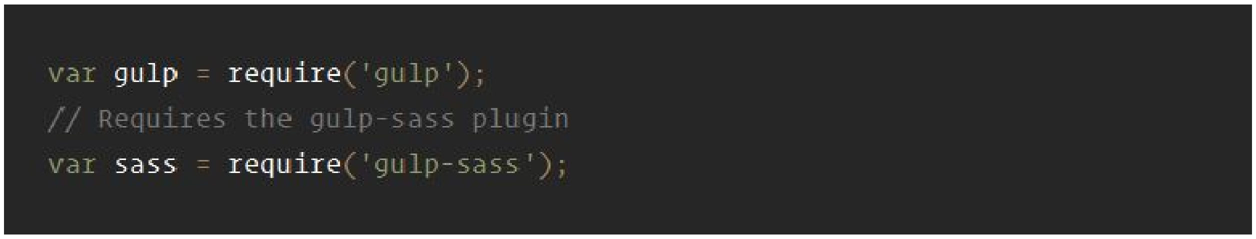
You can install gulp-sass into your project by using the npm install command like we did for gulp

We'd also want to use the --save-dev flag to ensure that gulp-sass gets added to devDependencies in package.json.

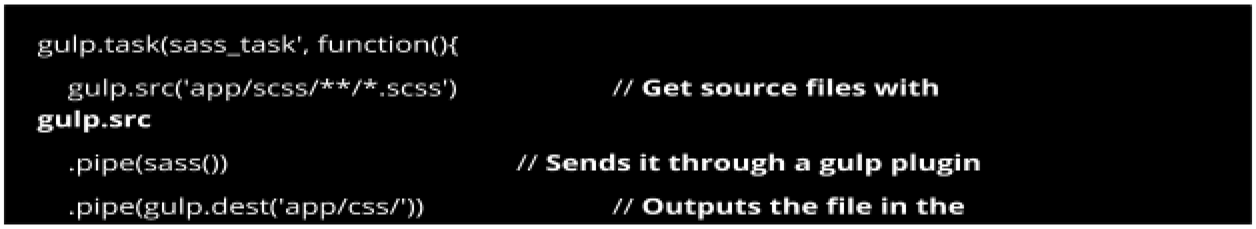


```
$ npm install gulp-sass --save-dev
```

We have to require gulp-sass from the node_modules folder just like we did with gulp before we can use the plugin.



```
var gulp = require('gulp');
// Requires the gulp-sass plugin
var sass = require('gulp-sass');
```



```
gulp.task(sass_task', function(){
  gulp.src('app/scss/**/*.scss')                    // Get source files with
gulp.src
  .pipe(sass())                               // Sends it through a gulp plugin
  .pipe(gulp.dest('app/css/'))                      // Outputs the file in the
```

gulp.task defines your tasks. Its arguments are name and function.

gulp.src points to the files we want to use. It uses .pipe for chaining it's output into other plugins.

gulp.dest points to the output folder we want to write files.

Watch

Gulp provides us with a watch method that checks to see if a file was saved.

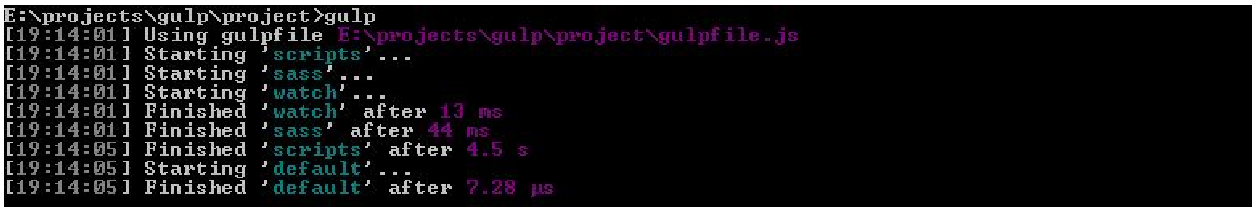
```
gulp.watch('files-to-watch' ,['task','to','run']);
```

```
gulp.task('watch',function(){
        gulp.watch('app/scss/**/*.scss' ,['sass_t']);
});
```

```
gulp.task('default', ['sass_', 'watch']);
```

We can run this default task in the command line

```
gulp
```



```
E:\projects\gulp\project>gulp
[19:14:01] Using gulpfile E:\projects\gulp\project\gulpfile.js
[19:14:01] Starting 'scripts'...
[19:14:01] Starting 'sass'...
[19:14:01] Starting 'watch'...
[19:14:01] Finished 'watch' after 13 ms
[19:14:01] Finished 'sass' after 44 ms
[19:14:05] Finished 'scripts' after 4.5 s
[19:14:05] Starting 'default'...
[19:14:05] Finished 'default' after 7.28 μs
```

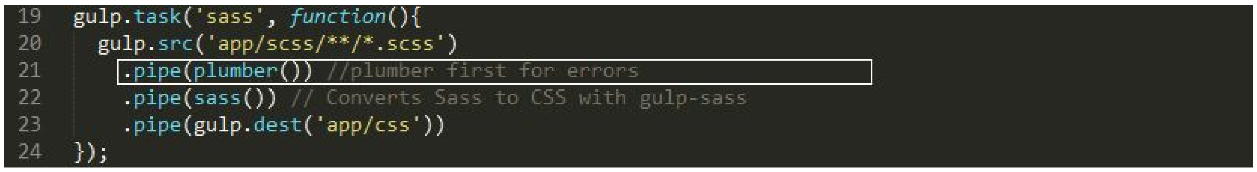
If there is a syntax error in your scss file, gulp would stop.

Once you rectify the error you have to run the gulp command again.

To prevent gulp from stopping we can use gulp-plumber plugin.

```
npm install gulp-plumber --save-dev
```

```
var plumber = require('gulp-plumber')
```



```
gulp.task('sass', function(){
  gulp.src('app/scss/**/*.scss')
    .pipe(plumber()) //plumber first for errors
    .pipe(sass()) // Converts Sass to CSS with gulp-sass
    .pipe(gulp.dest('app/css'))
});
```

Wrapping it up

We've gone through the absolute basics of Gulp and created a workflow that's able to compile Sass into CSS while watching HTML and JS files for changes at the same time. We can run this task with the gulp command in the command line.

There's a lot more to Gulp and workflows that we can explore to make this process even better. Here are some ideas for you:

1. Using Autoprefixer to write vendor-free CSS code
2. Gulp livereload auto reloads the browser when scss file is changed
3. Gulp image optimiser to reduce the size of images

And many more Click Here to find more.