FREE eBook

LEARNING gulp

Free unaffiliated eBook created from
Stack Overflow contributors.

#gulp

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gulp

It is an unofficial and free gulp ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official gulp.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with gulp

Remarks

Gulp is a JavaScript build system, node.js-based task runner like Grunt. It allows you to automate common tasks during your development process. Gulp uses the streams-concept and code-over-configuration for a simpler and more intuitive build process. The code-over-configuration concept allows to create more readable and simple tasks, whereas grunt tasks are highly over-configured.

Versions

Version	Release date
3.4	2014-01-17
3.7	2014-06-01
3.8	2014-06-10
3.9	2015-06-02
3.9.1	2016-02-09
4.0.0	2016-06-21

Examples

Installation or Setup

1. Install Node.js and NPM:

Gulp requires Node.js and NPM, Node's package manager. Most installers include NPM with Node.js. Refer to the installation documentation or confirm it is already installed by running the following command in your terminal,

```
npm -v
// will return NPM version or error saying command not found
```

2. Install gulp globally:

If you have previously installed a version of gulp globally, please run `npm rm --global gulp` to make sure your old version doesn't collide with **gulp-cli**.

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

3. Initialize your project directory:

```
$ npm init
```

4. Install gulp in your project devDependencies:

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

5. Create a gulpfile.js at the root of your project:

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

gulp.task('default', function() {
  // place code for your default task here
});
```

6. Run gulp:

```
$ gulp
```

The default task will run and do nothing.

To run individual tasks, use `gulp <task> <othertask>`.

Task dependency

You can run tasks in series, by passing a second parameter to `gulp.task()`.

This parameters is an array of tasks to be executed and completed before your task will run:

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

gulp.task('one', function() {
   // compile sass css
});

gulp.task('two', function() {
   // compile coffeescript
});

// task three will execute only after tasks one and two have been completed
// note that task one and two run in parallel and order is not guaranteed
gulp.task('three', ['one', 'two'], function() {
   // concat css and js
});
```

```
// task four will execute only after task three is completed
gulp.task('four', ['three'], function() {
   // save bundle to dist folder
});
```

You can also omit the function if you only want to run a bundle of dependency tasks:

```
gulp.task('build', ['array', 'of', 'task', 'names']);
```

Note: The tasks will run in parallel (all at once), so don't assume that the tasks will start/finish in order. Starting gulp v4, `gulp.series()` should be used if the order of execution of dependency tasks is important.

Concat js file in sub folder using gulp

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

// include plug-ins
var uglify = require('gulp-uglify'),
    concat = require('gulp-concat');

// Minified file
gulp.task('packjsMin', function() {
   return gulp.src('node_modules/angular/*.js')
      .pipe(concat('script.js'))
      .pipe(uglify())
      .pipe(gulp.dest('Scripts'));
});

//Not minified file
gulp.task('packjs', function () {
    return gulp.src('node_modules/angular/*.js')
      .pipe(concat('allPackages.js'))
      .pipe(gulp.dest('Scripts'));
});
```

gulp CLI docs

Flags

gulp has very few flags to know about. All other flags are for tasks to use if needed.

- `-v` or `--version` will display the global and local gulp versions
- `--require <module path>` will require a module before running the gulpfile. This is useful for transpilers but also has other applications. You can use multiple `--require` flags
- `--gulpfile <gulpfile path>` will manually set path of gulpfile. Useful if you have multiple gulpfiles. This will set the CWD to the gulpfile directory as well
- `--cwd <dir path>` will manually set the CWD. The search for the gulpfile, as well as the relativity of all requires will be from here
- `-T` or `--tasks` will display the task dependency tree for the loaded gulpfile
- `--tasks-simple` will display a plaintext list of tasks for the loaded gulpfile
- `--color` will force gulp and gulp plugins to display colors even when no color support is

detected
- `--no-color` will force gulp and gulp plugins to not display colors even when color support is detected
- `--silent` will disable all gulp logging

The CLI adds process.env.INIT_CWD which is the original cwd it was launched from.

Task specific flags

Refer to this StackOverflow link for how to add task specific flags

Tasks

Tasks can be executed by running `gulp <task> <othertask>`. Just running `gulp` will execute the task you registered called `default`. If there is no `default` task gulp will error.

Compilers

You can find a list of supported languages at interpret. If you would like to add support for a new language send pull request/open issues there.

Read Getting started with gulp online: https://riptutorial.com/gulp/topic/1341/getting-started-with-gulp

Chapter 2: Comprehensive Guide to a Front end Workflow with Gulpjs 2 of 2

Remarks

Cool. So we are all done with our workflow automation.

We now have a gulp file , that

- Responsifies and minifies images
- cleans,autoprefixes,concatenates and minifies Css
- Concatenates and minifies JS
- Watches for changes to assets be it HTML | CSS | JS and triggers associated tasks
- Creates a build directory , and stores all processed deployment ready code inside it. And all that , in the background, while you just develop your app.

References

Run-Sequence Browser-sync

Examples

Setting up Browser sync And Configuring Watchers for Style and Script

NOTE

> **This page illustrates use of gulp plugins like browser-sync , gulp-watch and run-sequence , and continues discussing gulp-workflow-automation from where we left off at Gulpjs-workflow-automation-1 of 2. In case you landed here , consider** going through that post first.

- Default Task
- Watchdog task - to continually build your deployment ready assets on the fly , whenever anything image | JS | css changes on the course of development.

Let us begin with browser-sync.

Gulp Watchdog Task

Let us begin with the watchdog task.

The goal , is to watch for changes you make while developing. Any change , should trigger the

corresponding gulp task.

Also , we need a functionality that syncs your changes on the browser.

Browser sync

So , we need to install Browser Sync.

```
bash $ npm install browser-sync --save-dev
```

With that premise, let us open our gulpfile.js and add the watch functionality. Let us require browser sync and define some variables to use its functionality.

At the top of the gulpfile , add the below snippet. Place it just below the image compression declarations.

like so:

```
//Browser-sync

var sync = require('browser-sync').create();
var reload = sync.reload;
```

Having browser sync sync your development on to the browser , is a simple configuration. Let us create a task called watchdog.

like so:

```
$.task('watchdog', function() {


})
```

Now , If we browse through browser sync options here , and search for the server setting , we can see how easy it is .

We just need to place the below inside of our watchdog task

Snippet - 1 - inside watchdog boilerplate

```
/*
 Initiate Browser sync
 @documentation - https://www.browsersync.io/docs/options/
 */
 sync.init({
  server: {
    baseDir: "./"
 },
 port: 8000 //change it as required
 });
```

Insert the above inside of your watchdog boilerplate above.

The next snippet , is to define a watcher for styles, with a goal to reprocess changed css files or new ones , and trigger a browser reload automatically.

snippet - 2 - inside watchdog boilerplate

```
$.watch(['css/**/*', 'fonts/google/**/*.css'], reload).on('change', function(event) {
console.log(event.type + ':' + event.path)
if (event.type === 'deleted') {
uncache('styles', event.path);
$$.remember.forget('auto-prefixed-stylesheets', event.path);
}
sequence('optimizeStyles')
});
```

Insert the above inside of your watchdog boilerplate above.

So we are monitoring" `[fonts/google/**/*.css , /**/*.css ]`" i.e,

all css files under css/ all css files under fonts/google/ When anything changes, or a new file is added , it triggers the reload method, which is defined at the top of our gulpfile , in the browsersync declaration.

> Note : You might notice , that we have a **.on** event handler attached to the watcher.

```
$.watch(['css/**/*', 'fonts/google/**/*.css'], reload).on('change', function(event)
```

Basically , anything CUD(Create| Update | Delete) triggers the reload function , and passes an event Object as a parameter to the callback function.

The callback is a vital function , where we can achieve operations like uncache on asset deletion.Now the event object has parameters like

- path
- type - Create/Update/Delete

If an asset is deleted , we need to make sure the caches we built in our earlier minification functions , via gulp-cached and gulp-remember , need updation.

we are handling that in the snippet below , which is inside the callback on change.

```
if (event.type === 'deleted') {
uncache('styles', event.path);
$$.remember.forget('auto-prefixed-stylesheets', event.path);
}
```

Note

$ - >alias for gulp

$$ - > alias for gulp-load-plugins

you might also notice , that I have a `sequence('optimizeStyles');` after I wrote the uncache invocation

The sequence method , ensures , synchronous method runs in an asynchronous by default javascript.

installing it is simple

DO

```
bash $ npm install run-sequence
```

then , declare it in your gulpfile just below the browser sync declaration.

```
var sequence = require('run-sequence');
```

So with that understanding , the watcher for scripts is an easy one to write. just different globs!

So, add this snippet below the style watcher inside the watchdog boilerplate.

Snippet - 3 - inside Watchdog task boilerplate

```
/*
on addition or change or deletion of a file in the watched directories
the change event is triggered. An event object with properties like
path,
event-type
is available for perusal passed to the callback

*/
$.watch('js/**/*', reload).on('change', function(event) {
console.log(event.type + ':' + event.path)
if (event.type === 'deleted') {
uncache('scripts', event.path);
$$.remember.forget('linted-scripts', event.path);
}
sequence('optimizeScripts');
});
```

Note

We used two functions in our snippets above.

- uncache

- $$.remember.forget Note:

$-> Alias for gulp

$$-> Alias for gulp-load-plugins

Let us define the function uncache somewhere in our gulpfile.js , before it is invoked.

```
/*
Deletes a cache entry
*/
var uncache = function(cacheName, cacheKey) {
        var cache = $$.cached;
        if (cache.caches[cacheName] && cache.caches[cacheName][cacheKey])
            return delete cache.caches[cacheName][cacheKey];
        return false;
    }
    /*
    logs current cache created via gulp-cached
    */
var viewCache = function() {
    console.log($$.cached.caches)
}
```

Defining a Default Task

So now, let us finish the gulpfile code , by defining a Default task.

the default task is the one that runs, when you just say

```
gulp
```

on a command prompt under the root of your project.

```
$.task('default', ['generateResponsiveImages'], function() {
 $.start('watchdog');
 console.log('Starting Incremental Build');
});
```

Read Comprehensive Guide to a Front end Workflow with Gulpjs 2 of 2 online: https://riptutorial.com/gulp/topic/6343/comprehensive-guide-to-a-front-end-workflow-with-gulpjs-2-of-2

Chapter 3: Comprehensive Guide to a Front-end Workflow Automation with Gulpjs -1 of 2

Syntax

- npm install [plugin-name] --save-dev
- npm install [plugin-name] --save
- Function `<function-name>` (glob) { $.src(glob).pipe([plugin 1]).pipe([plugin 2])....pipe([plugin n]).pipe($.dest(`<destination-name>`) }
- `$.task(<taskname> , [dependencies] , <body>);`

Remarks

Continue Reading - Creating a default task and setting up browser-sync

References

- Gulp-Clean-Css
- Gulp-Uglify
- Gulp-Autoprefixer - for Css Prefixes that maintain browser compatibility
- Gulp-Cached
- Gulp-Remember
- Gulp-Jshint

Step-By-Step Guide to Gulp Workflow Automation for absolute beginners that I documented iny my blog

Examples

Loading All The Plugins from Package.JSON

Assuming , you have a grasp of how to install gulp, let us dive right down to requiring all the gulp-dependencies from package.json under your projects root folder.

In case you do not have a gulpfile yet , please create an empty file with the name

gulpfile.js

First , we require gulp. like so:

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

Next up , let us load all our plugins, into our gulpfile , by the below snippet

Note

Please read through the comments in all of the snippets we will be including in this read, they provide more insight into every functionality

```
/*
require gulp-load-plugins, and call it
gulp-load-plugins will require individually,
all the plugins installed in package.json at the root directory
these required plugins, are lazy loaded
that is, gulp-load-plugins will only require them when they are referenced in this file
plugins are available for perusal, via camelcased names, minus gulp
eg: gulp-clean-css is accessed by $$.cleanCss
*/
var $$ = require('gulp-load-plugins')();
```

NOTE

Throughout the many examples we will have in the article , we alias

1. gulp as $ and
2. gulp-load-plugins as $$

purely for ease of typing.

Installing Plugins for Responsive images|Css Minification|Js minification

Our goal , is to

1. Make your Images conform to widths and scale appropriately, by generating a battery of images of varied widths, and to minify them
2. Lint your Javascript
3. Minimize your assets - JS/CSS/HTML , thus enabling you to host a lighter than lightest code
4. Watch the css/js/image files for change , and rebuild optimizations
5. Synching your changes while in development, on to a browser serving your site.

We need a number of plugins, so let us install all of them. Please run all these in the project's root folder.

Image processing plugins

```
bash $ npm install --save-dev gulp-responsive
bash $ npm install --save-dev gulp-imagemin
bash $ npm install --save-dev imagemin
bash $ npm install --save-dev imagemin-jpeg-recompress
bash $ npm install --save-dev imagemin-pngquant
```

Asset optimizer plugins

```
bash $ npm install --save-dev gulp-clean-css
bash $ npm install --save-dev gulp-uglify
bash $ npm install --save-dev gulp-minify-html
bash $ npm install --save-dev gulp-jshint
bash $ npm install --save-dev gulp-concat
bash $ npm install --save-dev gulp-autoprefixer
```

Anatomy of a gulp function

```
[Function <name>] (glob) {

$.src(glob)

.pipe([plugin 1])
.pipe([plugin 2])
.
.
.
.pipe([plugin n])
.pipe( $.dest(<destination-name>)

}
```

Note

pipe is a method that streams all the files matching the glob input , to our plugins(minifyhtml in this case) .

It is simple to picture it like so:

$.src is what builds the stream and pipe pipes out each individual file matching the glob downwards to each plugin in the pipeline.Each plugin the file is passed to , modifies its contents in memory only until $.dest is reached , which then updates/creates files streamed by $.src

Where ,

$ - > gulp

$$ - > gulp-load-plugins

Asset Optimization and Minification

So, Before writing out optimiser functions , we need to install a couple of caching plugins.

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

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

You might wonder why two caches eh!. **gulp-cached** , passes only modified or new content down the pipeline to other plugins. So, Since we want files without change to be used for concatenating into a single file per asset(css | js) as well, we need **gulp-remember** in addition to gulp-cached

First we use **gulp-cached** to build a list of files that have changed

Second , we need **gulp-remember** to keep a track of all files that are passed through by that list in memory.

First Run : No files are cached , so gulp-cached will pass them all to gulp-remember

Subsequent runs : Only modified or new files are piped down by gulp-cached. Since the content of the current file has changed , gulp-remember updates its cache.

Cool , Let us write our first optimizer

Style Optimizer

```
// Goal

/*

1. cache existing files on the first run
2. each file ,
       a. Is autoprefixed with cross browser compatible prefixes for any css property (
justify-content for e.g)
       b. Is concatenated into app.css
3. app.css is minified
4. On subsequent runs , the above process is implemented on file modifications/additions only

*/

/*
*@src - input a glob pattern - a string eg 'images/**/*' or '**/*.css' or, an array eg
['glob1','glob2']
*/
var optimizeStyles = function(src) {

return $.src(src).
pipe($$.cached('styles')).
pipe($$.autoprefixer({
browsers: ['last 2 versions']
})).
pipe($$.remember('auto-prefixed-stylesheets')).
pipe($$.concat('app.css')).
pipe($.dest('build/css')).
pipe($$.cleanCss()).
pipe($$.rename({
suffix: '.min'
})).
pipe($.dest('build/css'))
}
```

Note

$ - > gulp

$$ - > gulp-load-plugins

$.src - > builds file streams matching the glob passed as src

$.dest - > saves the manipulated file in the path specified

Script Optimiser

```
// Goal

/*

1. cache existing files on the first run
2. each file ,
       a. Is linted with jshint
       b. Is concatenated into app.js
3. app.js is minified
4. On subsequent runs , the above process is implemented on file modifications/additions only

*/

/*
*@src - input a glob pattern - a string eg 'js/**/*' or '**/*.js' or, an array eg
['glob1','glob2']
*/

var optimizeScripts = function(src) {

    return $.src(src).
    pipe($$.cached('scripts')).
    pipe($$.jshint()).
    pipe($$.jshint.reporter('default')).
    pipe($$.jshint.reporter('fail')).
    pipe($$.remember('linted-scripts')).
    pipe($$.concat('app.js')).
    pipe($.dest('build/js')).
    pipe($$.uglify()).
    pipe($$.rename({
        suffix: '.min'
    })).
    pipe($.dest('build/js'))


}
```

Note

$ - > gulp

$$ - > gulp-load-plugins

$.src - > builds file streams matching the glob passed as src

$.dest - > saves the manipulated file in the path specified

Generate Responsive Images

Let us now move on to image processing. So, the aim , is to have an array of sizes for each image you are going to serve.

Why?

> Well , to understand why we need a battery of images with a range of widths , we need to ponder over the fact , that there are probably zillions of devices with varied resolutions. We need an image to scale without much pixelation. At the same time , we need to improve page load times , by downloading just the one image , which fits the width it is contained by , and is also with the smallest possible dimension , to do so. There are scholarly blogs like the one Eric Portis wrote , which highlights the ineffectiveness of just media queries and serves as a comprehensive guide to understanding concepts like srcsets and sizes.

You can refer to Eric Portis' epic write up here

Now, Our function , needs to take a glob , and a width as inputs, and do its magic and push the file each run generates , to a destination and minify the responsified image.

We have installed two image compression plugins in our first example

Since these plugins **DO NOT** start with a "gulp-" prefixed, we need to manually load them onto our gulpfile.

SO Let us require them manually , after the gulp-load-plugins declaration at the top of our gulpfile.

like so:

```
var compressJpg = require('imagemin-jpeg-recompress');
var pngquant = require('imagemin-pngquant');
```

> It is worthwhile to note , that gulp-responsive comes with the sharp image processor , which is better than imagemagick BY FAAAAR!. Sharp is what is used by gulp-responsive , to crop your images to desired widths.
>
> you might look at gulp-responsive-configuration-options,for a comprehensive list of configuration params. I have only used
>
> - width - to crop our images to a width w, passed as a parameter
> - rename - to add a suffix to the image name, so that it remains unique

in my configuration function below. so our function , will crop the image to the width passed as input , for all matching images deciphered via the glob input. then, each image is compressed using jpeg-recompress or pngquant and saved inside build/images.

With that in mind, our function would be like so:

```
/*
@generateResponsiveImages
*@Description:takes in a src of globs, to stream matching image files , a width,
*to resize the matching image to, and a dest to write the resized and minified files to
*@src - input a glob pattern - a string eg 'images/** /*' or 'images/*' or, an array
eg ['glob1','glob2']
*@return returns a stream
*/
var generateResponsiveImages = function(src, width, dest) {

    //declare a default destination
    if (!dest)
        dest = 'build/images';
    //case 1: src glob -  images/**/*
    // the base is the directory immediately preceeding the glob - images/ in this case
    //case 2: images/fixed/flourish.png : the base here is images/fixed/ - we are overriding
    // that by setting base to images.This is done so that, the path beginning after images/
    // - is the path under our destination - without us setting the base, dest would be,
    //build/images/flourish.png.But with us setting the base, the destination would be
    // build/images/fixed/flourish.png
    return $.src(src, {
        base: 'images'
    })

    //generate resized images according to the width passed
    .pipe($$.responsive({
            //match all pngs within the src stream
            '**/*.png': [{
                width: width,
                rename: {
                    suffix: '-' + width
                },
                withoutEnlargement: false,
            }],
            //match all jpgs within the src stream
            '**/*.jpg': [{
                width: width,
                rename: {
                    suffix: '-' + width
                },
                progressive: true,
                withoutEnlargement: false,
            }]
        }, {

            errorOnEnlargement: false,
            errorOnUnusedConfig: false,
            errorOnUnusedImage: false

        }))
        //once the file is resized to width, minify it using the plugins available per format
        .pipe($$.if('*.jpg', compressJpg({
```

```
            min: 30,
            max: 90,
            target: 0.5
        })()))
        //use file based cache gulp-cache and it will minify only changed or new files
        //if it is not a new file and if the contents havent changed, the file is served from
cache
        .pipe($$.cache($$.imagemin({
            verbose: true
        })))


    //write to destination - dest + path from base
    .pipe($.dest(dest));
}
```

Note

$ - > gulp

$$ - > gulp-load-plugins

$.src - > builds file streams matching the glob passed as src

$.dest - > saves the manipulated file in the path specified

Further references

- Gulp-Responsive
- Gulp-Imagemin

HTML Minifier

```
*
 *@minifyHtml
 *Description:takes in a glob src, and minifies all '.html' files matched by the glob
 *@src - input a glob pattern - a string eg '/**/*.html /*' or '*.html' or, an array eg
['glob1','glob2']
 *@dest=file.base means, the modified html file will be in the same directory as the src file
being minified
 *often means, the process is just a modification on the existing src file
 *@return returns a stream
 */
var minifyHtml = function(src) {
    return $.src(src)
        .pipe($$.minifyHtml())
        .pipe($.dest(function(file) {
            //file is provided to a dest callback -
            // Refer here https://github.com/gulpjs/gulp/blob/master/docs/API.md#path
            return file.base;
        }));
}
```

Anatomy of a gulp task

The anatomy of a task definition is like so:

```
$.task(<taskname> , [dependencies] , <body>);
```

dependencies , is an array of tasks that HAVE to finish before the current task you are defining , runs. More like forcing a synchronous execution instead of the default Asynchronous functionality.

Adding Gulp Tasks

So, we now have

- A function defined Above to optimise Styles
- A function defined Above to optimise scripts
- A function defined Above to optimise HTML
- A function to generate multiple images per image Above

All we need to do now, is to invoke them when needed.

Let us write our tasks according to the syntax we defined earlier

```
/*
* $.task('name','dependency array',function)
results in building a task object as below
task:{
'name':name,
'dep':[array of dependencies],
'fn':function
}
*/


//*@return returns a stream to notify on task completion
$.task('optimizeHtml', function() {
    var src = ['**/*.html', '!{node_modules}/**/*.html'];
    return minifyHtml(src);
});

//*@return returns a stream to notify on task completion
$.task('optimizeScripts', function() {
    var src = ['js/**/*.js'];
    return optimizeScripts(src);
});

//*@return returns a stream to notify on task completion
$.task('optimizeStyles', function() {
    var src = ['css/**/*.css', 'fonts/google/**/*.css'];
    return optimizeStyles(src);
});

//Take in a callback to ensure notifying the gulp engine, that the task is done
//required since, you are not returning a stream in this task
$.task('generateResponsiveImages', function(callback) {
    var src = ['images/**/*.{jpg,png}'];
    for (var i = widths.length - 1; i >= 0; i--) {
```

```
        generateResponsiveImages(src, widths[i]);
    }
    callback();

});
```

Read Comprehensive Guide to a Front-end Workflow Automation with Gulpjs -1 of 2 online: https://riptutorial.com/gulp/topic/6342/comprehensive-guide-to-a-front-end-workflow-automation-with-gulpjs--1-of-2

Chapter 4: Concatenating files

Examples

Concat all css files into one using gulp-concat

First, Install `gulp` and `gulp-concat` plugin to your project localy

```
 npm install --save-dev gulp gulp-concat
```

and add gulp-concat task to your gulpfile.js

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

gulp.task('default', function() {
});

gulp.task('css', function() {
    return gulp.src('css/*.css')
        .pipe(concat('concat.css'))
        .pipe(gulp.dest('css/dist'));
});

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

After starting `gulp` command, plugin gulp-concat will take all the CSS files located in the `css/` directory and concatenate them into a one file `css/dist/concat.css`

Concat and Uglify JS and CSS files

Remember to npm install all the files into devDependencies first. E.g.

```
npm install --save-dev gulp gulp-concat gulp-rename gulp-uglify gulp-uglifycss
```

Gulpfile.js

```
var gulp = require('gulp');
var gulp_concat = require('gulp-concat');
var gulp_rename = require('gulp-rename');
var gulp_uglify = require('gulp-uglify');
var uglifycss = require('gulp-uglifycss');

var destDir = './public/assets/dist/'; //or any folder inside your public asset folder
var tempDir = './public/assets/temp/'; //any place where you want to store the concatenated,
but unuglified/beautified files
//To concat and Uglify All JS files in a particular folder
gulp.task('js-uglify', function(){
    return gulp.src(['./public/js/**/*.js','./public/assets/js/*.js']) //Use wildcards to
select all files in a particular folder or be specific
        .pipe(gulp_concat('concat.js')) //this will concat all the files into concat.js
```

```
        .pipe(gulp.dest(tempDir)) //this will save concat.js in a temp directory defined above
        .pipe(gulp_rename('uglify.js')) //this will rename concat.js to uglify.js
        .pipe(gulp_uglify()) //this will uglify/minify uglify.js
        .pipe(gulp.dest(destDir)); //this will save uglify.js into destination Directory
defined above
});
//To Concat and Uglify all CSS files in a particular folder
gulp.task('css-uglify', function () {
  gulp.src('./public/assets/css/*.css') //Use wildcards to select all files in a particular
folder or be specific
  .pipe(gulp_concat('concat.css')) //this will concat all the source files into concat.css
        .pipe(gulp.dest(tempDir)) //this will save concat.css into a temp Directory
        .pipe(gulp_rename('uglify.css')) //this will rename concat.css into uglify.css, but
will not replace it yet.
    .pipe(uglifycss({
      "maxLineLen": 80,
      "uglyComments": true
    })) //uglify uglify.css file
    .pipe(gulp.dest(destDir)); //save uglify.css
});
```

Run them by following commands

```
gulp js-uglify
gulp css-uglify
```

Read Concatenating files online: https://riptutorial.com/gulp/topic/4398/concatenating-files

Chapter 5: Create a watcher

Examples

Watcher task

`config.paths.html` represents the path to your HTML file.

```
gulp.task("watch", function() {
    gulp.watch(config.paths.html, ["html"]);
});
```

The task should be added to default as well:

```
gulp.task("default", ["html", "watch"]);
```

Read Create a watcher online: https://riptutorial.com/gulp/topic/5026/create-a-watcher

Chapter 6: Create documentation with gulp-jsdoc3

Examples

Installation

First of all, install `gulp` and `gulp-jsdoc3` to your project:

```
npm install gulp-jsdoc3 --save-dev
```

Then add it to your gulpfile.js

```
var gulp = require('gulp');
var jsdoc = require('gulp-jsoc3');



gulp.task('doc', function (cb){
    gulp.src('src/*.js')
    .pipe(jsdoc(cb));
});
```

In order to documentate, for example, a function, you have to add a comment just at the top of the function, like this:

```
/**
 * @function example
 * @summary This is a short description of example
 * @author Whoever
 * @param {any} cb
 * @returns
 */
function example(cb){
    //Code
}
```

If you want to know more block tags to use, please visit usejsdoc.org

Read Create documentation with gulp-jsdoc3 online: https://riptutorial.com/gulp/topic/7365/create-documentation-with-gulp-jsdoc3

Chapter 7: Delete Files Using Gulp

Remarks

Note on using the globbing pattern (`**`):

The globbing pattern matches all `children` **and** the `parent`. In order to avoid that we add `'!public'` to our del task so that the `public` directory itself doesn't get deleted

Examples

Delete files using del

First, Install `gulp` and `del` to project directory locally

```
npm install --save-dev gulp del
```

Then add the `clean` task to your `gulpfile.js`

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

gulp.task('default', function() {
});

// Task to delete target build folder
gulp.task('clean', function() {
  return del(['public/**', '!public']);
});

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

This task deletes all files in the public directory

The task in the code is added as a dependency for the `'default'` task so every time `default` will run, `clean` will run before it.

You can also call the `clean` task manually by running the command:

```
gulp clean
```

Read Delete Files Using Gulp online: https://riptutorial.com/gulp/topic/7189/delete-files-using-gulp

Chapter 8: Gulp Path

Examples

Creating Simple Gulp Path to the app

Before running gulp you need to install `Node.JS` and `npm`

The head of the Gulp.js is:

```
var gulp = require('gulp');
    clean = require('gulp-clean'); // A gulp plugin for removing files and folders.
    imagemin = require('gulp-imagemin'); // Minify PNG, JPEG, GIF and SVG images
```

Then assign path of the gulp files

```
var bases = {
 app: 'app/',         // path to your app
 dist: 'dist/',       // path to the compiled application
};

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

Example 1:

```
// Copy all other files to dist directly
    gulp.task('copy', ['clean'], function() {
 // Copy html
       gulp.src(index.html)
         .pipe(gulp.dest(bases.dist)); // same as .pipe(gulp.dest('dist'));
    });
```

Example 2:

```
// 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/')); // same as .pipe(gulp.dest('dist/images/'));
});
```

Default watch

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

Read Gulp Path online: https://riptutorial.com/gulp/topic/7759/gulp-path

Chapter 9: Image lossless compression (with gulp-imagemin)

Syntax

1. imagemin([plugins], {options})

Parameters

Argument	Description
`sourcePath`	Images' source directory (for example: `/assets/images`)
`buildPath`	Destination path (for example: `/static/dist/`)

Remarks

First argument to `imagemin` constructor is plugin array. By default, following plugins are used: `[imagemin.gifsicle(), imagemin.jpegtran(), imagemin.optipng(), imagemin.svgo()]`

Second argument are options. In the above example following options are used:

```
{
  progressive: true,
  interlaced: true,
  svgoPlugins: [{removeUnknownsAndDefaults: false}, {cleanupIDs: false}]
}
```

Those are completely optional.

`progressive` is used by `imagemin-jpegtran`.

`interlaced` is used by `imagemin-gifsicle`.

`removeUnknownsAndDefaults` and `cleanupIDs` are used by `imagemin-svgo`.

Examples

Installation and usage

Dependency installation (https://www.npmjs.com/package/gulp-imagemin)

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

Usage

```
/*
 * Your other dependencies.
 */

var imagemin = require('gulp-imagemin');

/*
 * `gulp images` - Run lossless compression on all the images.
 */
gulp.task('images', function() {
  return gulp.src(sourcePath) // e.g. /assets/images
    .pipe(imagemin({
      progressive: true,
      interlaced: true,
      svgoPlugins: [{removeUnknownsAndDefaults: false}, {cleanupIDs: false}]
    }))
    .pipe(gulp.dest(buildPath + 'images')); // e.g. /static/dist/
});
```

Read Image lossless compression (with gulp-imagemin) online:
https://riptutorial.com/gulp/topic/6549/image-lossless-compression--with-gulp-imagemin-

Chapter 10: Minifying CSS

Examples

Using gulp-clean-css and gulp-rename

First, Install `gulp`, `gulp-clean-css` and `gulp-rename` to project directory localy

```
npm install --save-dev gulp gulp-clean-css gulp-rename
```

Than add following `minify-css` task to your `gulpfile.js`

```
var gulp = require('gulp');
var cleanCSS = require('gulp-clean-css');
var rename = require('gulp-rename');

gulp.task('minify-css', function() {
    return gulp.src('css/dist/dist.css')
        .pipe(cleanCSS())
        .pipe(rename('dist.min.css'))
        .pipe(gulp.dest('css/dist'));
});

gulp.task('watch', function(){
  gulp.watch('css/dist/**/*.css', ['minify-css']);
  // Other watchers
});

gulp.task('default', ['minify-css', 'watch']);
```

Here `.pipe(cleanCSS())` executes minification of your `css/dist/dist.css` file and `.pipe(rename('concat.min.css'))` renames it to `dist.min.css`

Sass and Css - Preprocessing with Gulp

Before starting gulp we need to install `node.js` and `npm`. Then install gulp-sacc

```
$ npm i gulp-sass --save-dev // i = install
```

Gulp Head

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

Gulp Body

```
gulp.task('sass', function(){
  return gulp.src('app/scss/*.scss') // searching for sass files
```

```
    .pipe(sass()) // Converts Sass to CSS with gulp-sass
    .pipe(gulp.dest('app/css')) // destination of the css file
});
```

Gulp watch

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

Read Minifying CSS online: https://riptutorial.com/gulp/topic/4396/minifying-css

Chapter 11: Minifying HTML

Examples

Minify HTML using gulp-htmlmin

First, Install `gulp` and `gulp-htmlmin` to project directory locally

```
npm install --save-dev gulp gulp-htmlmin
```

Then add the `minify-html` task to your `gulpfile.js`

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

// Task to minify HTML
gulp.task('minify-html', function() {
return gulp.src('source/*.html')
.pipe(htmlmin())
.pipe(gulp.dest('public/'));
});

gulp.task('watch', function (){
    gulp.watch('source/*.html', ['minify-html']);
// other tasks
});

gulp.task('default', ['minify-html', 'watch']);
```

This task finds all files in the `source` directory with a `.html` extension, minifies them and then outputs the resulting files to the `public` directory.

The task in the code is added as a dependency for the `'default'` task so every time `default` will run, `minify-html` will run before it.

You can also call the `minify-html` task manually by running the command:

```
gulp minify-html
```

Read Minifying HTML online: https://riptutorial.com/gulp/topic/7187/minifying-html

Chapter 12: Minifying JS

Syntax

- `ext` An object that specifies output source and minified file extensions.
- `source` The suffix string of the filenames that output source files ends with.
- `min` When string: The suffix string of the filenames that output minified files ends with.
- When Array: The regex expressions to be replaced with input filenames. For example: [/.(.*)-source.js$/, '$1.js']
- `exclude` Will not minify files in the dirs.
- `noSource` Will not output the source code in the dest dirs.
- `ignoreFiles` Will not minify files which matches the pattern.
- `mangle` Pass `false` to skip mangling names.
- `output` Pass an object if you wish to specify additional `output options`. The defaults are optimized for best compression.
- `compress` Pass an object to specify custom `compressor options`. Pass false to skip compression completely.
- `preserveComments` A convenience option for options.output.comments. Defaults to preserving no comments.
- `all` Preserve all comments in code blocks
- `some`Preserve comments that start with a bang `(!)` or include a Closure Compiler directive `(@preserve, @license, @cc_on)`
- `function` Specify your own comment preservation function. You will be passed the current node and the current comment and are expected to return either `true` or `false`.

Remarks

Useful Links to gulp-minify

Examples

Minify JS using gulp-minify

First, Install `gulp` and `gulp-minify` to project directory locally

```
npm install --save-dev gulp gulp-minify
```

Then add following `min-js` task to your `gulpfile.js`

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

gulp.task('min-js', function() {
    return gulp.src('lib/*.js')
        .pipe(minify({
```

```
            ext: {
                min: '.min.js'
            },
            ignoreFiles: ['-min.js']
        }))
        .pipe(gulp.dest('lib'))
});

gulp.task('watch', function(){
  gulp.watch('lib/*.js', ['min-js']);
  // Other watchers
});

gulp.task('default', ['min-js', 'watch']);
```

This task find all js files in `lib` directory, minfy it and save to `lib` directory with `.min.js` suffix. For example, after minify `lib/app.js` file will be created a `lib/app.min.js` file

Besides running as a dependency for the `'default'` gulp task, this task can be run manually by typing the following command:

```
gulp min-js
```

Read Minifying JS online: https://riptutorial.com/gulp/topic/4397/minifying-js

Chapter 13: Show errors with gulp-jslint

Examples

Installation and usage

Installation

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

Usage

In gulpfile.js add:

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

gulp.task('lint', function(){
    return gulp.src(['source.js'])
        .pipe(jslint({ /* this object represents the JSLint directives being passed down */
}))
        .pipe(jslint.reporter( 'my-reporter' ));
});
```

for use this task:

```
$ ./node_modules/gulp/bin/gulp.js lint
```

Read Show errors with gulp-jslint online: https://riptutorial.com/gulp/topic/7415/show-errors-with-gulp-jslint

Chapter 14: Using Browserify

Parameters

Options	Details
transform	Specifies a pipeline of functions (or module names) through which the browserified bundle will be run.
debug	Enable source map support. `!gulp.env.production` would work well.
extensions	Array of extensions that you want to skip in `require()` calls in addition to .js and .json. Don't forget .
ignore	Array of paths which should be passed to the ignore function of browserify.
resolve	Custom module name resolution function.
nobuiltins	Remove builtins modules defined in `lib/builtins.js` (browserify module). `opts.builtins` must be not defined and `opts.nobuiltins` can be an Array of Strings or simply a String.

Examples

Using Browserify with Vanilla Javascript

First install gulp and browserify via `npm i gulp gulp-browserify`. This will install browserify into your `node_modules` folder.

gulpfile.js

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

gulp.task('script', function() {
    gulp.src('./src/script.js')
        .pipe(browserify({
            insertGlobals: true
        })
        .pipe(gulp.dest('./build/'));
}
```

Using Browserify with Coffeescript

First install gulp and browserify via npm i gulp gulp-coffeeify. This will install browserify into your node_modules folder.

gulpfile.js

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

gulp.task('script', function() {
    gulp.src('./src/script.coffee')
        .pipe(coffeeify())
        .pipe(gulp.dest('./build/'));
}
```

Read Using Browserify online: https://riptutorial.com/gulp/topic/6364/using-browserify

Chapter 15: Using file filters.

Examples

Creating Images, JS, and CSS(SASS) rule for ordering files

Installing Gulp and His Tasks

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

Determining Folder Structure

In this structure, we will use the app folder for development purposes, while the dist folder is used to contain optimized files for the production site.

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

Gulp Preprocessing

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

Gulp Task

```
gulp.task('sass', function(){
  return gulp.src('app/scss/**/*.scss') //selection all files in this derectory
    .pipe(sass()) // Using gulp-sass
    .pipe(gulp.dest('dist'))
```

```
});
gulp.task('gulp-uglify', function(){
  return gulp.src('app/js/*.js')
    // Minifies only if it's a JavaScript file
    .pipe(uglify())
    .pipe(gulp.dest('dist'))
});
gulp.task('images', function(){
  return gulp.src('app/images/**/*.+(png|jpg|gif|svg)')
  .pipe(imagemin())
  .pipe(gulp.dest('dist/images'))
});
```

Gulp Watch

```
gulp.task('watch', function(){
  gulp.watch('app/js/**/*.js', ['gulp-uglify']);
    gulp.watch('app/scss/**/*.scss', ['sass']);
    gulp.watch('app/images/**/*.*', ['images']);
  // Other watchers
});
gulp.task('build', [`sass`, `gulp-uglify`, `images`], function (){
  console.log('Building files');
});
gulp.task('default', function() {});
```

Read Using file filters. online: https://riptutorial.com/gulp/topic/7818/using-file-filters-

Credits

S. No	Chapters	Contributors
1	Getting started with gulp	cl3m, Community, fracz, João Silva, naresh, Nhan, Pejman, Pierre-Loup Pagniez, Pyloid, Sender, Tim
2	Comprehensive Guide to a Front end Workflow with Gulpjs 2 of 2	Schrodinger's cat
3	Comprehensive Guide to a Front-end Workflow Automation with Gulpjs -1 of 2	Schrodinger's cat
4	Concatenating files	Andrew Plakhotnyi, fracz, John Strood, maoizm, Mikhail, noob
5	Create a watcher	Muaaz Rafi, Nhan
6	Create documentation with gulp-jsdoc3	Manuel
7	Delete Files Using Gulp	Mor Paz
8	Gulp Path	GYTO
9	Image lossless compression (with gulp-imagemin)	Siegmeyer
10	Minifying CSS	fracz, GYTO, Mikhail, SteveLacy
11	Minifying HTML	GYTO, Mor Paz
12	Minifying JS	fracz, GYTO, Mikhail, Mor Paz, Saiful Azad
13	Show errors with gulp-jslint	jesussegado
14	Using Browserify	dome2k
15	Using file filters.	GYTO