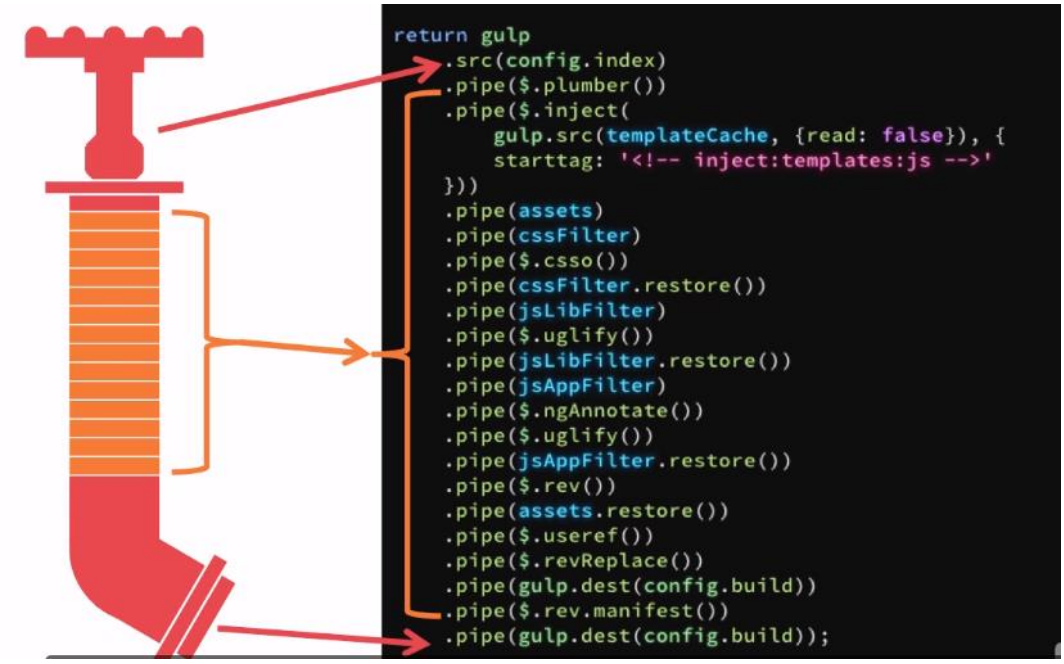
Gulp

JavaScript Build Automation with Gulp.js

Course Introduction

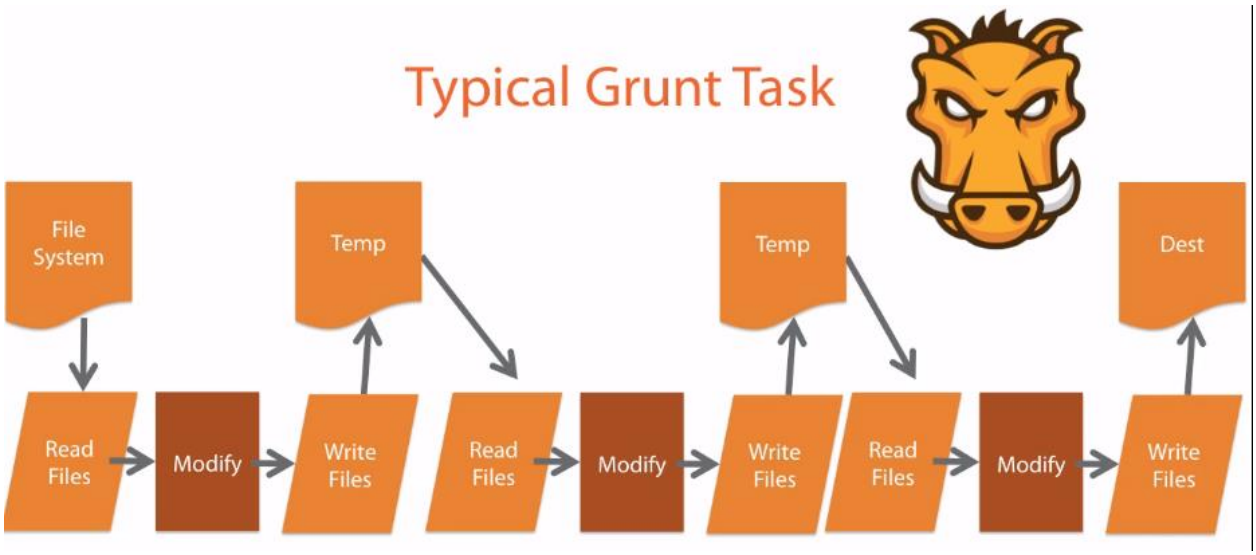


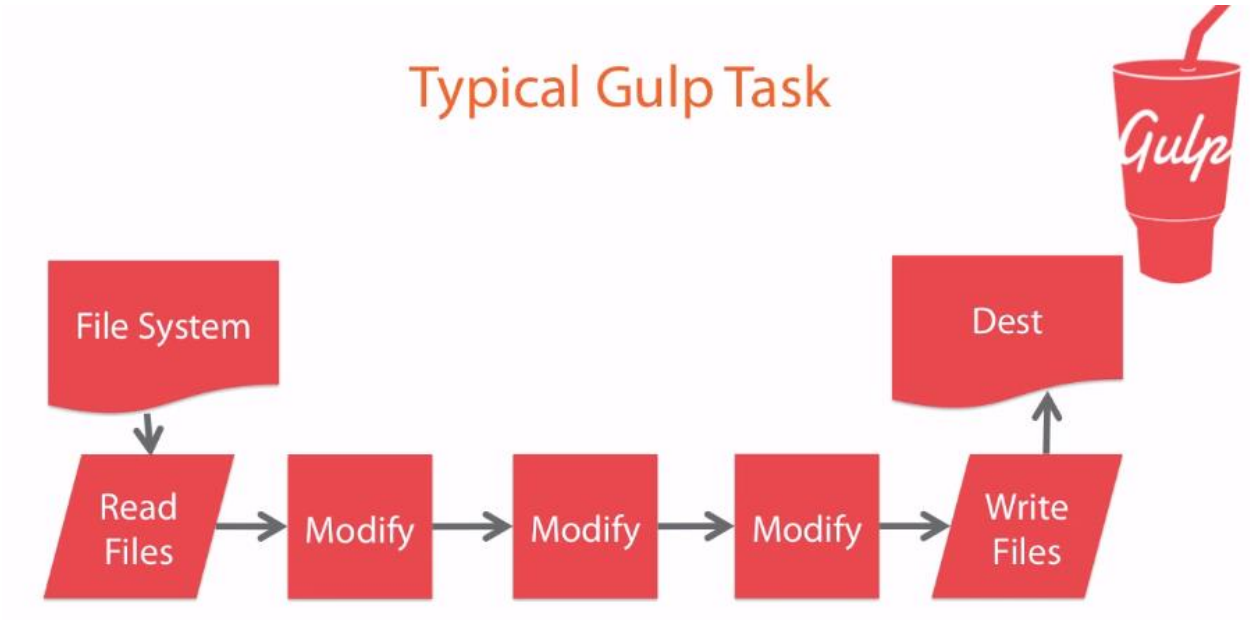
The Value of Gulp as a JavaScript Task Runner

Automated tasks for code quality, tests, and build pipeline.

In gulp files stays in memory stream and we can modify them when we want, in grunt it stores file in temporary location which make hard-disk access again and again for read-write operations, which makes it a bit slower.

Grunt and Gulp workflow: get files, modify them, and make new ones from them.






```
gulp.task('js', function() {
    return gulp
        .src('**/*.js')
        .pipe($.jshint())
        .pipe($.concat())
        .pipe($.uglify())
        .pipe(gulp.dest('./build/'));
});
```

4 Things You Need to Know About Gulp

Gulp has below 4 simple API

1. gulp.task
2. gulp.src
3. gulp.dest
4. gulp.watch

The gulp.task API – use it when making new tasks.

1. gulp.task (name [, dep], fn)

The dependencies will run in parallel before the task function will run.

```
gulp.task('js', ['jscs', 'jshint'], function() {
    return gulp
        .src('./src/**/*.js')
        .pipe(concat('all.js'))
        .pipe(uglify())
        .pipe(gulp.dest('./build/'));
});
```

The gulp.src API – it is the beginning part of the stream. Below glob is a file pattern match for source, options can ban like options.base:

2. gulp.src (glob [, options])

The gulp.dest API – use this to write files, write to same file or new file, write to destination different from source.

3. gulp.dest (folder [, options])

The gulp.watch API – it allows to watch files and then perform a task or function.

4. gulp.watch (glob [, options], tasks)

```
gulp.task('lint-watcher', function() {
    gulp.watch('./src/**/*.js', [
        'jshint',
        'jscs'
    ]);
});
```

Getting Started With Gulp

Getting Node.js

OSX

- Install Homebrew
  - http://brew.sh/

```
$ brew install node
```

Windows

- Install Chocolatey
  - https://chocolatey.org/

```
choco install nodejs
choco install nodejs.install
```

Installing the Gulp CLI:

```
$ npm install -g gulp bower
```

Creating a gulpfile.js

dependencies

- Needed at run-time
- Examples:
  - Express, Angular, Bootstrap
- Code
  - npm install --save
  - bower install --save

devDependencies

- Needed during development
- Examples:
  - Concat, JSHint, Uglify
- Code
  - npm install --save-dev
  - bower install --save-dev

```
gulp.task('hello-world', function() {
    console.log('Our first gulp task!');
});
```

Practice – Creating HelloWorld task in gulp

Code Analysis with JSHint and JSCS

Use JSCS for JavaScript code style checking:

JSCS

JavaScript Code Style Checker

Enforcing your style guide

http://catatron.com/node-jscs/

customers.controller.js | customer-detail.controller.js | .jscsrc

```
{
    "excludeFiles": ["node_modules/**", "bower_components/**"],

    "requireCurlyBraces": [
        "if",
        "else",
        "for",
        "while",
        "do",
        "try",
        "catch"
    ],
    "requireOperatorBeforeLineBreak": true,
    "requireCamelCaseOrUpperCaseIdentifiers": true,
    "maximumLineLength": {
      "value": 100,
```

Practice – using code analysis with JSHint and JSCS

Displaying the source file using gulp-print or gulp-if, use yargs for passing command line arguments:

customers.controller.js | customer-detail.controller.js | .jscsrc

```
{
    "excludeFiles": ["node_modules/**", "bower_components/**"],

    "requireCurlyBraces": [
        "if",
        "else",
        "for",
        "while",
        "do",
        "try",
        "catch"
    ],
    "requireOperatorBeforeLineBreak": true,
    "requireCamelCaseOrUpperCaseIdentifiers": true,
    "maximumLineLength": {
      "value": 100,
```


Use gulp-load-plugins for lazy loading of gulp plugins

```
var $ = require('gulp-load-plugins')({lazy: true});
```

```
.pipe($.if(args.verbose, $.print()))
.pipe($.jscs())
.pipe($.jshint())
.pipe($.jshint.reporter('jshint-stylish', {verbose: true}))
.pipe($.jshint.reporter('fail'));
```

Reusable configuration module by using file gulp.config.js

customer-detail.controller.js | gulpfile.js | gulp.config.js

```
1  module.exports = function() {
2      var config = {
3          alljs: [
4              './src/**/*.js',
5              './*.js'
6          ]
7      };
8
9      return config;
10 };
11
```

By this we can replace below:

```
return gulp
    .src([
        './src/**/*.js',
        './*.js'
    ])
```

With this:

```
var config = require('./gulp.config')();
```

```
return gulp
    .src(config.alljs)
    .pipe($.if(args.verbose, $.print()))
```

CSS Compilation

Use library autoprefixer to add post vendor prefixes on the CSS styles.

```
.sample-style {
    transform-origin: center bottom;
}
```

Added the vendor prefix

```
.sample-style {
    -webkit-transform-origin: center bottom;
            transform-origin: center bottom;
}
```

Creating a less and autoprefixer gulp task

```
gulp.task('styles', function() {
    log('Compiling Less --> CSS');

    return gulp
        .src(config.less) //TODO add the config
        .pipe($.less())
        .pipe($.autoprefixer({browsers: ['last 2 version', '> 5%']}))
        .pipe(gulp.dest(config.temp));
});
```

Practice - Creating a less and autoprefixer gulp task

Deleting files in a dependency task: run this task before the styles task as a dependency task -

```
gulp.task('clean-styles', function() {
    var files = config.temp + '**/*.css';
    del(files);
});
```

Make sure that we do the callback when we are not actually returning the stream in a task:

```
gulp.task('clean-styles', function(done) {
    var files = config.temp + '**/*.css';
    clean(files, done);
});

////////////

function clean(path, done) {
    log('Cleaning: ' + $.util.colors.blue(path));
    del(path, done);
}
```

Creating a watch task to compile CSS

```
gulp.task('less-watcher', function() {
    gulp.watch([config.less], ['styles']);
});
```

Handling errors and using gulp plumber

Using on error event:

```
.pipe($.less())
.on('error', errorLogger)
.pipe($.autoprefixer({browsers: ['last 2 vers
```

```
15:12:27] Finished 'less-watcher' after 4.55 ms
15:12:31] Starting 'styles'...
15:12:31] Compiling Less --> CSS
15:12:31] *** Start of Error ***
15:12:31] Parse
15:12:31] /Users/john/code/pluralsight-gulp/src/client/styles/styles.less
15:12:31] 164
15:12:31] 8
15:12:31] NaN
15:12:31] undefined
15:12:31] 14
15:12:31] ,@color_nothing: #000000,
15:12:31] Unrecognised input in file /Users/john/code/pluralsight-gulp/src/c
```

Using plumber library:

```
return gulp
    .src(config.less)
    .pipe($.plumber())
    .pipe($.less())
```

```
[15:14:03] Compiling Less --> CSS
[15:14:03] Finished 'styles' after 346 ms
[15:14:13] Starting 'styles'...
[15:14:13] Compiling Less --> CSS
[15:14:13] Plumber found unhandled error:
Error in plugin 'gulp-less'
Message:
    Unrecognised input in file /Users/john/code/pluralsight-gulp/src/cli
les/styles.less line no. 10
Details:
```

HTML Injection

The wiredep is used to inject bower dependencies into HTML. To inject custom dependencies into HTML use gulp-inject.

```
<html ng-app="app">
    <head>
    <!-- bower:css -->
    <!-- endbower -->

    <!-- inject:css -->
    <!-- endinject -->
    </head>
    <body>
        <div ng-include="'app/layout/shell.html'"></div>
        <!-- bower:js -->
        <!-- endbower -->

        <!-- inject:js -->
        <!-- endinject -->
    </body>
</html>
```

Inject CSS from bower

Inject application's CSS

Inject JavaScript from bower

Inject applications' JavaScript

```
gulp.task('wiredep', function() {
    var options = config.getWiredepDefaultOptions();
    var wiredep = require('wiredep').stream;

    return gulp
        .src(config.index)
        .pipe(wiredep(options))
        .pipe($.inject(gulp.src(config.js)))
        .pipe(gulp.dest(config.client));
});
```

```
config.getWiredepDefaultOptions = function() {
    var options = {
        bowerJson: config.bower.json,
        directory: config.bower.directory,
        ignorePath: config.bower.ignorePath
    };
    return options;
};
```

Practice – using gulp wiredep

We can add bower files automatically after bower install of any other new dependency.

```
gulpfile.js   gulp.config.js   index.html   bower.json   .bowerrc

1  {
2      "directory": "bower_components",
3      "scripts": {
4          "postinstall": "gulp wiredep"
5      }
6  }
7
```

Injecting custom CSS

```
gulp.task('inject', ['wiredep', 'styles'], function() {
    log('Wire up the app css into the html, and call wiredep ');

    return gulp
        .src(config.index)
        .pipe($.inject(gulp.src(config.css)))
        .pipe(gulp.dest(config.client));
});
```

Serving Your Dev Build

When we do changes in node code the node server should be re-started. We can use the nodemon which do restart the node server, watch files, and handle events.

```
gulp.task('serve-dev', ['inject'], function() {
    var isDev = true;

    var nodeOptions = {
        script: config.nodeServer, //TODO app.js
        delayTime: 1,
        env: {
            'PORT': port,
            'NODE_ENV': isDev ? 'dev' : 'build'
        }

    };

    return $.nodemon(nodeOptions);
});
```

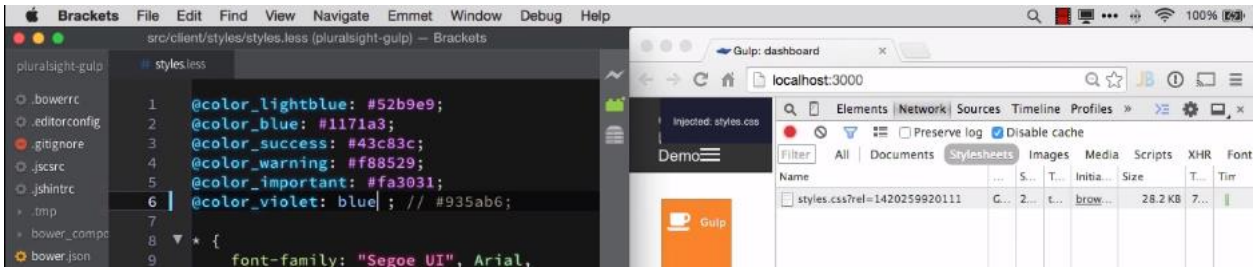
Attaching events on nodeman process:

```
return $.nodemon(nodeOptions)
    .on('restart', ['vet'], function(ev) {
        log('*** nodemon restarted');
        log('files changed on restart:\n' + ev);
    })
    .on('start', function() {
        log('*** nodemon started');
    })
    .on('crash', function() {
        log('*** nodemon crashed: script crashed for some reason');
```

Practice – using nodemon for dev build

Keeping Your Browser in Sync

Automating browser load and unload using tool browsersync, it uses socket.io to do injecting file changes. Like in below case if the CSS file gets change it serve it over the network using socket.io, so it will not reload the whole page, unlike html or javascript files:



We can configure it like below:

```
function startBrowserSync() {
    if (browserSync.active) {
        return;
    }

    log('Starting browser-sync on port ' + port);

    browserSync(options);
}
```

Using the ghost mode setting, if we do changes on one opened browser then same will be reflected on another opened browser.

Practice – using browsersync for synching the browser

Building Assets and Keeping Organized

Creating a distribution, or production, or build folder, use gulp-imagemin for compressing the images, we can use gulp-task-listing package to create list of task in our project.

Create task listing: it will show all the task which we have defined.

```
gulp.task('help', $.taskListing);
gulp.task('help', function() {
    $.taskListing();
});
```

```
gulp.task('help', $.taskListing);
gulp.task('default', ['help']);
```

Copying fonts, and put them into build folder:

```
gulp.task('fonts', function() {
    log('Copying fonts');

    return gulp
        .src(config.fonts)
        .pipe(gulp.dest(config.build + 'fonts'));
});
```

Optimizing images

```
gulp.task('images', function() {
    log('Copying and compressing the images');

    return gulp
        .src(config.images)
        .pipe($.imagemin({optimizationLevel: 4}))
        .pipe(gulp.dest(config.build + 'images'));
});
```

Cleaning – before running a build task we should clean out that build folder for each task:

```
gulp.task('clean-fonts', function(done) {
    clean(config.build + 'fonts/**/*.*', done);
});

gulp.task('clean-images', function(done) {
    clean(config.build + 'images/**/*.*', done);
});
```

```
gulp.task('clean', function(done) {
    var delconfig = [].concat(config.build, config.temp);
    log('Cleaning: ' + $.util.colors.blue(delconfig));
    del(delconfig, done);
});
```

Practice – creating build folder

Caching HTML Templates for Angular

To reduce the HTTP XHR calls we should use angular template cache to improve the performance like directives, and routes. In templateCache works as key/value pair, the URL becomes the key which is used by angular to check if it already have this file at that URL.

Using gulp we can create a task which will gather all templates, then minify the HTML and them add them into $templateCache then put them in an angular module.

Note: in directive on templateURL property, when angular app runs it find its template at the first time, then make it cache, using above gulp approach we try to make warm load by making it available in the template cache at the first time to avoid the HTTP call.

```
gulp.task('clean-code', function(done) {
    var files = [].concat(
        config.temp + '**/*.js',
        config.build + '**/*.html',
        config.build + 'js/**/*.js'
    );
    clean(files, done);
});
```

```
gulp.task('templatecache', ['clean-code'], function() {
    log('Creating AngularJS $templateCache');

    return gulp
        .src(config.htmltemplates)
        .pipe($.minifyHtml({empty: true}))
        .pipe($.angularTemplatecache(
            config.templateCache.file,
            config.templateCache.options
            ))
        .pipe(gulp.dest(config.temp));
});
```

```
.tmp/templates.js (pluralsight-gulp) — Brackets
gulpfile.js    gulp.config.js    templates.js
1  angular.module("app.core").run(["$templateCache", function($templateCache)
   $templateCache.put "app/customers/customer-detail.html","<section class=mainbar><section
   class=matter><div class=container><div><button class=\"btn btn-info btn-form-md\" ng-
   click=vm.goBack()><i class=\"fa fa-hand-o-left\"></i>Back</button> <button class=\"btn btn-info
   btn-form-md\" ng-click=vm.cancel() ng-disabled=vm.isUnchanged()><i class=\"fa fa-undo\">
   </i>Cancel</button> <button class=\"btn btn-info btn-form-md\" ng-click=vm.save() ng-
   disabled=\"form.$invalid || vm.isUnchanged()\"><i class=\"fa fa-save\"></i>Save</button><span
   ng-hide=vm.isUnchanged() class=\"dissolve-animation ng-hide flag-haschanges\"><i class=\"fa fa-
   asterisk fa fa-asterisk-large\" rel=tooltip title=\"You have changes\"></i></span></div><div
   class=\"widget wblue\"><div ht-widget-header=\"\" title=\"Edit {{vm.getFullName() || \'New
   Customer\'}}\"></div><div class=\"widget-content user\"><div class=form-group><label
```

Practice – caching HTML templates for angular

Creating a Production Build Pipeline

```
<script src="/bower_components/bootstrap/dist/js/boo
<script src="/bower_components/extras.angular.plus/n
<script src="/bower_components/moment/moment.js"></s
<script src="/bower_components/angular-ui-router/rel
<script src="/bower_components/toastr/toastr.js"></s
<script src="/bower_components/angular-animate/angul
<!-- endbower -->
<!-- endbuild -->
```

Dev-time index.html

```
<!-- build:js js/app.js-->
<!-- inject:js -->
<script src="/src/client/app/app.module.js"></script
<script src="/src/client/app/core/core.module.js"></
<script src="/src/client/app/customers/customers.mod
<script src="/src/client/app/dashboard/dashboard.mod
<script src="/src/client/app/layout/layout.module.js
```

Build-time index.html

```
<script src="js/lib.js"></script>

<script src="js/app.js"></script>
```

To gather assets we use gulp-userref which parses HTML comments, it is similar to gulp-inject.

The gulp-useref API

.pipe($.useref.assets())	.pipe($.useref.assets.restore())	.pipe($.useref ())
Gathers assets from the HTML comments	Restore the files to the stream. index.html, for example	Concatenate files

Adding gulp task to add template.js reference inside index.html for build folder:

```
gulp.task('optimize', ['inject'], function() {
    log('Optimizing the javascript, css, html');

    var templateCache = config.temp + config.templateCache.file;

    return gulp
        .src(config.index)
        .pipe($.plumber())
        .pipe($.inject(gulp.src(templateCache, {read: false}), {
            starttag: '<!-- inject:templates:js -->'
        }))
        .pipe(gulp.dest(config.build));
});
```

Adding gulp task to add optimize files reference to index.html file:

```
    var assets = $.useref.assets({searchPath: './'});
    var templateCache = config.temp + config.templateCache.file;

    return gulp
        .src(config.index)
        .pipe($.plumber())
        .pipe($.inject(gulp.src(templateCache, {read: false}), {
            starttag: '<!-- inject:templates:js -->'
        }))
        .pipe(assets)
        .pipe(assets.restore())
        .pipe($.useref())
        .pipe(gulp.dest(config.build));
});
```

Serving the build code:

```
if(isDev) {
    gulp.watch([config.less], ['styles'])
        .on('change', function(event) { changeEvent(event); });
} else {
    gulp.watch([config.less, config.js, config.html], ['optimize', browserSync.reload])
        .on('change', function(event) { changeEvent(event); });
}
```

Practice - Creating a Production Build Pipeline

Minifying and Filtering

Minifying the assets – we can use gulp tool gulp-csso, gulp-uglify to optimize CSS, and JavaScript.

```
return gulp
    .src(config.index)
    .pipe($.plumber())
    .pipe($.inject(gulp.src(templateCache, {read: false}), {
        starttag: '<!-- inject:templates:js -->'
    }))
    .pipe(assets)
    .pipe(cssFilter)
    .pipe($.csso())
    .pipe(cssFilter.restore())
    .pipe(jsFilter)
    .pipe($.uglify())
    .pipe(jsFilter.restore())
    .pipe(assets.restore())
    .pipe($.useref())
    .pipe(gulp.dest(config.build));
```

If we use a directive ng-strict-di, then it will give the proper error message if we do not specify our angular dependency explicitly.

Angular Dependency Injections

We can avoid mangling of angular dependency injecting by writing code using manual injection as per link http://jpapa.me/ngstyles or use gulp to provide security blanket by using gulp-ng-annotate plugin. It searches for dependency injection and add injection code if not found.

```
.pipe(jsLibFilter)
.pipe($.uglify())
.pipe(jsLibFilter.restore())
.pipe(jsAppFilter)
.pipe($.ngAnnotate())
.pipe($.uglify())
.pipe(jsAppFilter.restore())
.pipe(assets.restore())
.pipe($.useref())
```

It will add below code automatically:

```
e.getCustomers().then(function(t){return r.customers=t,r.customers})}function s(e)
{t.go("customer.detail",{id:e.id})}var r=this;r.customers=
[],r.gotoCustomer=s,r.title="Dashboard",n()}angular.module("app.dashboard").controlle
r("Dashboard",t),t.$inject=["$state","dataservice","logger"]}(),function(){"use
strict";function t(t){t.configureStates(e(),"/")}function e()
```

We can provide some hints where ng-annotate can miss the places, like embedded function like below:

```
function configureStateHelper() {
    var resolveAlways = { /* @ngInject */
        ready: function (dataservice) {
            return dataservice.ready();
        }
    };
```

Practice – using ng-annotate

Static Asset Revisions and Version Bumping

Use gulp-rev tool to rename the files with revision using a content hash. We can also use gulp-rev-replace to rewrite occurrences of filenames that were updated by gulp-rev.

Adding static asset revisions and replacements

```
    .pipe(jsAppFilter)
    .pipe($.ngAnnotate())
    .pipe($.uglify())
    .pipe(jsAppFilter.restore())
    .pipe($.rev())
    .pipe(assets.restore())
    .pipe($.useref())
    .pipe($.revReplace())
    .pipe(gulp.dest(config.build));
```

Using below we can generate a revision manifest file to see old file, and new file:

```
    .pipe(jsAppFilter)
    .pipe($.ngAnnotate())
    .pipe($.uglify())
    .pipe(jsAppFilter.restore())
    .pipe($.rev())
    .pipe(assets.restore())
    .pipe($.useref())
    .pipe($.revReplace())
    .pipe(gulp.dest(config.build));
```

Bumping versions with Server using tool gulp-bump:

```
gulp.task('bump', function() {
    var msg = 'Bumping versions';
    var type = args.type;
    var version = args.version;
    var options = {};
    if (version) {
        options.version = version;
        msg += ' to ' + version;
    } else {
        options.type = type;
        msg += ' for a ' + type;
    }
    log(msg);
    return gulp
        .src(config.packages)
        .pipe($.bump(options))
        .pipe(gulp.dest(config.root));
});
```

Practice – using gulp-rev, and gulp-bump tasks

Testing

Handling testing using gulp by creating automated test runner, viewing code coverage, and automatically run tests on code change.

Karma lets us hook up to multiple different kinds of testing frameworks like QUnit, or jasmine, or Mocha.

In karma config, preprocessors generate some of the coverage information, and then reporters are the ones that are going to help you write out the reportage of code coverage that you have.

```
        coverage: {
            dir: report + 'coverage',
            reporters: [
                {type: 'html', subdir: 'report-html'},
                {type: 'lcov', subdir: 'report-lcov'},
                {type: 'text-summary'}
            ]
        },
        preprocesors: {}
    };
    options.preprocesors[clientApp + '**/!(*.spec)+(.js)'] = ['coverage'];
    return options;
}
```

```
{} gulp.config.js    {} gulpfile.js    {} karma.conf.js

11
12          // list of files / patterns to load in the browser
13          files: gulpConfig.karma.files,
14
15          // list of files to exclude
16          exclude: gulpConfig.karma.exclude,
17
18          proxies: {
19              '/': 'http://localhost:8888/'
20          },
21
22          // preprocess matching files before serving them to the browser
23          // available preprocessors: https://npmjs.org/browse/keyword/karma-pr
24          preprocessors: gulpConfig.karma.preprocessors,
25
26          // test results reporter to use
27          // possible values: 'dots', 'progress', 'coverage'
28          // available reporters: https://npmjs.org/browse/keyword/karma-report
29          reporters: ['progress', 'coverage'],
30
31          coverageReporter: {
32              dir: gulpConfig.karma.coverage.dir,
33              reporters: gulpConfig.karma.coverage.reporters
34          },
35
36          // web server port
37          port: 9876,
38
```

```
function startTests(singleRun, done) {
    var karma = require('karma').server;
    var excludeFiles = [];
    var serverSpecs = config.serverIntegrationSpecs; //TODO

    excludeFiles = serverSpecs;

    karma.start({
        config: __dirname + '/karma.conf.js',
        exclude: excludeFiles,
        single: !!singleRun
    }, karmaCompleted);

    function karmaCompleted(karmaResult) {
        log('Karma completed!');
        if (karmaResult === 1) {
            done('karma: tests failed with code ' + karmaResult);
        } else {
            done();
        }
    }
}
```

Practice – creating gulp task for karma test single time

Using wiredep we can get bower files:

```
var wiredep = require('wiredep');
var bowerFile = wiredep({devDependencies: true})['js'];
```

We need to install test related package like karma, sign-on, Mocha, Phantom JS to test our project. Karma, and related testing packages:

```
npm install --save-dev karma karma-chai karma-chai-sinon
karma-chrome-launcher karma-coverage karma-growl-reporter
karma-mocha karma-phantomjs-launcher karma-sinon  mocha
mocha-clean sinon-chai sinon phantomjs
```

Mocha and Chai are testing framework and assertion library, Sinon is a stubbing and mocking framework.

Continue running tests during development:

```
gulp.task('autotest', ['vet', 'templatecache'], function(done) {
    startTests(false /* singleRun */, done);
});
```

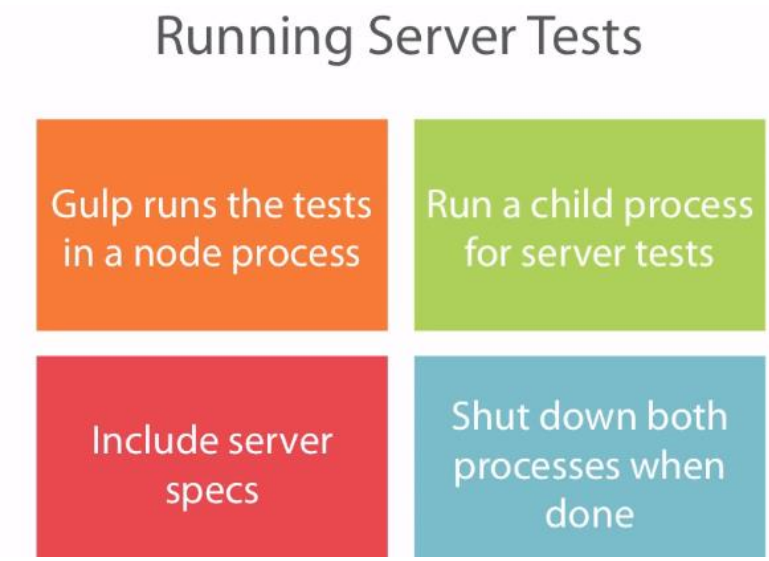
Practice – creating gulp task for karma test auto run

Integration Testing and HTML Test Runners

To run server integration tests we will crank up a second process inside of gulp, it will run the back-end server, so that the test running the first process can hit the back-end server and then run our

tests. Instead of viewing test into terminal by Karma we can also view them into HTML for better view.

Checklist for running integration tests:




```
var fork = require('child_process').fork;
var child = fork({
    nodeServer: './src/server/app.js',
    defaultPort: '7203'
});
```

Practice – running tests that require a node server

Setting up an HTML test runner task: setting up specs.html file:

```
<div id="mocha"></div>

    <!-- inject:testlibraries:js -->
    <!-- endinject -->

    <script>
        expect = chai.expect;
        AssertionError = chai.AssertionError;
        mocha.setup('bdd');
        mocha.traceIgnores = ['mocha.js', 'chai.js', 'angular.js']
    </script>

    <!-- bower:js -->
    <!-- endbower -->

    <!-- inject:js -->
    <!-- endinject -->

    <!-- inject:spechelpers:js -->
    <!-- endinject -->

    <!-- inject:specs:js -->
    <!-- endinject -->

    <!-- inject:templates:js -->
    <!-- endinject -->

    <script>
        mocha.run();
    </script>
```

```
gulp.task('build-specs', ['templatecache'], function() {
    log('building the spec runner');

    var wiredep = require('wiredep').stream;
    var options = config.getWiredepDefaultOptions();

    return gulp
        .src(config.specRunner)
        .pipe(wiredep(options))
        .pipe($.inject(gulp.src(config.testlibraries),
            {name: 'inject:testlibraries', read: false}))

        .pipe($.inject(gulp.src(config.js)))

        .pipe($.inject(gulp.src(config.specHelpers),
            {name: 'inject:spechelpers', read: false}))

        .pipe($.inject(gulp.src(config.specs),
            {name: 'inject:specs', read: false}))

        .pipe($.inject(gulp.src(config.temp + config.templateCache.file),
            {name: 'inject:templates', read: false}))

        .pipe(gulp.dest(config.client));
});
```

```
gulp.task('serve-specs', ['build-specs'], function(done) {
    log('run the spec runner');
    serve(true /* isDev */, true /* specRunner */);
    done();
});
```

While doing specs build, we can specify in wiredep explicitly to include dev dependencies like below:

```
gulp.task('build-specs', ['templatecache'], function() {
    log('building the spec runner');

    var wiredep = require('wiredep').stream;
    var options = config.getWiredepDefaultOptions();
    options.devDependencies = true;
```

If we use browsersync, then if we do changes in our spec file the browser will load automatically with updated test cases.

Practice – setting up an HTML test runner gulp task

Running server tests in the HTML test runner:

```
gulp.task('build-specs', ['templatecache'], function() {
    log('building the spec runner');

    var wiredep = require('wiredep').stream;
    var options = config.getWiredepDefaultOptions();
    var specs = config.specs;

    options.devDependencies = true;

    if (args.startServers) {
        specs = [].concat(specs, config.serverIntegrationSpecs);
    }
```

https://github.com/johnpapa/angularjs-styleguide#testing

Migrating to Gulp 4

We can use a new task engine gulp.series() where we can pass in a set of functions or tasks and run those in a series one after the another which is not easy to do in Gulp 3 as it runs everything in parallel. This function accepts a set of functions or task (strings).

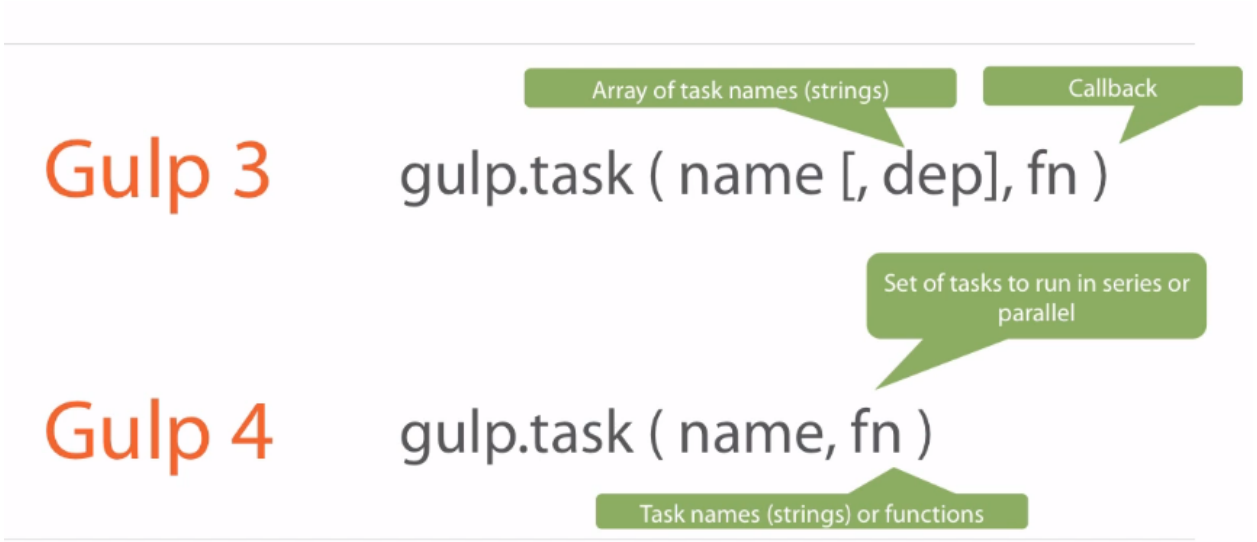
```
gulp.task('styles', gulp.series('clean-styles', styles));

function styles() {
    return gulp
        .src(config.less)
```

For parallelism we can use gulp.paraller() function.

```
gulp.task('assets', gulp.parallel('fonts', 'styles', 'images'));
```

We can mix both of these functions:

```
gulp.task('build',
            gulp.series(
                gulp.parallel('vet', 'test'),
                gulp.parallel('wiredep', 'styles', 'templatecache'),
                optimize
            )
        );
```




```
gulp.task('inject', gulp.series(
    gulp.parallel('wiredep', 'styles', 'templatecache'),
    function() {
        log('Wire up the app css into the html, and call wiredep ');

        return gulp
            .src(config.index)
            .pipe($.inject(gulp.src(config.css)))
            .pipe(gulp.dest(config.client));
    }))
);
```

Practice – migrating from gulp 3 to gulp 4

We can use command gulp –tasks-simple to see all the task list, to show the task tree type gulp –tasks command.