Building and Deployment: Objectives and Outcomes

In this lesson you will learn about building and deploying your web project. You will learn to automate your web development tasks using NPM scripts. At the end of this lesson, you will be able to:

* Configure NPM scripts and automate your web development
* Prepare your project for being hosted on a web server

# Exercise (Instructions): NPM Scripts Part 1

### **Objectives and Outcomes**

In this exercise, you will learn to set up NPM scripts by modifying the package.json file. At the end of this exercise, you will be able to:

* Watch for changes to the styles.scss file and automatically compile it to the css file.
* Run multiple NPM scripts in parallel using parallelshell NPM module.

### **Moving JS to Script file**

* Create a folder named js and in that folder create a file named scripts.js.
* Open index.html and from this file cut out all the JQuery script that we added to it and move the code to the scripts.js file that we created above.
* Then, update the index.html file to include the scripts.js file by adding the following line:



1

<script src="js/scripts.js"></script>

* Add the same line to the scripts block in aboutus.html and contactus.html:

### **Watching for Changes and Parallelshell**

* First, we install two NPM packages onchange and parallelshell as follows:



1

npm install --save-dev onchange@3.3.0 parallelshell@3.0.2

* Then, add the following two script items to package.json if you are doing the exercise on a MacOS computer or a Linux computer:



1

2

"watch:scss": "onchange 'css/\*.scss' -- npm run scss",

"watch:all": "parallelshell 'npm run watch:scss' 'npm run lite'"

* **NOTE:**If you are doing the exercise on a Windows computer, please use the following two script items instead of the above:



1

2

"watch:scss": "onchange \"css/\*.scss\" -- npm run scss",

"watch:all": "parallelshell \"npm run watch:scss\" \"npm run lite\""

* You will also update the start script as follows:



1

"start": "npm run watch:all",

* Then, type the following at the prompt to start watching for changes to the SCSS file, compile it to CSS, and run the server:



1

npm start

* Now, whenever you make any changes to styles.scss file, it will automatically be compiled to the corresponding css file.
* Do a Git Commit with the message "NPM Scripts Part 1".

### **Conclusions**

In this exercise, you learnt how to set up a watch task to watch for changes to a file and automatically run tasks upon detecting changes.

# Exercise (Instructions): NPM Scripts Part 2

### **Objectives and Outcomes**

In this exercise you will learn to build a distribution folder containing the files that can be deployed on a web server hosting your project. This distribution folder would be built from your project files using various NPM packages and scripts. At the end of this exercise, you will be able to:

* Clean out a folder using the clean NPM module.
* Copy files from one folder to another
* Prepare a minified and concatenated css file from all the css files used in your project
* Prepare an uglified and concatenated JS file containing all the JS code used in your project

### **Cleaning up a Distribution Folder**

* Install the rimraf npm module by typing the following at the prompt:



1

npm install --save-dev rimraf@2.6.2

* Then, set up the following script:



1

"clean": "rimraf dist",

### **Copying Fonts**

* Your project uses font-awesome fonts. These need to be copied to the distribution folder. To help us do this, install the copyfiles NPM module globally as follows:



1

npm -g install copyfiles@2.0.0

Remember to use sudo on mac and Linux.

* Then set up the following script:



1

"copyfonts": "copyfiles -f node\_modules/font-awesome/fonts/\* dist/fonts",

Compressing and Minifying Images

* We use the imagemin-cli NPM module to help us to compress our images to reduce the size of the images being used in our project. Install the imagemin-cli module as follows:



1

npm -g install imagemin-cli@3.0.0

Remember to use sudo on mac and Linux. **NOTE:** Some students have encountered issues with imagemin-cli not installing its plugins due to issues with global permissions on Mac. In that case try



1

sudo npm install -g imagemin-cli@3.0.0 --unsafe-perm=true --allow-root

* Then set up the following script:



1

"imagemin": "imagemin img/\* --out-dir='dist/img'",

### **Preparing the Distribution Folder**

* Open .gitignore and update it as follows. We do not want the dist folder to be checked into the git repository.



1

2

node\_modules

dist

* Then, install the usemin-cli, cssmin, uglifyjs and htmlmin NPM packages as follows:



1

npm install --save-dev usemin-cli@0.5.1 cssmin@0.4.3 uglifyjs@2.4.11 htmlmin@0.0

  .7

* Add the following two scripts to the package.json file:



1

2

3

"usemin": "usemin contactus.html -d dist --htmlmin -o dist/contactus.html &&

      usemin aboutus.html -d dist --htmlmin -o dist/aboutus.html && usemin index

      .html -d dist --htmlmin -o dist/index.html",

"build": "npm run clean && npm run imagemin && npm run copyfonts && npm run

      usemin"

* Open index.html and surround the css links inclusion code as follows:



1

2

3

4

5

6

<!-- build:css css/main.css -->

<link rel="stylesheet" href="node\_modules/bootstrap/dist/css/bootstrap.min

      .css">

<link rel="stylesheet" href="node\_modules/font-awesome/css/font-awesome.min

      .css">

<link rel="stylesheet" href="node\_modules/bootstrap-social/bootstrap-social

      .css">

<link href="css/styles.css" rel="stylesheet">

<!-- endbuild -->

* Do the same change in aboutus.html and contactus.html
* Similarly, open index.html and surround the js script inclusion code as follows:



1

2

3

4

5

6

<!-- build:js js/main.js -->

<script src="node\_modules/jquery/dist/jquery.slim.min.js"></script>

<script src="node\_modules/popper.js/dist/umd/popper.min.js"></script>

<script src="node\_modules/bootstrap/dist/js/bootstrap.min.js"></script>

<script src="js/scripts.js"></script>

<!-- endbuild -->

* Do the same change in aboutus.html and contactus.html
* To build the distribution folder, you can type the following at the prompt:



1

npm run build

* This will build the dist folder containing the files that are a self-contained version of your project. You can now copy the contents of this folder to a web server that hosts your website.
* After verifying that the dist folder is built correctly, you can now do a git commit with the message "NPM Scripts Part 2"

### **Conclusions**

In this exercise, you learnt the various steps to build the project for deployment using NPM scripts.

# Building and Deployment: NPM Scripts: Additional Resources

### **PDFs of Presentations**

**3-Building-Deployment.pdf**PDF File

**4-NPM-Scripts.pdf**PDF File

### **Additional Resources**

* [Why npm Scripts?](https://css-tricks.com/why-npm-scripts/)
* [How to Use npm as a Build Tool](https://www.keithcirkel.co.uk/how-to-use-npm-as-a-build-tool/)
* [The Command Line for Web Design](https://webdesign.tutsplus.com/series/the-command-line-for-web-design--cms-777)

### **NPM Modules**

* [onchange](https://github.com/Qard/onchange)
* [parallelshell](https://github.com/keithamus/parallelshell)
* [rimraf](https://github.com/isaacs/rimraf)
* [copyfiles](https://github.com/calvinmetcalf/copyfiles)
* [imagemin-cli](https://github.com/imagemin/imagemin-cli)
* [usemin-cli](https://github.com/nelsyeung/usemin-cli)
* [cssmin](https://github.com/jbleuzen/node-cssmin)
* [uglifyjs](https://github.com/mishoo/UglifyJS)
* [htmlmin](https://github.com/jserme/htmlmin)

Building and Deployment: Task Runners: Objectives and Outcomes

In this lesson you will learn about JavaScript based Task runners, Grunt and Gulp. You will learn to automate your web development tasks using these tools. At the end of this lesson, you will be able to:

* Configure Grunt tasks and automate your web development using Grunt
* Define Gulp tasks in code to automate the web development using Gulp

# Exercise (Instructions): Grunt Part 1

### **Objectives and Outcomes**

In this exercise, you will learn to use Grunt, the task runner. You will install Grunt CLI and install Grunt packages using NPM. Thereafter you will configure a Grunt file with a set of tasks to build and serve your web project. At the end of this exercise, you will be able to:

* Install Grunt CLI and Grunt packages in your project
* Configure a Grunt file with a set of tasks to build a web project from a source, and serve the built project using a server.

### **Installing Grunt**

* At the command prompt, type the following to install Grunt command-line interface (CLI):



1

npm install -g grunt-cli@1.2.0

This will install the Grunt CLI globally so that you can use them in all projects.

* Next install Grunt to use within your project. To do this, go to the conFusion folder and type the following at the prompt:



1

npm install grunt@1.0.2 --save-dev

This will install local per-project Grunt to use within your project.

### **Creating a Grunt File**

* Next you need to create a Grunt file containing the configuration for all the tasks to be run when you use Grunt. To do this, create a file named Gruntfile.js in the conFusion folder.
* Next, add the following code to Gruntfile.js to set up the file to configure Grunt tasks:



1

2

3

4

5

6

7

8

'use strict';

module.exports = function (grunt) {

// Define the configuration for all the tasks

grunt.initConfig({

});

};

This sets up the Grunt module ready for including the grunt tasks inside the function above.

### **Compiling SCSS to CSS**

* Next, we are going to set up our first Grunt task. The SASS task converts the SCSS code to CSS. To do this, you need to include some Grunt modules that help us with the tasks. Install the following modules by typing the following at the prompt:



1

2

3

npm install grunt-sass@2.1.0 --save-dev

npm install time-grunt@1.4.0 --save-dev

npm install jit-grunt@0.10.0 --save-dev

The first one installs the Grunt module for SCSS to CSS conversion. The time-grunt module generates time statistics about how much time each task consumes, and jit-grunt enables us to include the necessary downloaded Grunt modules when needed for the tasks.

* Now, configure the SASS task in the Gruntfile as follows, by including the code inside the function in Gruntfile.js:



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

'use strict';

module.exports = function (grunt) {

// Time how long tasks take. Can help when optimizing build times

require('time-grunt')(grunt);

// Automatically load required Grunt tasks

require('jit-grunt')(grunt);

// Define the configuration for all the tasks

grunt.initConfig({

sass: {

dist: {

files: {

'css/styles.css': 'css/styles.scss'

}

}

}

});

grunt.registerTask('css', ['sass']);

};

* Now you can run the grunt SASS task by typing the following at the prompt:



1

grunt css

### **Watch and Serve Tasks**

* The final step is to use the Grunt modules watch and browser-sync to spin up a web server and keep a watch on the files and automatically reload the browser when any of the watched files are updated. To do this, install the following grunt modules:



1

2

npm install grunt-contrib-watch@1.0.0 --save-dev

npm install grunt-browser-sync@2.2.0 --save-dev

* After this, we will configure the browser-sync and watch tasks by adding the following code to the Grunt file:



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

,

watch: {

files: 'css/\*.scss',

tasks: ['sass']

},

browserSync: {

dev: {

bsFiles: {

src : [

'css/\*.css',

'\*.html',

'js/\*.js'

]

},

options: {

watchTask: true,

server: {

baseDir: "./"

}

}

}

}

* Then add the following task to the Grunt file:



1

grunt.registerTask('default', ['browserSync', 'watch']);

* Now if you type the following at the command prompt, it will start the server, and open the web page in your default browser. It will also keep a watch on the files in the css folder, and if you update any of them, it will compile the scss file into css file and load the updated page into the browser (livereload)



1

grunt

* Do a Git commit with the message "Grunt Part 1".

### **Conclusions**

In this exercise you have learnt how to configure a Grunt file to perform several tasks. You were able to start a server with livereload to serve the web page.

# Exercise (Instructions): Grunt Part 2

### **Objectives and Outcomes**

In this exercise, you will continue to learn to use Grunt, the task runner. You will configure the Grunt file with a set of additional tasks to build your web project. At the end of this exercise, you will be able to:

* Configure a Grunt file with a set of tasks to build your web project from a source.

### **Copying the Files and Cleaning Up the Dist Folder**

* Next you will install the Grunt modules to copy over files to a distribution folder named dist, and clean up the dist folder when needed. To do this, install the following Grunt modules:



1

2

npm install grunt-contrib-copy@1.0.0 --save-dev

npm install grunt-contrib-clean@1.1.0 --save-dev

* You will now add the code to perform the copying of files to the dist folder, and cleaning up the dist folder. To do this, add the following code to Gruntfile.js. This should be added right after the configuration of the SASS task.:



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

,

copy: {

html: {

files: [

{

//for html

expand: true,

dot: true,

cwd: './',

src: ['\*.html'],

dest: 'dist'

}]

},

fonts: {

files: [

{

//for font-awesome

expand: true,

dot: true,

cwd: 'node\_modules/font-awesome',

src: ['fonts/\*.\*'],

dest: 'dist'

}]

}

},

clean: {

build: {

src: [ 'dist/']

}

}

* Remember to add the comma after the end of the SASS task.

### **Compressing and Minifying Images**

* Next we install the grunt-contrib-imagemin module and use it to process the images. To install this module type at the prompt:



1

npm install grunt-contrib-imagemin@2.0.1 --save-dev

* Then, configure the imagemin task as shown below in the Gruntfile:



1

2

3

4

5

6

7

8

9

10

11

,

imagemin: {

dynamic: {

files: [{

expand: true, // Enable dynamic expansion

cwd: './', // Src matches are relative to

                      this path

src: ['img/\*.{png,jpg,gif}'], // Actual patterns to match

dest: 'dist/' // Destination path prefix

}]

}

}

### **Preparing the Distribution Folder and Files**

* We are now going to use the Grunt usemin module together with concat, cssmin, uglify and filerev to prepare the distribution folder. To do this, install the following Grunt modules:



1

2

3

4

5

6

npm install grunt-contrib-concat@1.0.1 --save-dev

npm install grunt-contrib-cssmin@2.2.1 --save-dev

npm install grunt-contrib-htmlmin@2.4.0 --save-dev

npm install grunt-contrib-uglify@3.3.0 --save-dev

npm install grunt-filerev@2.3.1 --save-dev

npm install grunt-usemin@3.1.1 --save-dev

* Next, update the task configuration within the Gruntfile.js with the following additional code to introduce the new tasks:



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

,

useminPrepare: {

foo: {

dest: 'dist',

src: ['contactus.html','aboutus.html','index.html']

},

options: {

flow: {

steps: {

css: ['cssmin'],

js:['uglify']

},

post: {

css: [{

name: 'cssmin',

createConfig: function (context, block) {

var generated = context.options.generated;

generated.options = {

keepSpecialComments: 0, rebase: false

};

}

}]

}

}

}

},

// Concat

concat: {

options: {

separator: ';'

},

// dist configuration is provided by useminPrepare

dist: {}

},

// Uglify

uglify: {

// dist configuration is provided by useminPrepare

dist: {}

},

cssmin: {

dist: {}

},

// Filerev

filerev: {

options: {

encoding: 'utf8',

algorithm: 'md5',

length: 20

},

release: {

// filerev:release hashes(md5) all assets (images, js and css )

// in dist directory

files: [{

src: [

'dist/js/\*.js',

'dist/css/\*.css',

]

}]

}

},

// Usemin

// Replaces all assets with their revved version in html and css files.

// options.assetDirs contains the directories for finding the assets

// according to their relative paths

usemin: {

html: ['dist/contactus.html','dist/aboutus.html','dist/index.html'],

options: {

assetsDirs: ['dist', 'dist/css','dist/js']

}

},

htmlmin: { // Task

dist: { // Target

options: { // Target options

collapseWhitespace: true

},

files: { // Dictionary of

                  files

'dist/index.html': 'dist/index.html', // 'destination':

                      'source'

'dist/contactus.html': 'dist/contactus.html',

'dist/aboutus.html': 'dist/aboutus.html',

}

}

}

* Next, update the jit-grunt configuration as follows, to inform it that useminPrepare task depends on the usemin package:



1

2

3

require('jit-grunt')(grunt, {

useminPrepare: 'grunt-usemin'

});

* Next, update the Grunt build task as follows:



1

2

3

4

5

6

7

8

9

10

11

12

grunt.registerTask('build', [

'clean',

'copy',

'imagemin',

'useminPrepare',

'concat',

'cssmin',

'uglify',

'filerev',

'usemin',

'htmlmin'

]);

* Now if you run Grunt, it will create a dist folder with the files structured correctly to be distributed to a server to host your website. To do this, type the following at the prompt:



1

grunt build

### **Conclusions**

In this exercise you have learnt how to configure a Grunt file to perform several tasks. You were able to build a distribution folder for your web project.

# Exercise (Instructions): Gulp Part 1

### **Objectives and Outcomes**

In this exercise, you will learn to use Gulp, the task runner. You will install Gulp CLI and install Gulp plugins using NPM. Thereafter you will configure a Gulp file with a set of tasks to build and serve your web project. At the end of this exercise, you will be able to:

* Install Gulp CLI and Gulp plugins in your project
* Configure a Gulp file with a set of tasks to build a web project from a source, and serve the built project using a server.

### **Installing Gulp**

* At the command prompt, type the following to install Gulp command-line interface (CLI) globally:



1

npm install -g gulp-cli@2.0.1

This will install the Gulp globally so that you can use it in all projects.

* Next install Gulp to use within your project. To do this, go to the conFusion folder and type the following at the prompt:



1

npm install gulp@3.9.1 --save-dev

This will install local per-project Gulp to use within your project.

### **Install Gulp Plugins for SASS and Browser-Sync**

* Install all the Gulp plugins that you will need for this exercise. To do this, type the following at the command prompt:



1

npm install gulp-sass@3.1.0 browser-sync@2.23.6 --save-dev

### **Creating a Gulp File**

* Next you need to create a Gulp file containing the tasks to be run when you use Gulp. To do this, create a file named gulpfile.js in the conFusion folder.

### **Loading Gulp Plugins**

* Load in all the Gulp plugins by including the following code in the Gulp file:



1

2

3

4

5

'use strict';

var gulp = require('gulp'),

sass = require('gulp-sass'),

browserSync = require('browser-sync');

### **Adding Gulp Tasks for SASS and Browser-Sync**

* Next, we will add the code for the SASS task, the Browser-Sync task and the default task as follows:



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

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

return gulp.src('./css/\*.scss')

.pipe(sass().on('error', sass.logError))

.pipe(gulp.dest('./css'));

});

gulp.task('sass:watch', function () {

gulp.watch('./css/\*.scss', ['sass']);

});

gulp.task('browser-sync', function () {

var files = [

'./\*.html',

'./css/\*.css',

'./img/\*.{png,jpg,gif}',

'./js/\*.js'

];

browserSync.init(files, {

server: {

baseDir: "./"

}

});

});

// Default task

gulp.task('default', ['browser-sync'], function() {

gulp.start('sass:watch');

});

* Save the Gulp file

### **Running the Gulp Tasks**

* At the command prompt, if you type gulp it will run the default task:



1

gulp

* Do a Git commit with the message "Gulp Part 1".

### **Conclusions**

In this exercise, you learnt to use Gulp, install Gulp plugins, configure the gulpfile.js and then use Gulp to automate the web development tasks.

# Exercise (Instructions): Gulp Part 2

### **Objectives and Outcomes**

In this exercise, you will continue to learn to use Gulp. Thereafter you will configure a Gulp file with a set of tasks to build and serve your web project. At the end of this exercise, you will be able to:

* Configure the Gulp file with a set of tasks to build the distribution folder for the web project.

### **Copying the Files and Cleaning up the Dist Folder**

* We will now create the tasks for copying the font files and cleaning up the distribution folder. To do this we will first install the del Node module and require it in the Gulp file as follows:



1

npm install del@3.0.0 --save-dev



1

2

3

var ...

del = require('del'),

...

* Next, we will add the code for the Clean task and the copyfonts task as follows:



1

2

3

4

5

6

7

8

9

// Clean

gulp.task('clean', function() {

return del(['dist']);

});

gulp.task('copyfonts', function() {

gulp.src('./node\_modules/font-awesome/fonts/\*\*/\*.{ttf,woff,eof,svg}\*')

.pipe(gulp.dest('./dist/fonts'));

});

### **Compressing and Minifying Images**

* We will now install the gulp-imagemin plugin and configure the imagemin task. To do this we install the plugin and require it as follows:



1

npm install gulp-imagemin@4.1.0 --save-dev



1

2

3

var ...

imagemin = require('gulp-imagemin'),

...

* Next, we create the imagemin task as follows:



1

2

3

4

5

6

// Images

gulp.task('imagemin', function() {

return gulp.src('img/\*.{png,jpg,gif}')

.pipe(imagemin({ optimizationLevel: 3, progressive: true, interlaced: true }

      ))

.pipe(gulp.dest('dist/img'));

});

### **Preparing the Distribution Folder and Files**

* We now install the gulp-usemin and other related Gulp plugins and require them as follows:



1

2

npm install gulp-uglify@3.0.0 gulp-usemin@0.3.29 gulp-rev@8.1.1 gulp-clean-css@3

  .9.3 gulp-flatmap@1.0.2 gulp-htmlmin@4.0.0 --save-dev



1

2

3

4

5

6

7

var ...

uglify = require('gulp-uglify'),

usemin = require('gulp-usemin'),

rev = require('gulp-rev'),

cleanCss = require('gulp-clean-css'),

flatmap = require('gulp-flatmap'),

htmlmin = require('gulp-htmlmin');

* We configure the usemin and the build task as follows:



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

gulp.task('usemin', function() {

return gulp.src('./\*.html')

.pipe(flatmap(function(stream, file){

return stream

.pipe(usemin({

css: [ rev() ],

html: [ function() { return htmlmin({ collapseWhitespace: true })} ]

              ,

js: [ uglify(), rev() ],

inlinejs: [ uglify() ],

inlinecss: [ cleanCss(), 'concat' ]

}))

}))

.pipe(gulp.dest('dist/'));

});

gulp.task('build',['clean'], function() {

gulp.start('copyfonts','imagemin','usemin');

});

* Save the Gulp file

### **Running the Gulp Tasks**

* At the command prompt, if you type gulp build it will run the build task:



1

gulp build

* Do a Git commit with the message "Gulp Part 2"

### **Conclusions**

In this exercise, you learnt to use Gulp, install Gulp plugins, configure the gulpfile.js and then use Gulp to automate the web development tasks.

# Building and Deployment: Task Runners: Additional Resources

### **PDFs of Presentations**

**5-Task-Runners.pdf**PDF File

### **Grunt Resources**

* [Grunt](http://gruntjs.com/)
* [Writing an Awesome Build Script with Grunt](http://www.sitepoint.com/writing-awesome-build-script-grunt/)
* [Clean Grunt](http://anders.janmyr.com/2014/01/clean-grunt.html)
* [File Globbing](http://gruntjs.com/configuring-tasks#globbing-patterns)
* [The Command Line for Web Design: Automation With Grunt](https://webdesign.tutsplus.com/tutorials/the-command-line-for-web-design-automation-with-grunt--cms-23454)

### **Grunt Plugins**

* [grunt-contrib-jshint](https://github.com/gruntjs/grunt-contrib-jshint)
* [jshint-stylish](https://github.com/sindresorhus/jshint-stylish)
* [grunt-contrib-copy](https://github.com/gruntjs/grunt-contrib-copy)
* [grunt-contrib-clean](https://github.com/gruntjs/grunt-contrib-clean)
* [grunt-usemin](https://github.com/yeoman/grunt-usemin)
* [grunt-contrib-concat](https://github.com/gruntjs/grunt-contrib-concat)
* [grunt-contrib-cssmin](https://github.com/gruntjs/grunt-contrib-cssmin)
* [grunt-contrib-htmlmin](https://github.com/gruntjs/grunt-contrib-htmlmin)
* [grunt-contrib-uglify](https://github.com/gruntjs/grunt-contrib-uglify)
* [grunt-filerev](https://github.com/yeoman/grunt-filerev)

### **Gulp Resources**

* [Gulp](http://gulpjs.com/)
* [An Introduction to Gulp.js](http://www.sitepoint.com/introduction-gulp-js/)
* [Getting started with gulp](https://markgoodyear.com/2014/01/getting-started-with-gulp/)
* [Building with Gulp](http://www.smashingmagazine.com/2014/06/building-with-gulp/)
* [The Command Line for Web Design: Automation with Gulp](https://webdesign.tutsplus.com/tutorials/the-command-line-for-web-design-automation-with-gulp--cms-23642)

### **Gulp Plugins**

* [gulp](https://github.com/gulpjs/gulp)
* [gulp-sass](https://github.com/dlmanning/gulp-sass)
* [browser-sync](https://github.com/BrowserSync/browser-sync)
* [del](https://github.com/sindresorhus/del)
* [gulp-imagemin](https://github.com/sindresorhus/gulp-imagemin)
* [gulp-uglify](https://github.com/terinjokes/gulp-uglify)
* [gulp-usemin](https://github.com/zont/gulp-usemin)
* [gulp-rev](https://github.com/sindresorhus/gulp-rev)
* [gulp-clean-css](https://github.com/scniro/gulp-clean-css)
* [gulp-flatmap](https://github.com/mariusGundersen/gulp-flatMap)
* [gulp-htmlmin](https://github.com/jonschlinkert/gulp-htmlmin)

### **Tasks**

* [Minification](https://en.wikipedia.org/wiki/Minification_(programming))
* [UglifyJS](http://lisperator.net/uglifyjs/)
* [JSHint](http://jshint.com/)

### **General Resources**

* [Node, Grunt, Bower and Yeoman - A Modern web dev's Toolkit](http://juristr.com/blog/2014/08/node-grunt-yeoman-bower/)
* [The Advantages of Using Task Runners](https://www.dbswebsite.com/blog/2015/02/24/the-advantages-of-using-task-runners/)
* [Gulp vs Grunt. Why one? Why the Other?](https://medium.com/@preslavrachev/gulp-vs-grunt-why-one-why-the-other-f5d3b398edc4)
* [Why we should stop using Grunt & Gulp](http://blog.keithcirkel.co.uk/why-we-should-stop-using-grunt/)
* [Why I Left Gulp and Grunt for npm Scripts](https://medium.freecodecamp.com/why-i-left-gulp-and-grunt-for-npm-scripts-3d6853dd22b8)

# Front-End Web UI Frameworks: Bootstrap 4: Conclusions

### **PDFs of Presentations**

**7-Conclusion.pdf**PDF File