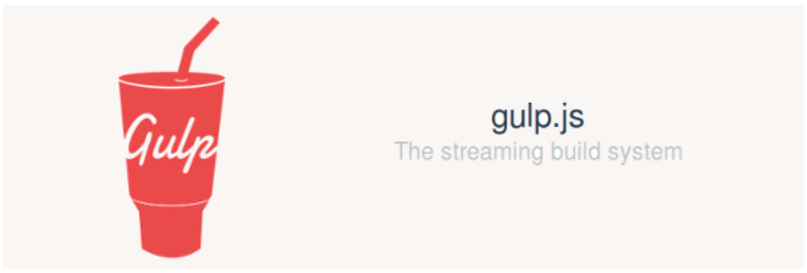
JUSTIN MCCANDLESS

A Tutorial for Getting Started with Gulp



Awhile back I wrote a **tutorial covering the basics of Grunt (http://www.justinmccandless.com/blog/A+Tutorial+for+Getting+Started+with+Grunt)**. Now that a new frontend build tool has entered the scene, I thought I'd take a look and see if I could write a simple guide for **Gulp (http://gulpjs.com/)** at the same level. This guide will take you from installation and getting set up through examples of real world build processes while laying out all of the tools out there to help you on the way.

Resources

This article aims to be thorough, but you'll also be referring to a few other resources while working with Gulp.

The **Gulp Github repo (https://github.com/gulpjs/gulp/)** is the main place to go for help. The README has a nice sample gulpfile, and the repo also contains **Gulp's documentation (https://github.com/gulpjs/gulp/tree/master/docs)**. The **getting started doc (https://github.com/gulpjs/gulp/blob/master/docs/getting-started.md)** shows you how to install and create a blank gulpfile.

The **Gulp website (http://gulpjs.com/)** mainly just refers to the Github repo, but it does contain an invaluable **plugin directory (http://gulpjs.com/plugins/)** as well.

Why a Frontend Build Tool?

Sure you can just upload your raw development files to a web server and call it a day, but there are a lot of tools out there that can make a developer's life a lot easier, from **CoffeeScript (http://coffeescript.org/)** to **jshint (http://www.jshint.com/)** to **Jasmine (http://pivotal.github.io/jasmine/)** testing. If you're going to utilize even a very simple build process, a tool like Gulp can make things much more automated.

Gulp is a command line tool that you run at various points in your build process?—?or even continuously. If your build process previously consisted of compiling **Sass (http://sass-lang.com/)** and minifying your JavaScript, Gulp could handle both of those for you at once, and it's much easier to configure than a bunch of bash commands. Even more importantly, it's portable thanks to npm, so you can build on any machine you check out your repo to after a simple `npm install`.

Gulp and Grunt

The main difference you'll want to keep in mind between **Grunt (http://gruntjs.com/)** and Gulp is that Grunt is controlled through configuration and Gulp is controlled through code. You tell Grunt what to do through a json file, but you create your Gulp build process by writing JavaScript and invoking Gulp's API.

Besides that, you've got two pretty similar tools. Both have the goal of automating your frontend build process. Both are written mainly in Node and set up with a variety of plugins using npm. And both have a friendly root level file to tell them what to do (Gruntfile.js and gulpfile.js).

Installation

The documentation contained in the **Gulp Github repo (https://github.com/gulpjs/gulp)** for getting up and running is actually quite effective, so check out the **official getting started guide (https://github.com/gulpjs/gulp/blob/master/docs/getting-started.md)** referenced above. You'll just be installing Gulp globally and then locally to your project using npm, so make sure you have **Node (http://nodejs.org/)** installed. Just two simple commands here:

```
npm install -g gulp
npm install --save-dev gulp
```

Hello World

You can continue with the getting started guide to create a minimal gulpfile. Here's what my idea of a super basic one might look like:

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

gulp.task('default', function() {
 console.log('Hello world.');
});
```

gistfile1.js view raw (https://gist.github.com/justinmc/9179795/raw/54490fd6b3d2c45cc6e5786d1724b4a1236e98ea/gistfile1.js) (https://gist.github.com/justinmc/9179795#file-gistfile1-js) hosted with ♡ by **GitHub (https://github.com)**

You could now jump back to the command line and run this with the command `gulp` . As you can probably tell, the only thing this does is output 'Hello world. '

Let's take a look at what 's going on in this gulpfile. The first thing you might notice is that this is just JavaScript, it's not a json config file like with Grunt. It's using Node's require function to include `gulp` , then defining a task.

Tasks are the same concept from Grunt: you can define arbitrary tasks and run them from the command line with `gulp` . `default` is the task called when no parameter is given. In Gulp though, tasks are just functions, and you can place whatever code you want inside of them.

A Useful Gulpfile - Copying Files

Now to a gulpfile that does something useful. This is might be a setup you have for a simple **Yeoman (http://yeoman.io/)** style app/ working directory and dist/ production directory. All it does is copy the javascript and html from the app/ directory to the dist/ directory.

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

var paths = {
 scripts: ['app/scripts/**/*.js'],
 html: ['app/index.html', '!app/test.html'],
 dist: 'dist/'
};

gulp.task('default', function(){
 gulp.src(paths.scripts.concat(paths.html))
 .pipe(gulp.dest(paths.dist));
});
```

gistfile1.js view raw (https://gist.github.com/justinmc/9179801/raw/cf6a7d30e964ba0c27ae4c1785f074378c896dff/gistfile1.js) (https://gist.github.com/justinmc/9179801#file-gistfile1-js) hosted with ♡ by GitHub (https://github.com)

Let's look at that `paths` object we're defining. That's not something specific to Gulp - remember, we're just writing JavaScript here. We can define any sort of variable we like and use it however. Just notice the actual path strings I'm setting. They work just like in Grunt, so `**` is a wildcard meaning this and any subdirectory, and `*` is a typical wildcard for the filename. Prepending a path with an exclamation mark tells Gulp to exclude that directory.

In Gulp, you define the instructions for each task in a function passed as an argument to the task. `gulp.src` takes an array of source paths. `gulp.dest` copies results to a given directory. And `pipe` chains tasks together.

Note how I'm concatenating the two paths arrays we defined in `paths` . Again, you might be used to being stuck in the confines of Grunt's configuration specifics, but here we're just passing a parameter to a function. `gulp.src` accepts an array of paths, so we can pass that however we want.

Try this gulpfile out if you want, and you'll see the specified .html and .js files in the app/ directory will be copied into the new dist/ directory.

A Real Gulpfile

Let's take a look at one more example, this time something you might find in a real app. The main idea for this gulpfile is similar to our above example; we're still building development files from an app/ folder into a production dist/ folder, we're just doing a little bit of processing in between.

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

var clean = require('gulp-clean');
var jshint = require('gulp-jshint');
var concat = require('gulp-concat');
var uglify = require('gulp-uglify');
var imagemin = require('gulp-imagemin');

var bases = {
 app: 'app/',
 dist: 'dist/',
};

var paths = {
 scripts: ['scripts/**/*.js', '!scripts/libs/**/*.js'],
 libs: ['scripts/libs/jquery/dist/jquery.js', 'scripts/libs/underscore/underscore.js', 'scripts/backbone/backbone.
 styles: ['styles/**/*.css'],
 html: ['index.html', '404.html'],
 images: ['images/**/*.png'],
 extras: ['crossdomain.xml', 'humans.txt', 'manifest.appcache', 'robots.txt', 'favicon.ico'],
};

// Delete the dist directory
gulp.task('clean', function() {
 return gulp.src(bases.dist)
 .pipe(clean());
});

// Process scripts and concatenate them into one output file
gulp.task('scripts', ['clean'], function() {
 gulp.src(paths.scripts, {cwd: bases.app})
 .pipe(jshint())
 .pipe(jshint.reporter('default'))
 .pipe(uglify())
 .pipe(concat('app.min.js'))
 .pipe(gulp.dest(bases.dist + 'scripts/'));
});

// Imagemin images and ouput them in dist
gulp.task('imagemin', ['clean'], function() {
 gulp.src(paths.images, {cwd: bases.app})
 .pipe(imagemin())
 .pipe(gulp.dest(bases.dist + 'images/'));
});

// Copy all other files to dist directly
gulp.task('copy', ['clean'], function() {
 // Copy html
 gulp.src(paths.html, {cwd: bases.app})
 .pipe(gulp.dest(bases.dist));

 // Copy styles
 gulp.src(paths.styles, {cwd: bases.app})
 .pipe(gulp.dest(bases.dist + 'styles'));

 // Copy lib scripts, maintaining the original directory structure
 gulp.src(paths.libs, {cwd: 'app/**'})
 .pipe(gulp.dest(bases.dist));

 // Copy extra html5bp files
 gulp.src(paths.extras, {cwd: bases.app})
 .pipe(gulp.dest(bases.dist));
});

// A development task to run anytime a file changes
gulp.task('watch', function() {
 gulp.watch('app/**/*', ['scripts', 'copy']);
});

// Define the default task as a sequence of the above tasks
gulp.task('default', ['clean', 'scripts', 'imagemin', 'copy']);
```

gistfile1.js (https://gist.github.com/justinmc/9149719#file-gistfile1-js) view raw (https://gist.github.com/justinmc/9149719/raw/cd827f522a1ece064aac07612d0401ef0d77f071/gistfile1.js) hosted with ♡ by GitHub (https://github.com)

We're using five new plugins this time around: gulp-clean, gulp-jshint, gulp-concat, gulp-uglify, and gulp-imagemin. You can see the full list of plugins in the plugin directory on the Gulp site. Just like in Grunt, when you use a new plugin in Gulp, you'll have to install it with npm. So to install these plugins and write them to your package.json, use something like this:

npm install --save-dev gulp-clean gulp-jshint gulp-concat gulp-uglify gulp-imagemin

Notice the default task definition down towards the bottom of this gulpfile. You can define a task as a sequence of other tasks by passing an array of task names instead of a function. Here, the default task runs `clean`, `scripts`, `imagemin`, and `copy`, so let's take a close look at each of these.

gulp-clean (https://github.com/peter-vilja/gulp-clean)

This plugin will simply delete files for you, allowing you to get rid of old or transitional files. The first thing this gulpfile does is run gulp-clean on the production dist/ directory, just to make sure that nothing created in a previous build and later removed in app/ could be lingering around.

scripts

The scripts task is going to lint and uglify all of the app's JavaScript, concatenate it all into one file, and output it into our production directory.

gulp-jshint (https://github.com/wearefractal/gulp-jshint)

`jshint` is a great tool for catching errors and bad habits in your code, and this plugin just runs it on the given files. Note that we also need to use a reporter here if we want to see jshint's output on the command line. `jshint.reporter('default')` is fine if you're not picky about what it shows you, but go ahead and look at the **Github repo (https://github.com/wearefractal/gulp-jshint)** if you want to get more specific. And don 't forget that this plugin will be looking in the root of your project for a `.jshintrc` file to get your configuration preferences.

gulp-uglify (https://github.com/terinjokes/gulp-uglify)

This plugin will minify your code by removing extra whitespace and other stuff that only matter to a human reader, not the browser. Pipe in the files you want it to operate on like normal.

gulp-concat (https://github.com/wearefractal/gulp-concat)

This will concatenate all given files into one file to reduce the number of HTTP requests required to get your code. Just pass in the filename that you want it output as. Note that you should still add a `gulp.dest` to the end of this to actually output the file into a given directory like normal.

gulp-imagemin (https://github.com/sindresorhus/gulp-imagemin)

This plugin reduces the weight of your images. Here it's used with `gulp.src` and `gulp.dest` like normal to output the final files to the production folder.

Multiple Instructions in One Task

The copy task is very basic, just like in the previous example. Notice that this time we're creating multiple sets of instructions in one task, copying independent `gulp.src` files to independent `gulp.dest` destinations within the task function. This is perfectly OK in Gulp. Once again, it 's just JavaScript.

cwd Option

Note the cwd parameter I'm passing to gulp.src in most tasks. The basic usage of this does what you might expect, changing the working directory of the task so you don't have to repeat redundant subdirectories on all of your paths.

It can also be used to tell Gulp to preserve your directory structure when outputting files, however. If you use the directory wildcard in your cwd like `app/**` in the libs portion of the `copy` task, Gulp will compare the directories it finds there with the paths provided in the base path in the first parameter, and use this to reconstruct your directory structure in the output. If this is omitted, all of your output files will be flattened into a single directory level.

watch

The last thing to notice in this gulpfile is the `watch` task. This watches the given path for any files to change, and executes the array of tasks when they do. This is great for when you're in the middle of development. It's a pain to switch to the terminal and run `gulp` every time you make a small change, so set up a watch task like this to save you the trouble. It's not included in the `default` task here, so keep in mind you would run it with `gulp watch` from the command line.

Tasks in Parallel and Series

A big caveat to watch out for when using Gulp is that **all tasks are executed in parallel by default**. This is very useful, but can cause problems if you're not careful.

In the gulpfile above, our clean task must complete before any of our tasks that output files into the dist/ folder (scripts, imagemin, and copy). We indicate that by passing `['clean']` in as the second parameter to the definitions of these tasks. This parameter is an array of any tasks that need to complete before the task being defined is run, so you can include as many as you want.

Just make sure that your task that you want to complete first returns the Gulp stream (which is the result of a Gulp operation) when it completes, as done above in the clean task. This tells Gulp that it has completed. You can also take a callback as the parameter to your task and call it when the task completes, in case you aren't generating a stream. The **documentation explains this in detail (https://github.com/gulpjs/gulp/blob/master/docs/recipes/running-tasks-in-series.md)**

From Here

You should be able to set up the build process for a sizeable app with this guide alone, but I hope you don't stop there with learning Gulp. Don't forget to refer to the **Gulp plugin directory (http://gulpjs.com/plugins/)** to get ideas on how you can use Gulp to automate much more than is covered in this article. There are already hundreds of plugins available, but Gulp is still lagging far behind Grunt in this area. If you're feeling really adventurous, check out the **documentation on creating your own plugin! (https://github.com/gulpjs/gulp/blob/master/docs/writing-a-plugin/README.md)**

Update 2014.04.13 I added the above clarification about tasks in parallel and series due to running into problems with this in my own Gulp projects.

FEBRUARY 23, 2014

SHARE: (HTTP://TWITTER.COM/SHARE?
TEXT=A%20TUTORIAL%20FOR%20GETTING%20STARTED%20WITH%20GULP&URL=HTTPS%3A%2F%2FJU
TUTORIAL-FOR-GETTING-STARTED-WITH-GULP%2F) (HTTPS://WWW.FACEBOOK.COM/
U=HTTPS%3A%2F%2FJUSTINMCCANDLESS.COM%2FPOST%2FA-TUTORIAL-FOR-GETTING-S
(HTTPS://PLUS.GOOGLE.COM/SHARE?URL=HTTPS%3A%2F%2FJUSTINMCCANDLESS.COM%2FPOST%2F
WITH-GULP%2F)

About the author

I'm Justin McCandless, a software engineer with Autodesk's Life Sciences group in San Francisco. In my free time I'm working on building the most solid and straight forward way to create diagrams at **popvex.com (http://www.popvex.com)**.

3 Comments The Blog of Justin McCandless 1 Login

Recommend Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Me • a year ago

Simple, concise and informative. Thanks bro

1 • Reply • Share ›

Kim Richardson • a year ago

I always wanted to use gulp for my static sites, this made is so easy to understand. Thank you

1 • Reply • Share ›

Ouyang Chao • a year ago

Great post, many thanks.

1 • Reply • Share ›