



scotch

LOGIN

SIGN UP

Automate Your Tasks Easily with Gulp.js

Justin Rexroad (@justinrexroad)

January 26, 2015



Justin Rexroad

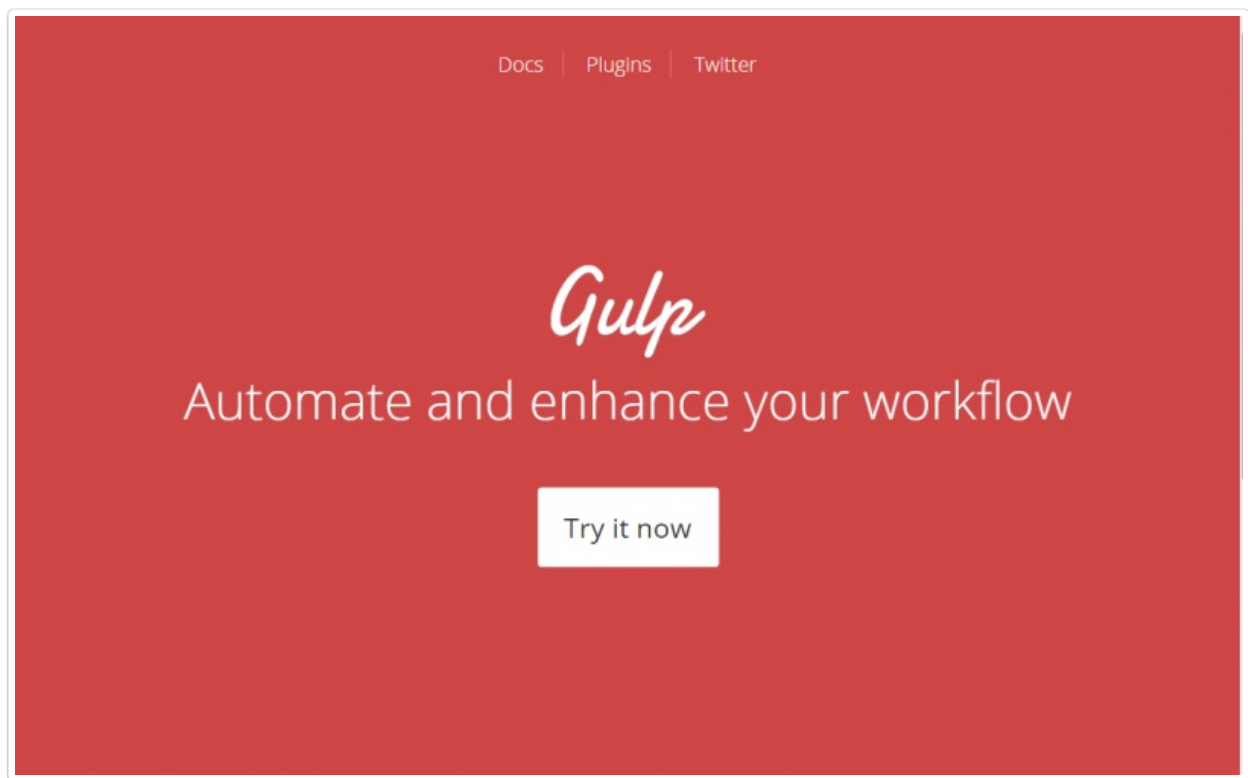


AUTOMATE YOUR TASKS EASILY WITH GULP.JS

Code

As developers we often need to look at the tools we use and decide if we are using the right tool for the job. Chris did an awesome write up on [Grunt](#) early last year. But maybe Grunt just isn't right for you.

[Gulp](#) is a streaming build system, by using node's streams file manipulation is all done in memory, and a file isn't written until you tell it to do so.



Much like Grunt, Gulp is a javascript task runner. Gulp however prefers code over configuration. Being that your tasks are written in code, gulp feels more like a build framework, giving you the tools to create tasks that fit your specific needs.

Installation

Gulp is easy to get installed and running. The steps are:

1. Install Gulp Globally
2. Install Gulp In devDependencies
3. Create a `gulpfile.js`

The first step is to get gulp installed globally.

```
$ npm install --global gulp
```

After that, you'll need gulp as a `devDependencies` on any of your projects you want to use it in. Make sure that you have your `package.json` created by manually creating it or typing `npm init`. Once you have your `package.json`, let's install gulp into `devDependencies` with:

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

And finally you'll need a `gulpfile.js` in your project root that contains your tasks. As an intermediary step we'll add the gulp utilities plugin so we have a runnable task that visibly shows it executed.

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

In the `gulpfile.js` file that you just created, we'll make a simple gulpfile that just logs that gulp is running.

```
/* File: gulpfile.js */

// grab our gulp packages
var gulp = require('gulp'),
    gutil = require('gulp-util');

// create a default task and just log a message
gulp.task('default', function() {
  return gutil.log('Gulp is running!')
});
```

And if everything went as expected running `gulp` in your command line should give you output similar to.

```
> gulp
[12:32:08] Using gulpfile ~/Projects/gulp-scotch-io/gulpfile.js
[12:32:08] Starting 'default'...
[12:32:08] Gulp is running!
[12:32:08] Finished 'default' after 1 ms
```

Overview

Directory Structure for this Tutorial

We should probably take a second to define our project's structure. For this simple demo we'll use the following structure, you can leave the files blank for now.

```
public/  
  | assets/  
  | | stylesheets/  
  | | | style.css  
  | | javascript/  
  | | | vendor/  
  | | | | jquery.min.js  
  | | | bundle.js  
source/  
  | javascript/  
  | | courage.js  
  | | wisdom.js  
  | | power.js  
  | scss/  
  | | styles.scss  
  | | grid.scss  
gulpfile.js  
packages.json
```

`source` is the folder where we will do our work. `assets/style.css` will be created by gulp when we process and combine our SASS files in `source/scss`. The `bundle.js` file will be created by gulp when we minify and combine all our JS files.

A brief overview of gulp

Gulp is a streaming build system. It's streaming nature is what allows it to pipe and pass around the data being manipulated or used by it's plugins. The plugins are intended to only do one job each, so it's not uncommon to pass a singular file through multiple plugins.

The gulp api is incredibly light containing **4 top level functions**. They are

- `gulp.task`
- `gulp.src`
- `gulp.dest`
- `gulp.watch`

`gulp.task` defines your tasks. Its arguments are name, deps and fn.

Where name is a string, deps is an array of task names, and fn is the function that performs your task. Deps is optional so gulp.task in it's two forms are:

```

gulp.task('mytask', function() {
  //do stuff
});

gulp.task('dependenttask', ['mytask'], function() {
  //do stuff after 'mytask' is done.
});

```

`gulp.src` points to the files we want to use. Its parameters are globs and an optional options object. It uses `.pipe` for chaining its output into other plugins.

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

`gulp.src` and `gulp.dest` used to simply copy files looks like:

```

gulp.task('copyHtml', function() {
  // copy any html files in source/ to public/
  gulp.src('source/*.html').pipe(gulp.dest('public'));
});

```

`gulp.watch` like `gulp.task` has two main forms. Both of which return an EventEmitter that emits `change` events. The first of which takes a glob, an optional options object, and an array of tasks as its parameters.

```

gulp.watch('source/javascript/**/*.js', ['jshint']);

```

Simply put, when any of the files matched by the glob change, run the tasks. In the above code block, when any files in the `source/javascript` subfolders that have an extension of `.js` change, then the task `jshint` will be run against those files.

The second form takes the glob, an optional options object, and an optional callback that will run when a change is picked up.

You can compare this to grunt, which requires a secondary package to have the watch feature. Gulp has it built right in.

For more information refer to the [api docs](#).

Tasks that are actually useful.

Being able to tell us that it is running is a fine task, but lets get gulp to do some real tasks for us.

We'll start with simple tasks and work our way up.

Jshint on save

Our first task will lint our javascript (check for errors) using [jshint](#) and we'll also set it up to run this task each time we save a javascript file.

To begin we'll need the [gulp-jshint](#) package, grab it with npm. We'll also need a reporter for jshint to make the output nicely formatted and color coded; we'll grab that too.

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

Now we'll add the lint task to our gulpfile.

```
/* File: gulpfile.js */

// grab our packages
var gulp = require('gulp'),
    jshint = require('gulp-jshint');

// define the default task and add the watch task to it
gulp.task('default', ['watch']);

// configure the jshint task
gulp.task('jshint', function() {
  return gulp.src('source/javascript/**/*.js')
    .pipe(jshint())
    .pipe(jshint.reporter('jshint-stylish'));
});

// configure which files to watch and what tasks to use on file changes
gulp.task('watch', function() {
  gulp.watch('source/javascript/**/*.js', ['jshint']);
});
```

So lets step through what we've done.

We've rewritten our default task to have the watch task as dependency. What this means is that running

```
$ gulp
```

will run the watch task.

Now lets look at the new jshint task. It sources any `.js` files that exist in `source/javascript` or any of it's subdirectories. So a file at `source/javascript/carousel/main.js` would be picked up for the task just as well. These files are then passed into our gulp-jshint plugin, which then passes it into the stylish reporter to give us the jshint results.

We can run this task by doing:

```
$ gulp jshint
```

Super easy!

Alright, now what about that watch task. It's simple actually, if a change is detected in any of our javascript files, it runs the jshint task.

Sass Compilation with libsass

[Sass](#) serves as a way to extend CSS giving support for variables, nested rules, mixins, inline imports, and more.

Ken Wheeler has already done an awesome write up on Sass that you can find [here](#).

For sass compilation we'll use [gulp-sass](#)

NOTE: gulp-sass uses node-sass which in turn uses libsass. On windows you'll need to install python 2.7.x and Visual Studio Express 2013 in order to compile libsass. Mac and Linux will use whatever gcc is available.

An alternative is to use gulp-ruby-sass, which uses ruby and the sass gem instead.

```

/* file: gulpfile.js */

var gulp    = require('gulp'),
    jshint   = require('gulp-jshint'),
    sass     = require('gulp-sass');

/* jshint task would be here */

gulp.task('build-css', function() {
  return gulp.src('source/scss/**/*.scss')
    .pipe(sass())
    .pipe(gulp.dest('public/assets/stylesheets'));
});

/* updated watch task to include sass */

gulp.task('watch', function() {
  gulp.watch('source/javascript/**/*.js', ['jshint']);
  gulp.watch('source/scss/**/*.scss', ['build-css']);
});

```

We can also add sourcemaps using [gulp-sourcemaps](#) . If you've never used sourcemaps they're an awesome feature that map processed, minified , or other modified files to their original sources.

A list of the plugins that support gulp-sourcemaps can be found [here](#).

```

/* file: gulpfile.js */
var gulp      = require('gulp'),
    jshint     = require('gulp-jshint'),
    sass      = require('gulp-sass'),
    sourcemaps = require('gulp-sourcemaps');

gulp.task('build-css', function() {
  return gulp.src('source/scss/**/*.scss')
    .pipe(sourcemaps.init()) // Process the original sources
    .pipe(sass())
    .pipe(sourcemaps.write()) // Add the map to modified source.
    .pipe(gulp.dest('public/assets/stylesheets'));
});

```

Javascript concat and minify

When working with a lot of javascript, you usually get to a point where you need to pull it all together. The general purpose plugin gulp-concat allows you to accomplish that easily.

We can also go a step further and run it through uglify also to get a much smaller filesize.

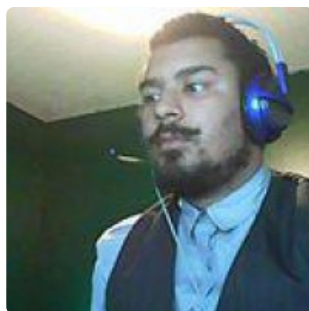
Additionally, we'll conditionally apply uglify based on whether we're building for production.

```
gulp.task('build-js', function() {  
  return gulp.src('source/javascript/**/*.js')  
    .pipe(sourcemaps.init())  
    .pipe(concat('bundle.js'))  
    //only uglify if gulp is ran with '--type production'  
    .pipe(gutil.env.type === 'production' ? uglify() : gutil.noop())  
    .pipe(sourcemaps.write())  
    .pipe(gulp.dest('public/assets/javascript'));  
});
```

Summary

We've only scratched the surface of gulp. Gulp can be as complex or as simple as you need it to be, and because it's just code you can do just about anything you want as a task.

From as simple as concating together javascript files, to automatically deploying to an S3 bucket on save. Gulp gives you the tools to do what you want quickly and easily.



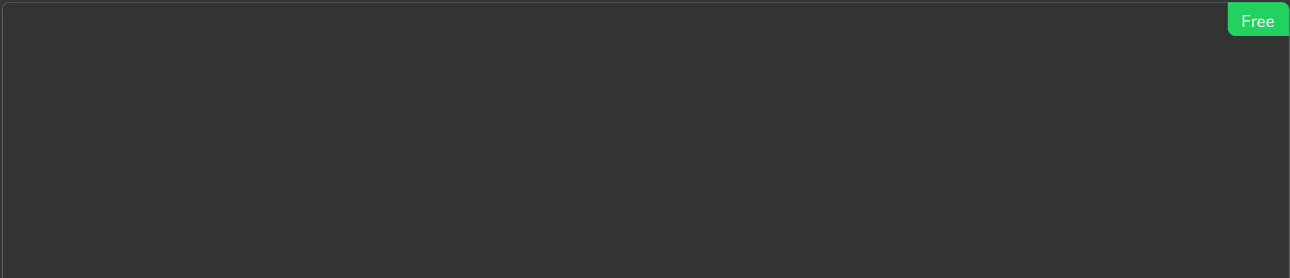
Justin Rexroad



Latest Video Courses

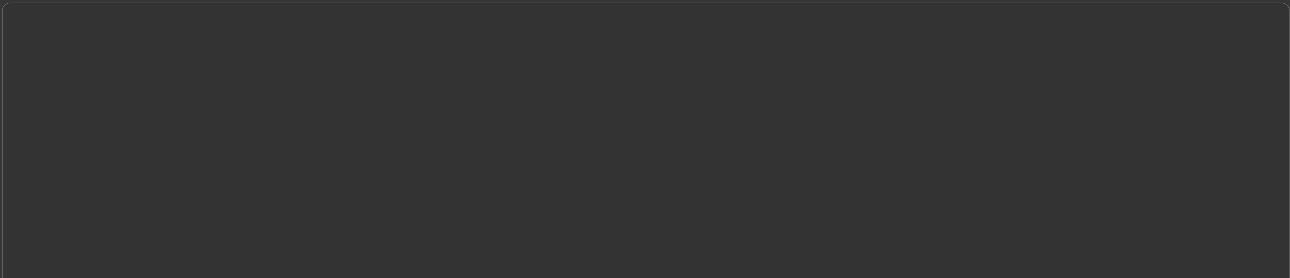
See More Courses 

Free



2.1 hours

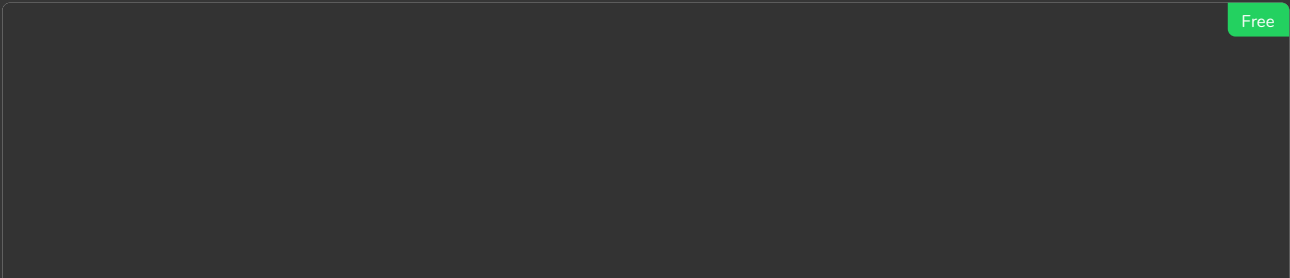
Getting Started with Angular v2+ (works with v4)



1.4 hours

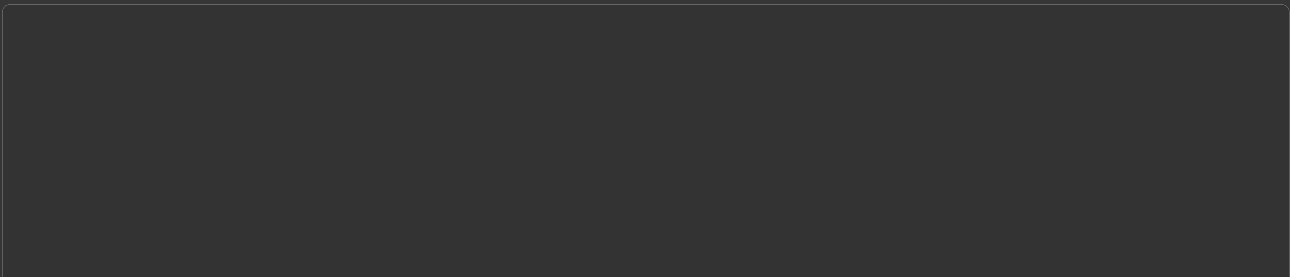
Getting Started with React

Free



4.7 hours

Getting Started with JavaScript for Web Development



0.9 hours

Get to Know Git

Join Scotch

High Quality Content

The best tutorials and content that you'll find for web development. Guides, courses, tutorials, and more great content to learn with.

Build Real Apps

We won't just go over concepts and "Hello Worlds"; we'll **build real apps together** that you can use at your job or for your portfolio.

Not Just How, But Why

There are many different ways to code the same project. We'll show **best practices** and why certain choices are better than others.

Scotch Free

Like your favorite posts

Bookmark content for reference

Post in the forums

Free

Scotch Premium

All of the free features

Access to **all premium content**

Downloadable videos

Access to **live chat**

No ads across all of Scotch

Track **completed** content

\$20



scotch

Top shelf learning. Informative tutorials explaining the code **and the choices behind it all.**



BROUGHT TO YOU BY...

Chris Sevilleja

Nick Cerminara

[FAQ](#)
[Privacy](#)
[Terms](#)
[Rules](#)
[Affiliates](#)

2017 © Scotch.io, LLC. All Rights Super Duper Reserved.

Proudly hosted by Digital Ocean