

Tools

Angular JS Course day 01

Edgar Bentkovskyi,
SoftServe Software Engineer,
August 2016

Agenda

1. JS Tools and CI Overview
2. Grunt
3. Gulp
4. npm/bower
5. Module Bundlers. WebPack

1. JS Tools and CI Overview

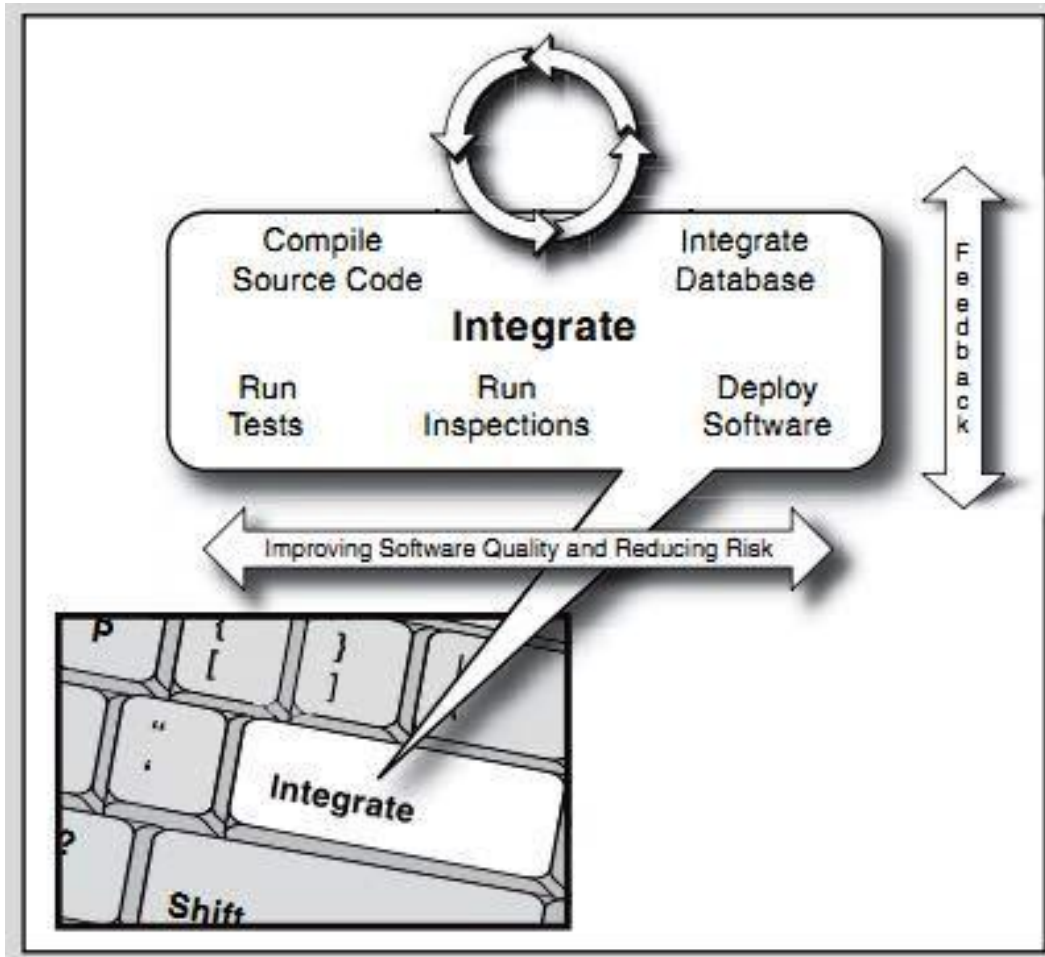
Continuous Integration is ...

... a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible



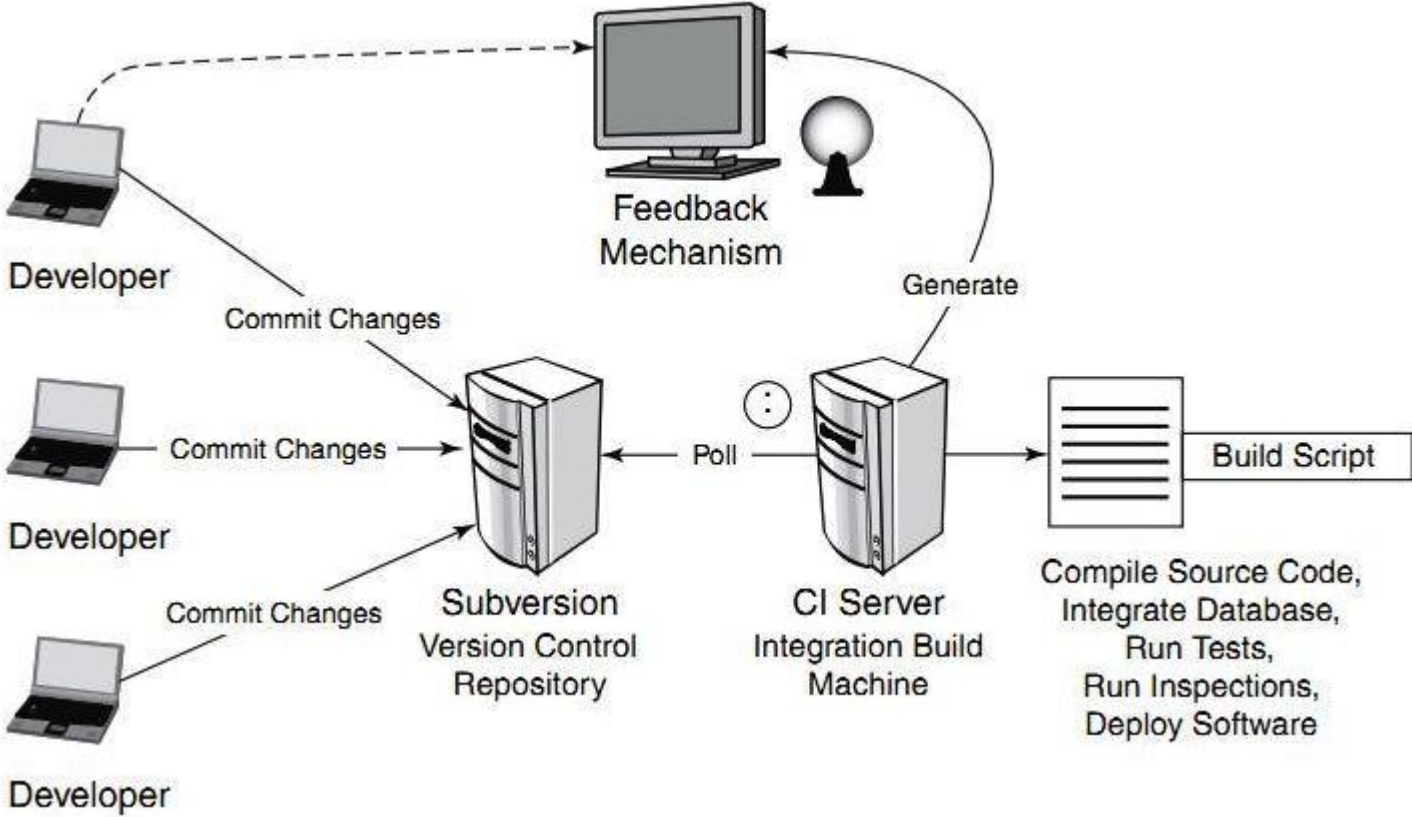
Martin Fowler

The Integrate Button



CI is a process that consists of **continuously** compiling, testing, inspecting and deploying source code

CI Workflow



Continuous Delivery & Continuous Deployment

CONTINUOUS DELIVERY



CONTINUOUS DEPLOYMENT



Travis CI

The image shows a browser window displaying the Travis CI website. The main heading is "Test and Deploy with Confidence" with the subtext "Easily sync your GitHub projects with Travis CI and you'll be testing your code in minutes!". A prominent green "Sign Up" button is centered below the text.

Below the main content, there is a screenshot of the Travis CI user interface for a repository named "green-eggs/ham". The interface shows a "build passing" status. On the left, a "My Repositories" list includes "green-egg/ham" (build #22), "one-fish/two-fish" (build #2686), and "hop-on/pop" (build #7001). The main area displays the current build details for "green-eggs/ham", including a commit message "adding in Oh the places you'll go! You'll be on your way up! You'll be seeing great sights!" by Sven Fuchs. Summary statistics show "# 209 passed", "Commit d019f29", "Compare 88f312a..d019f29", "ran for 53 sec", and "about 2 hours ago". A terminal log at the bottom shows the start of a build system information section.

Activation Steps



1 Activate GitHub Repositories

Once you're signed in, and we've initially synchronized your repositories from GitHub, go to your [profile](#) page for open source or for your private projects.

You'll see all the organizations you're a member of and all the repositories you have access to. The ones you have administrative access to are the ones you can enable the service hook for.

Flip the switch to on for all repositories you'd like to enable.

2 Add .travis.yml file to your repository

In order for Travis CI to build your project, you need to tell the systems a little bit about it. You'll need to add a file named `.travis.yml` to the root of your repository.

If `.travis.yml` is not in the repository, is misspelled or is not valid YAML, Travis CI will ignore it.

NOTE: The `language` value is case-sensitive. If you set `language: c`, for example, your project will be considered a Ruby project.

Here you can find some of our basic [language examples](#).



3 Trigger your first build with a git push

Once the GitHub hook is set up, push your commit that adds `.travis.yml` to your repository. That should add a build into one of the queues on Travis CI and your build will start as soon as one worker for your language is available.

To start a build, perform one of the following:

- Commit and push something to your repository
- Go to your repository's settings page, click on "Webhooks & Services" on the left menu, choose "Travis CI" in the "Services", and use the "Test service" button.

NOTE: You cannot trigger your first build using Test Hook button. It has to be triggered by a push to your repository.

Sample config

<https://github.com/ITsvetkoFF/Kv-012/blob/master/.travis.yml>

branches:

only:

- master

language: node_js

node_js:

- "4.1"

cache:

directories:

- TCMSApp/node_modules
- TCMSApp/bower_components

before_script:

- cd TCMSApp/
- npm install codecov.io
- npm install -b bower
- npm install -g gulp
- npm install

script:

- gulp test

after_script:

- cat ./report/coverage/report-lcov/lcov.info | ./node_modules/codecov.io/bin/codecov.io.js

Tools: linters

JSCS



The image shows a browser window with the URL `jscs.info`. The page features a yellow square logo with the text "JSCS" and "-- ^" below it. The main heading is "JSCS — JavaScript Code Style". Below the heading are navigation links: "Overview", "Rules", "Contributing", and "Changelog". The main text describes JSCS as a code style linter/formatter for enforcing style guides, mentioning 150 validation rules and presets for jQuery, Airbnb, and Google. A code block shows the command `npm install jscs -g`. At the bottom, there are several status indicators: "npm v2.10.1", "build failing", "coverage 89%", "dependencies up to date", "devDependencies up to date", "gitter", and "join chat".

JSCS

JSCS — JavaScript Code Style

[Overview](#) [Rules](#) [Contributing](#) [Changelog](#)

JSCS is a code style linter/formatter for programmatically enforcing your style guide. You can configure JSCS for your project/company using over 150 validation rules, including presets from popular style guides like jQuery, Airbnb, Google, and more.

```
npm install jscs -g
```

npm v2.10.1 build failing coverage 89% dependencies up to date
devDependencies up to date gitter join chat

Packages

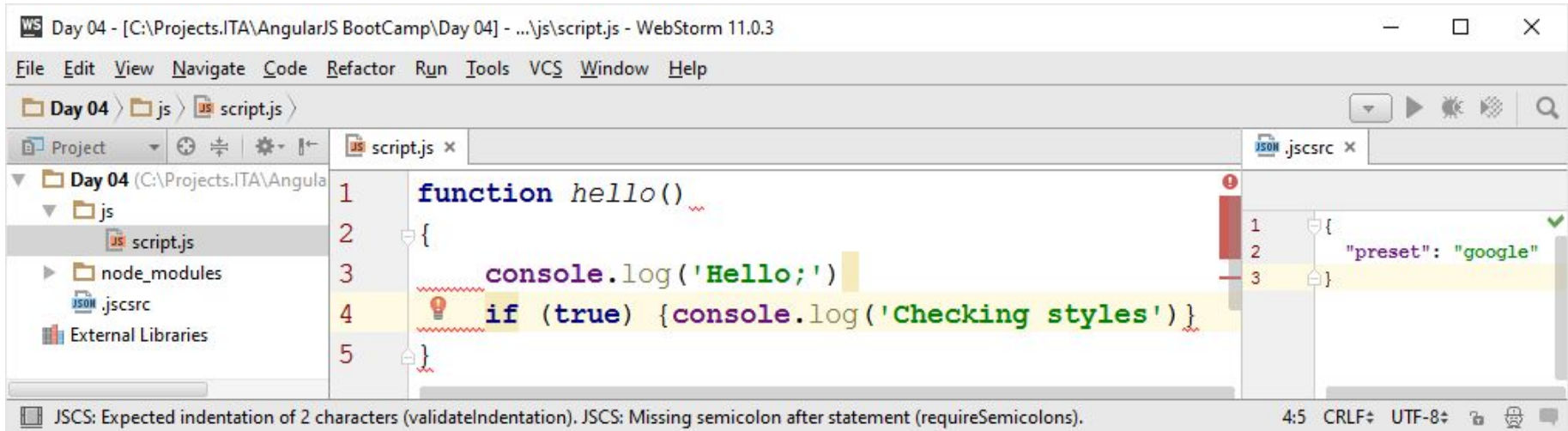
- Atom plugin: <https://atom.io/packages/linter-jscs>
- Brackets Extension: <https://github.com/globexdesigns/brackets-jscs>
- Grunt task: <https://github.com/jscs-dev/grunt-jscs/>
- Gulp task: <https://github.com/jscs-dev/gulp-jscs/>
- Overcommit Git pre-commit hook manager: <https://github.com/brigade/overcommit/>
- SublimeText 3 Plugin: <https://github.com/SublimeLinter/SublimeLinter-jscs/>
- Syntastic VIM Plugin: https://github.com/scrooloose/syntastic/.../syntax_checkers/javascript/jscs.vim/
- Web Essentials for Visual Studio 2013: <https://github.com/madskristensen/WebEssentials2013/>
- IntelliJ IDEA, RubyMine, WebStorm, PhpStorm, PyCharm plugin: <https://www.jetbrains.com/webstorm/help/jscs.html>
- Visual Studio Code extension: <https://github.com/microsoft/vscode-jscs>

Presets

- Note: the easiest way to use a preset is with the preset option described below.
- Airbnb — <https://github.com/airbnb/javascript>
- Crockford — <http://javascript.crockford.com/code.html>
- Google — <https://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml>
- Grunt — <http://gruntjs.com/contributing#syntax>
- Idiomatic — <https://github.com/rwaldron/idiomatic.js#idiomatic-style-manifesto>
- jQuery — <https://contribute.jquery.org/style-guide/js/>
- MDCS — <https://github.com/mrdoob/three.js/wiki/Mr.doob's-Code-Style™>
- node-style-guide - <https://github.com/felixge/node-style-guide>
- Wikimedia — https://www.mediawiki.org/wiki/Manual:Coding_conventions/JavaScript
- WordPress — <https://make.wordpress.org/core/handbook/coding-standards/javascript/>
- Yandex — <https://github.com/yandex/codestyle/blob/master/javascript.md>

```
{  
  "preset": "jquery",  
  "requireCurlyBraces": null // or false  
}
```

WebStorm Sample



The screenshot shows the WebStorm IDE interface. The main editor displays a JavaScript file named `script.js` with the following code:

```
1 function hello()  
2 {  
3     console.log('Hello;')  
4     if (true) {console.log('Checking styles')}  
5 }
```

The code is highlighted with a yellow background. A red squiggly line is visible under the closing curly brace of the `if` statement on line 4, indicating a missing semicolon. A lightbulb icon is also present next to the `if` statement, suggesting an IDE suggestion.

The right-hand pane shows a JSON file named `.jscsrc` with the following content:

```
1 {  
2     "preset": "google"  
3 }
```

The JSON file is also highlighted with a yellow background. A green checkmark is visible in the right margin of the JSON file, indicating it is valid.

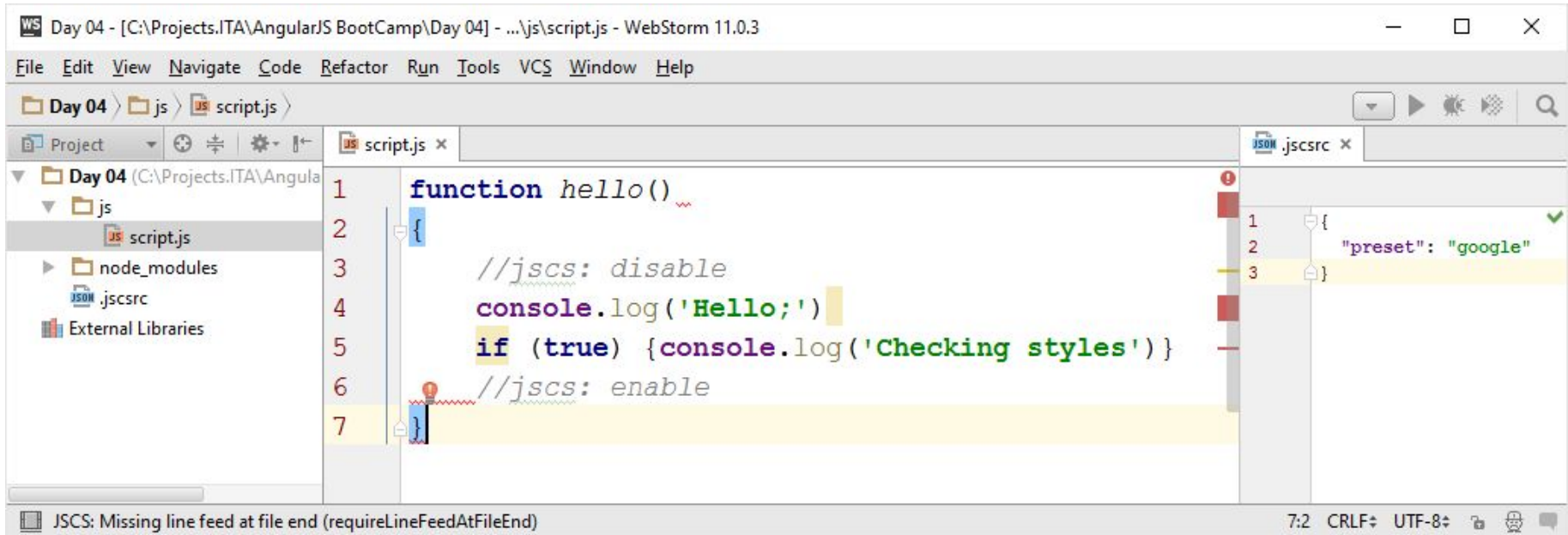
The status bar at the bottom of the IDE displays the following messages:

```
JSCS: Expected indentation of 2 characters (validateIndentation). JSCS: Missing semicolon after statement (requireSemicolons).
```

The status bar also shows the current cursor position as `4:5`, the encoding as `CRLF`, and the character set as `UTF-8`.

WebStorm Sample

- Disabling/Enabling in Code



The screenshot shows the WebStorm IDE interface. The main editor displays a JavaScript file named `script.js` with the following code:

```
1 function hello()  
2 {  
3     //jscs: disable  
4     console.log('Hello;')  
5     if (true) {console.log('Checking styles')}  
6     //jscs: enable  
7 }
```

The code is highlighted in yellow, and there are red squiggly lines under the `//jscs: disable` and `//jscs: enable` comments. The left sidebar shows the project structure with `script.js` selected. The right sidebar shows the `.jscsrc` file with the following configuration:

```
1 {  
2     "preset": "google"  
3 }
```

The status bar at the bottom indicates a JSCS error: "JSCS: Missing line feed at file end (requireLineFeedAtFileEnd)". The status bar also shows the current file encoding as UTF-8 and the line ending as CRLF.

A Comparison of JavaScript Linting Tools

- <https://www.sitepoint.com/comparison-javascript-linting-tools/>



Practice Task [homework]

- Install linter into your IDE
- Write correct/incorrect code, check linter output
- Try different styles
- Try options to disable warnings (in config file or in code directly)

2. Grunt

What is GRUNT?

JavaScript Task runner

- Cross-platform
- Works by executing tasks

Used for

- Develop
- Build
- Deploy



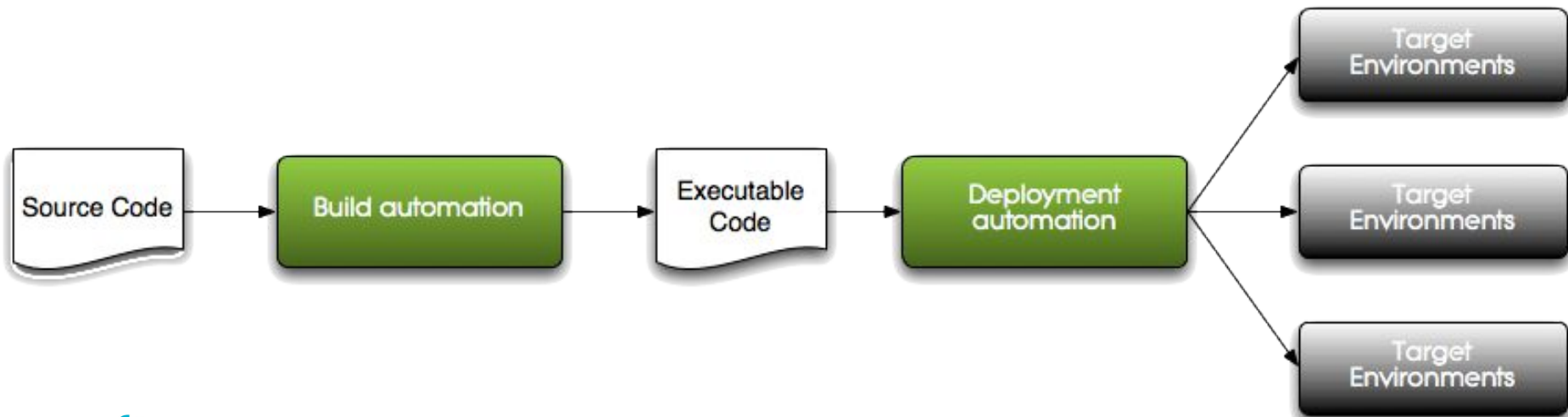
GRUNT JS & Automation

Enables team to write consistent code

Maintain coding standards within teams

Automate your build process

Automate testing and deployment and release process



Install

Install Node.js (with `npm!!!`)

Install Grunt

- `npm install -g grunt-cli`
- In the project directory (root level):
 - create file `package.json` or use `npm init`
 - Add Grunt as dev dependency
`npm install grunt --save-dev`
 - Create file `gruntfile.js`



Easy To Install



`package.json`

Plugins

In official Grunt site we find out that 5,829 plugins are available for Grunt (yesterday)

To use any plugin in project it have to added into the **package.json** manually or with npm

```
npm install <plugin> --save-dev
```



gruntfile.js

The **gruntfile.js** or **gruntfile.coffee** file is a valid JavaScript or CoffeeScript file that belongs in the root directory of your project.

A gruntfile is comprised of the following parts:

- The "wrapper" function
- Project and task configuration
- Loading Grunt plugins and tasks
- Custom tasks



Gruntfile.js

gruntfile.js

Every gruntfile starts out with some boilerplate code.

```
module.exports = function(grunt) {  
    // Our tasks  
}
```

First task

For create new task `grunt.registerTask()` is used

Task should to have a name and have to associated with callback function.

```
grunt.registerTask("default", function() {  
    console.log("Hello World from GRUNT");  
});
```

Save file ***gruntfile.js*** and run **grunt**

Run GRUNT at first time

```
gruntfile.js
1  module.exports = function(grunt) {
2      grunt.registerTask("default", function() {
3          console.log("Hello World from GRUNT");
4      });
5  }
```

gruntfile.js

```
MINGW32:/d/SS_ITA_Work/Groups/IF-054.Web-UI/GRUNT
ybezgach@IF151 /d/SS_ITA_Work/Groups/IF-054.web-UI/GRUNT
$ grunt
Running "default" task
Hello World from GRUNT

Done, without errors.

ybezgach@IF151 /d/SS_ITA_Work/Groups/IF-054.web-UI/GRUNT
$
```

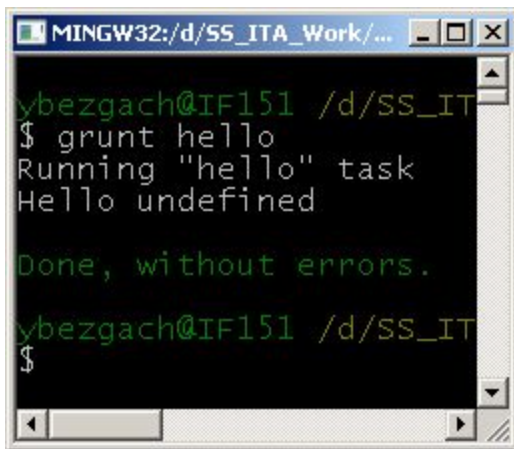
Output
[Result]

When we run **grunt** without parameters it will find the *default* task definition and run it

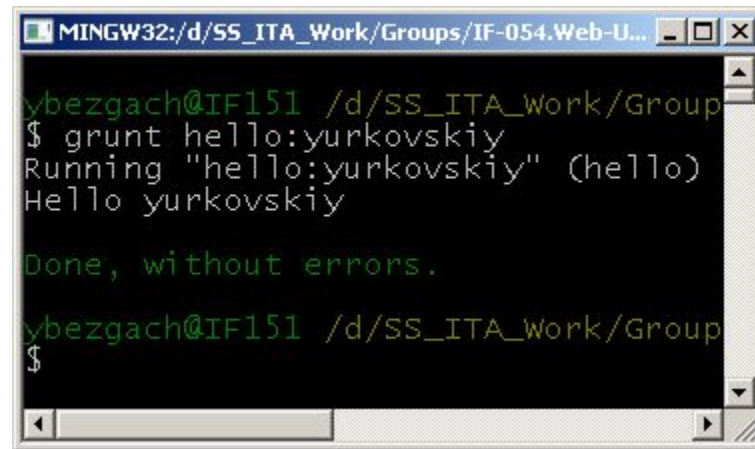
Tasks with parameters

The 'hello' task is defined as:

```
grunt.registerTask("hello", function (who) {  
    grunt.log.writeln("Hello " +who);  
});
```



```
MINGW32:/d/SS_ITA_Work/...  
ybezgach@IF151 /d/SS_ITA...  
$ grunt hello  
Running "hello" task  
Hello undefined  
  
Done, without errors.  
ybezgach@IF151 /d/SS_ITA...  
$
```



```
MINGW32:/d/SS_ITA_Work/Groups/IF-054.Web-U...  
ybezgach@IF151 /d/SS_ITA_Work/Group...  
$ grunt hello:yurkovskiy  
Running "hello:yurkovskiy" (hello)  
Hello yurkovskiy  
  
Done, without errors.  
ybezgach@IF151 /d/SS_ITA_Work/Group...  
$
```

Check parameters

The 'div' task is defined as

```
gruntfile.js x
1  module.exports = function(grunt) {
2      grunt.registerTask("div", function(a, b) {
3          if (isNaN(Number(a)) || isNaN(Number(b))) {
4              grunt.warn("The input parameters have to be a numbers!!!");
5          }
6          if (b == 0) {
7              grunt.warn("[ERROR] the second parameter couldn't be a 0")
8          }
9          grunt.log.writeln(a + " / " + b + " Answer: " + a/b);
10     });
11 }
```

Try to run grunt

```
$grunt div:aa:bb
```

```
$grunt div:1:0
```

```
$grunt div:10:2
```

Chaining tasks

Grunt has an ability to create one task that fires off other tasks

To make a task like this we use `registerTask()` and pass it an array of tasks instead of a callback function.

```
grunt.registerTask("default",  
["hello:yurkovskiy", "div:10:5"]);
```



Multitasks

One task many outputs

A multi task is a task that implicitly iterates over all of its named sub-properties



```
grunt.task.registerMultiTask(taskName,  
description, taskFunction)
```

Working with files and folders

Grunt has a **file** object which consist a lot of properties and methods for working with files and directories

Methods	Properties
<code>grunt.file.mkdir</code>	<code>grunt.file.defaultEncoding</code>
<code>grunt.file.delete</code>	<code>grunt.file.preserveBOM</code>
<code>grunt.file.copy</code>	
<code>grunt.file.read</code>	
<code>grunt.file.readJSON</code>	
<code>grunt.file.write</code>	

**more info on*

softserve experience matters <http://gruntjs.com/api/grunt.file>

The most useful plugins

contrib-watch

contrib-jshint

contrib-clean

contrib-uglify

contrib-copy

contrib-cssmin

contrib-less

- contrib-coffee
- contrib-htmlmin
- contrib-sass
- contrib-compress
- shell
- usemin
- contrib-jasmine



How to use plugins

Install plugin dependency

```
npm install <plugin-name> --save-dev
```

Write task definition

```
grunt.initConfig({<plugin>: {  
  <definition>  
}, ...})
```

Load task

```
grunt.loadNpmTasks("<plugin-name>");
```

Example

Using **contrib-uglify** plugin

```
gruntfile.js  x  input.js  x
1  module.exports = function(grunt) {
2      grunt.initConfig({
3          uglify: {
4              options: {
5                  mangle: true
6              },
7              my_target: {
8                  files: {
9                      'dest/output.min.js': ['src/input.js']
10                 }
11             }
12         }
13     });
14
15     grunt.loadNpmTasks("grunt-contrib-uglify");
16
17
18     grunt.registerTask("build", [
19         "uglify",
20     ]
21 );
22 }
```

```
gruntfile.js  x  input.js  x
1  function add(number1, number2) {
2      return (number1 + number2);
3  }
```

```
MINGW32:/d/SS_ITA_Work/Groups/IF-054.Web-UI/GRUNT
ybezgach@IF151 /d/SS_ITA_Work/Groups/IF-0
$ grunt build
Running "uglify:my_target" (uglify) task
>> 1 file created.

Done, without errors.

ybezgach@IF151 /d/SS_ITA_Work/Groups/IF-0
$
```

```
output.min.js  x
1  function add(a,b){return a+b}
```

3. Gulp

What is GULP?

JavaScript Task runner

- Cross-platform
- Works by executing tasks

Used for

- Develop
- Build
- Deploy

Install

Install Node.js (with `npm!!!`)

Install Gulp globally

- `npm install -g gulp`
- In the project directory (root level):
 - create file `package.json` or use `npm init`
 - Install Gulp as dev dependency
`npm install gulp --save-dev`
 - Create file `gulpfile.js`



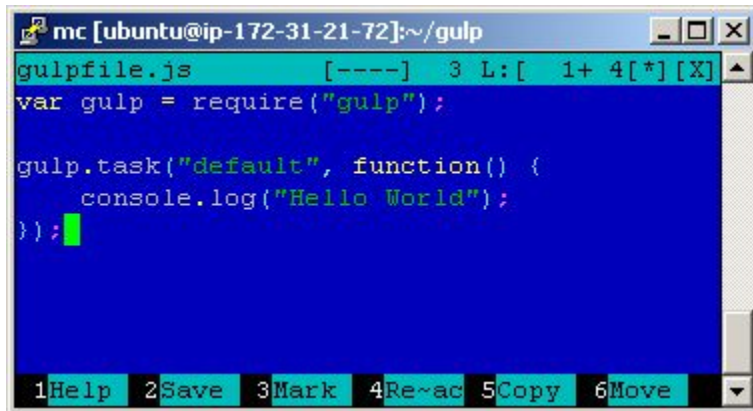
Easy To Install



package.json

Define a task

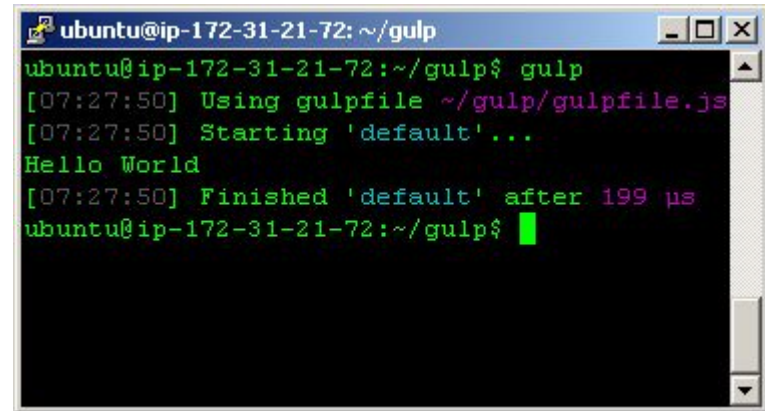
```
var gulp = require("gulp");  
gulp.task("default", function() {  
    // code for task  
});
```



A screenshot of a text editor window titled 'mc [ubuntu@ip-172-31-21-72]:~/gulp'. The editor shows the following code in 'gulpfile.js':

```
var gulp = require("gulp");  
  
gulp.task("default", function() {  
    console.log("Hello World");  
});
```

The editor has a status bar at the bottom with menu items: 1Help, 2Save, 3Mark, 4Re~ac, 5Copy, 6Move.



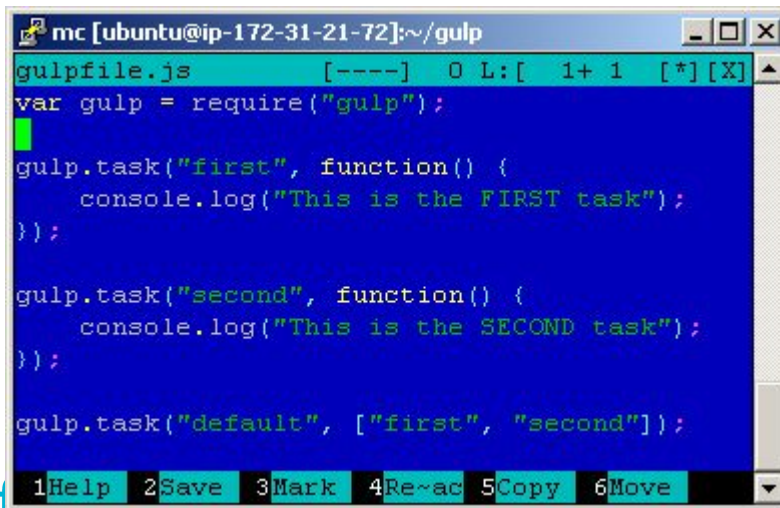
A screenshot of a terminal window titled 'ubuntu@ip-172-31-21-72: ~/gulp'. The terminal shows the execution of the 'gulp' command:

```
ubuntu@ip-172-31-21-72:~/gulp$ gulp  
[07:27:50] Using gulpfile ~/gulp/gulpfile.js  
[07:27:50] Starting 'default'...  
Hello World  
[07:27:50] Finished 'default' after 199 μs  
ubuntu@ip-172-31-21-72:~/gulp$
```

Task series, dependency

For create series of tasks we need to do next steps

- give it a hint to tell it when the task is done,
- and give it a hint that a task depends on completion of another.

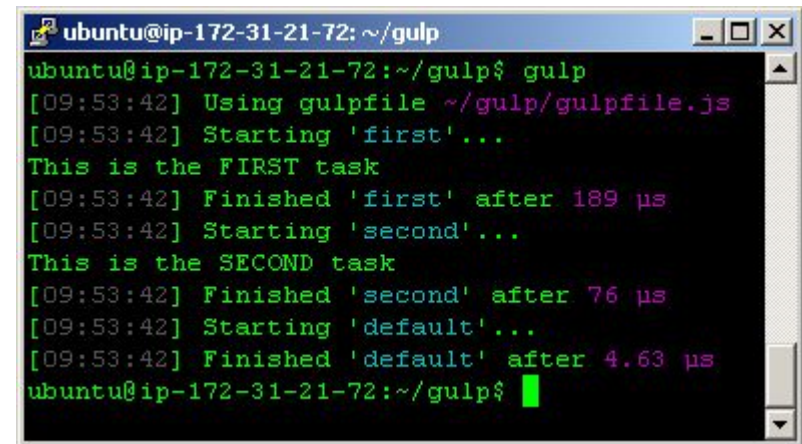


```
mc [ubuntu@ip-172-31-21-72]:~/gulp
gulpfile.js [----] 0 L:[ 1+ 1 [*] [X]
var gulp = require("gulp");

gulp.task("first", function() {
  console.log("This is the FIRST task");
});

gulp.task("second", function() {
  console.log("This is the SECOND task");
});

gulp.task("default", ["first", "second"]);
```



```
ubuntu@ip-172-31-21-72: ~/gulp
ubuntu@ip-172-31-21-72:~/gulp$ gulp
[09:53:42] Using gulpfile ~/gulp/gulpfile.js
[09:53:42] Starting 'first'...
This is the FIRST task
[09:53:42] Finished 'first' after 189 µs
[09:53:42] Starting 'second'...
This is the SECOND task
[09:53:42] Finished 'second' after 76 µs
[09:53:42] Starting 'default'...
[09:53:42] Finished 'default' after 4.63 µs
ubuntu@ip-172-31-21-72:~/gulp$
```


Gulp API

`gulp.task(name[, deps], fn)`

`gulp.src(globs[, options])`

`gulp.dest(path[, options])`

`gulp.watch(glob[, opts], tasks)`

Plugins

In official Gulp site we find out that 1866 plugins are available for Gulp (Aug, 2015)

To use any plugin in project it have to added into the **package.json** manually or with npm

```
npm install <plugin> --save-dev
```



Common Gulp plugins

gulp-minify-css

gulp-uglify

gulp-concat

gulp-ng-annotate

gulp-ngdocs

gulp-ng-html2js

Usually plugins includes to the project using
var plugin = require("<plugin_name>");



Gulp pipe() function

Investigating pipes

```
gulpfile.js
1  var gulp = require("gulp");
2  var uglify = require("gulp-uglify");
3  var concat = require("gulp-concat");
4
5  gulp.task("default", function() {
6    return gulp.src("src/*.js")
7      .pipe(uglify())
8      .pipe(concat("funcs.js"))
9      .pipe(gulp.dest("dist"));
10 });
```

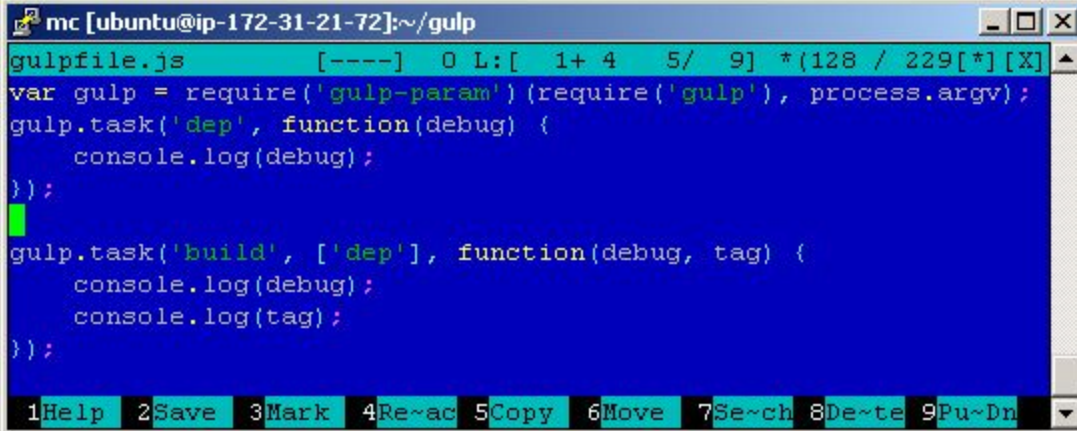


Command line arguments

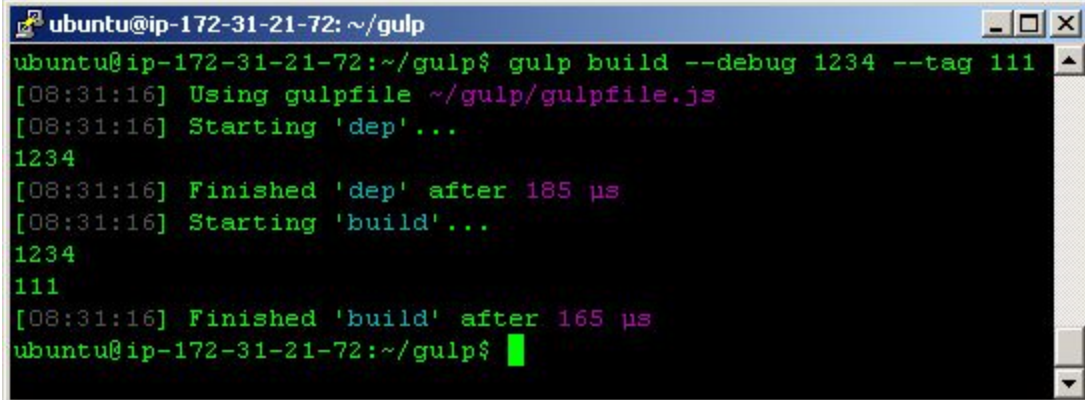
Gulp doesn't offer ability to pass parameters from command line

Plugins will help 😊

- yargs
- gulp-param

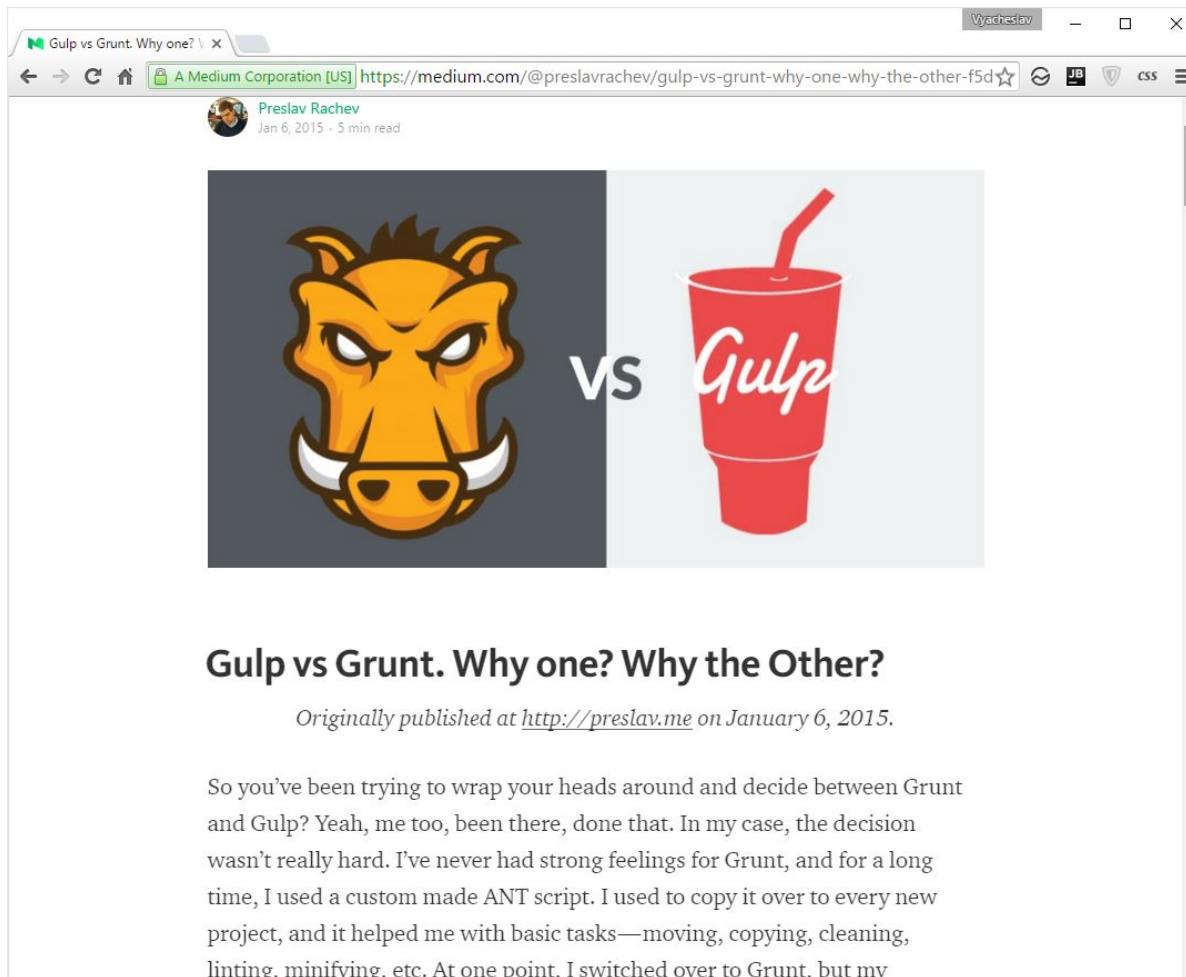


```
mc [ubuntu@ip-172-31-21-72]:~/gulp
gulpfile.js [----] 0 L:[ 1+ 4 5/ 9] *(128 / 229[*] [X]
var gulp = require('gulp-param')(require('gulp'), process.argv);
gulp.task('dep', function(debug) {
  console.log(debug);
});
gulp.task('build', ['dep'], function(debug, tag) {
  console.log(debug);
  console.log(tag);
});
1Help 2Save 3Mark 4Re~ac 5Copy 6Move 7Se~ch 8De~te 9Pu~Dn
```



```
ubuntu@ip-172-31-21-72: ~/gulp
ubuntu@ip-172-31-21-72:~/gulp$ gulp build --debug 1234 --tag 111
[08:31:16] Using gulpfile ~/gulp/gulpfile.js
[08:31:16] Starting 'dep'...
1234
[08:31:16] Finished 'dep' after 185 µs
[08:31:16] Starting 'build'...
1234
111
[08:31:16] Finished 'build' after 165 µs
ubuntu@ip-172-31-21-72:~/gulp$
```

Gulp vs Grunt



Gulp vs Grunt. Why one? Why the Other?

Originally published at <http://preslav.me> on January 6, 2015.

So you've been trying to wrap your heads around and decide between Grunt and Gulp? Yeah, me too, been there, done that. In my case, the decision wasn't really hard. I've never had strong feelings for Grunt, and for a long time, I used a custom made ANT script. I used to copy it over to every new project, and it helped me with basic tasks—moving, copying, cleaning, linting, minifying, etc. At one point, I switched over to Grunt, but my

<https://medium.com/@preslavrachev/gulp-vs-grunt-why-one-why-the-other-f5d3b398edc4#.jez2mtxgl>

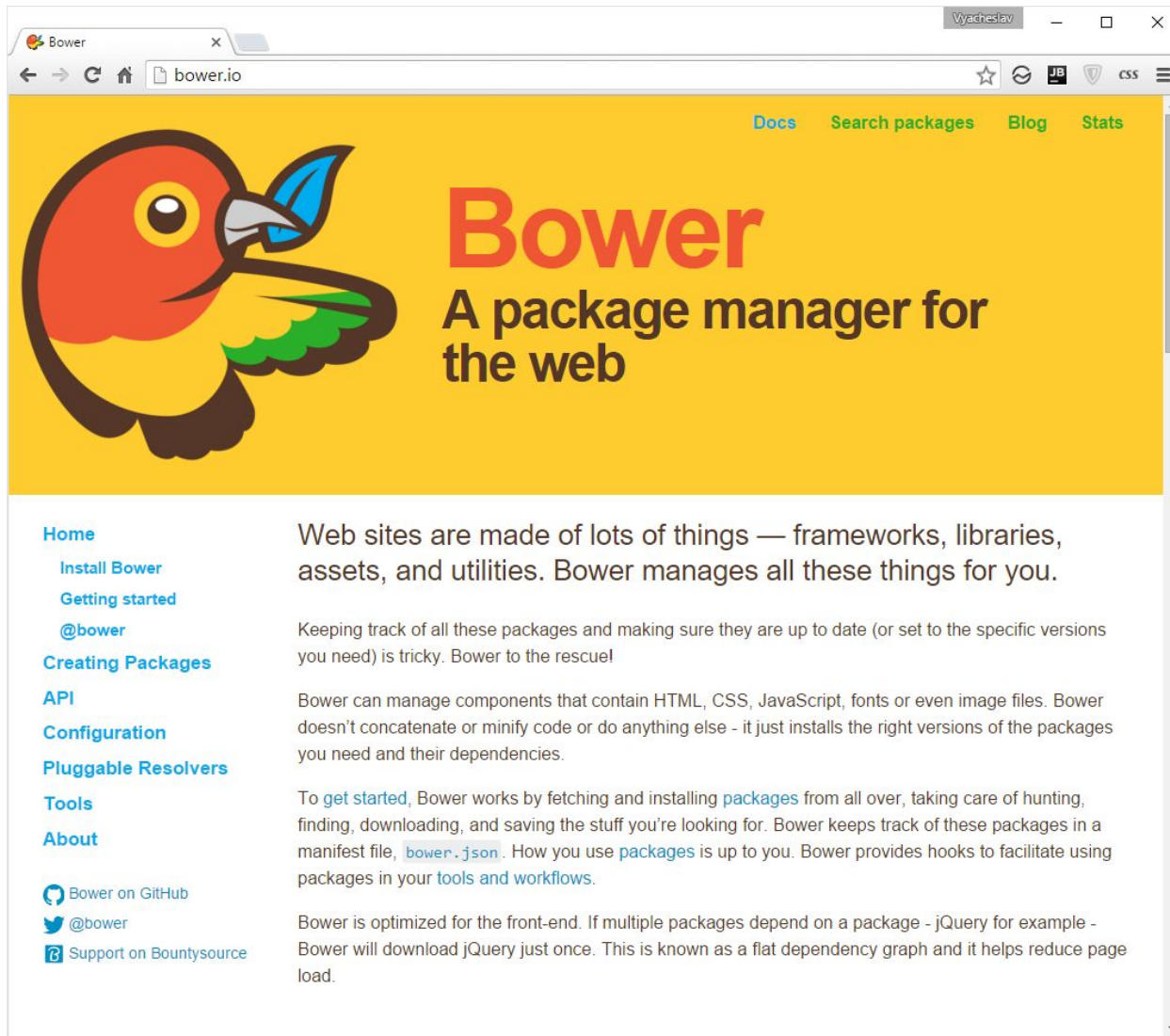
4. npm/bower

Intro to npm

The screenshot shows the npm website homepage. At the top, there is a navigation bar with links for "npm On-Site", "npm Private Packages", "npm Open Source", "documentation", and "support". Below this is a search bar with the text "find packages" and a "sign up or log in" button. The main heading reads "npm is the package manager for javascript." Below the heading are four statistics: 244,211 total packages, 140,914,634 downloads in the last day, 865,497,752 downloads in the last week, and 3,607,928,925 downloads in the last month. A section titled "packages people 'npm install' a lot" lists several popular packages:


Package Name	Description	Version	Published	Author
browserify	browser-side require() the node way	13.0.0	2 months ago	feross
grunt-cli	The grunt command line interface.	0.1.13	2 years ago	tkellen
bower	The browser package manager	1.7.7	4 weeks ago	sheerun
gulp	The streaming build system	3.9.0	9 months ago	phated
grunt				
express	Fast, unopinionated, minimalist web framework	4.13.4	1 month ago	dougwilson
npm	a package manager for JavaScript	3.7.1	3 weeks ago	iarna
cordova	Cordova command line interface tool	6.0.0	4 weeks ago	stevegill
forever	A simple CLI tool for ensuring that a given process stays alive	0.15.1	7 months ago	indexzero
less				
pm2	Production process manager for Node.js	1.0.0	2 months ago	tknew
karma	Spectacular Test Runner for JavaScript.	0.13.19	2 months ago	dignif...
coffee-script	Unfancy JavaScript	1.10.0	6 months ago	jashkenas
statsd	A simple, lightweight network daemon for monitoring	0.7.2	1 year ago	pkhzzrd
yo				

bower



The image shows a browser window displaying the Bower website. The browser's address bar shows "bower.io". The website has a yellow header with a navigation menu containing "Docs", "Search packages", "Blog", and "Stats". The main content area features a large illustration of a colorful bird (a parrot) on the left and the text "Bower A package manager for the web" on the right. Below this, there is a sidebar on the left with links for "Home", "Install Bower", "Getting started", "@bower", "Creating Packages", "API", "Configuration", "Pluggable Resolvers", "Tools", and "About". The main text area contains three paragraphs: the first describes Bower's purpose, the second explains how it works, and the third discusses its optimization for the front-end.

Docs Search packages Blog Stats



Bower

A package manager for the web

[Home](#)

- [Install Bower](#)
- [Getting started](#)
- [@bower](#)

[Creating Packages](#)

[API](#)

[Configuration](#)

[Pluggable Resolvers](#)

[Tools](#)

[About](#)

[Bower on GitHub](#)

[@bower](#)

[Support on Bountysource](#)

Web sites are made of lots of things — frameworks, libraries, assets, and utilities. Bower manages all these things for you.

Keeping track of all these packages and making sure they are up to date (or set to the specific versions you need) is tricky. Bower to the rescue!

Bower can manage components that contain HTML, CSS, JavaScript, fonts or even image files. Bower doesn't concatenate or minify code or do anything else - it just installs the right versions of the packages you need and their dependencies.

To [get started](#), Bower works by fetching and installing [packages](#) from all over, taking care of hunting, finding, downloading, and saving the stuff you're looking for. Bower keeps track of these packages in a manifest file, [bower.json](#). How you use [packages](#) is up to you. Bower provides hooks to facilitate using packages in your [tools](#) and [workflows](#).

Bower is optimized for the front-end. If multiple packages depend on a package - jQuery for example - Bower will download jQuery just once. This is known as a flat dependency graph and it helps reduce page load.

- What is bower?

- Bower is a package manager for the web
- Bower can manage components that contain HTML, CSS, JavaScript, fonts or even image files. Bower doesn't concatenate or minify code or do anything else - it just **installs the right versions of the packages** you need and their dependencies.
- Bower is a command line utility
- Bower required **npm** and **git**
- To install bower just type `npm install -g bower`

- ## Configuration

Bower can be configured using JSON in a `.bowerrc` file.

The config is obtained by merging multiple configurations by this order of importance:

- CLI arguments via `--config`
- Environment variables
- Local `.bowerrc` located in the current working directory
- All `.bowerrc` files upwards the directory tree
- `.bowerrc` file located in user's home folder (`~`)
- `.bowerrc` file located in the global folder (`/`)

- **Configuration parameters**

Detailed specifications of Bower configuration can be found here

<https://github.com/bower/spec/blob/master/config.md>

Definition of some of parameters

`directory` - The path in which installed components should be saved. If not specified this defaults to `bower_components`.

`proxy` - The proxy to use for http requests.

`timeout` - The timeout to be used when making requests in milliseconds, defaults to 60000 ms.

- **Install packages**

Install packages with `bower install`. Bower installs packages to `bower_components/`.

```
$ bower install [<options>]
```

```
$ bower install <endpoint> [<endpoint> ..]  
[<options>]
```

A package can be a GitHub shorthand, a Git endpoint, a URL, and more.

Project dependencies consist of:

- dependencies specified in `bower.json` of project
- All “external” dependencies not specified in `bower.json`, but present in `bower_components`
- Any additional `<endpoint>` passed as an argument to this command

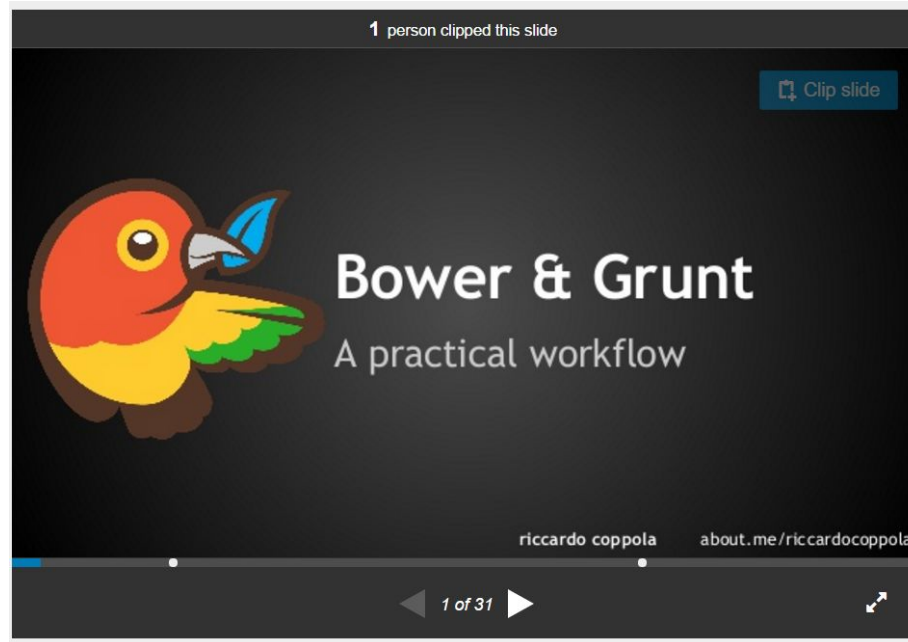
npm vs bower

	npm	Bower
What for	Commonly used for NodeJS modules	Front end asset package management
Under the hood	Nested dependency tree	Flat dependency tree
When to use	Great for the server, space is not a concern	Great for the front end, optimal size
Gotcha	No dependency conflict	Can have dependency conflict, when that happens, you need to be CREATIVE! Ask around 😊
File	package.json	bower.json
command	npm install <package_name>	bower install <package_name>

Task runners in Visual Studio 2015

- <https://blogs.msdn.microsoft.com/webdev/2016/01/06/task-runners-in-visual-studio-2015/>

Bower and Grunt – practical workflow



<http://www.slideshare.net/coppolariccardo/bower-grunt-a-practical-workflow>

5. Module Bundlers. WebPack

Why We Need Module Bundlers?

Difficulties of modern web-development:

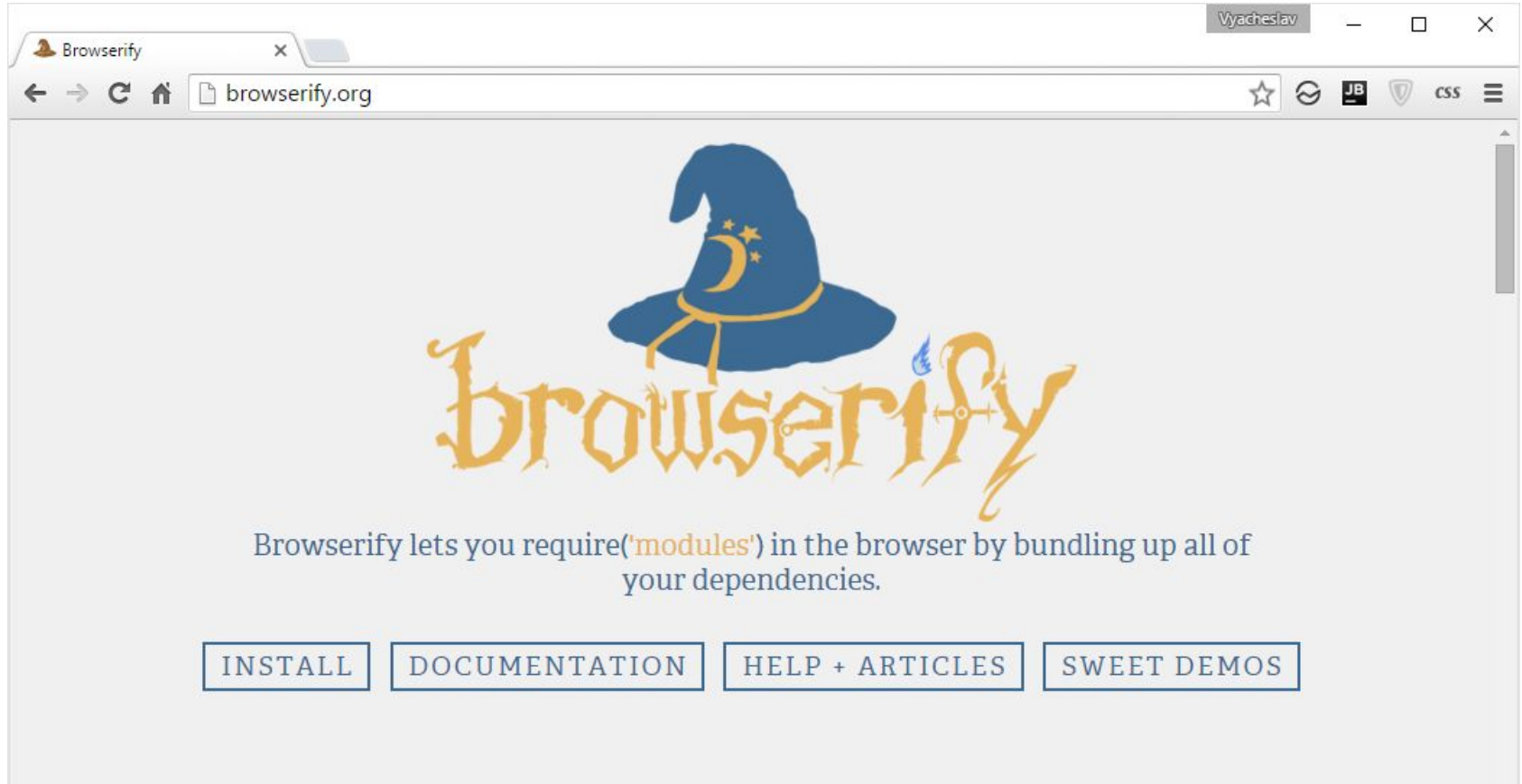
1. Different solutions (jQuery, Underscore, Knockout, Angular JS...)
2. Multiple versions (different versions of jQuery, Bootstrap...)
3. Pre-processing formats (less/sass/stylus, handlebars/jade/ejs, CoffeeScript/TypeScript/ES2015...)

What we need:

1. Modularity and isolation of a code
2. Safely connect third-party solutions
3. Use different version of libraries
4. Combine fragments into limited set of files

Browserify

<http://browserify.org/>

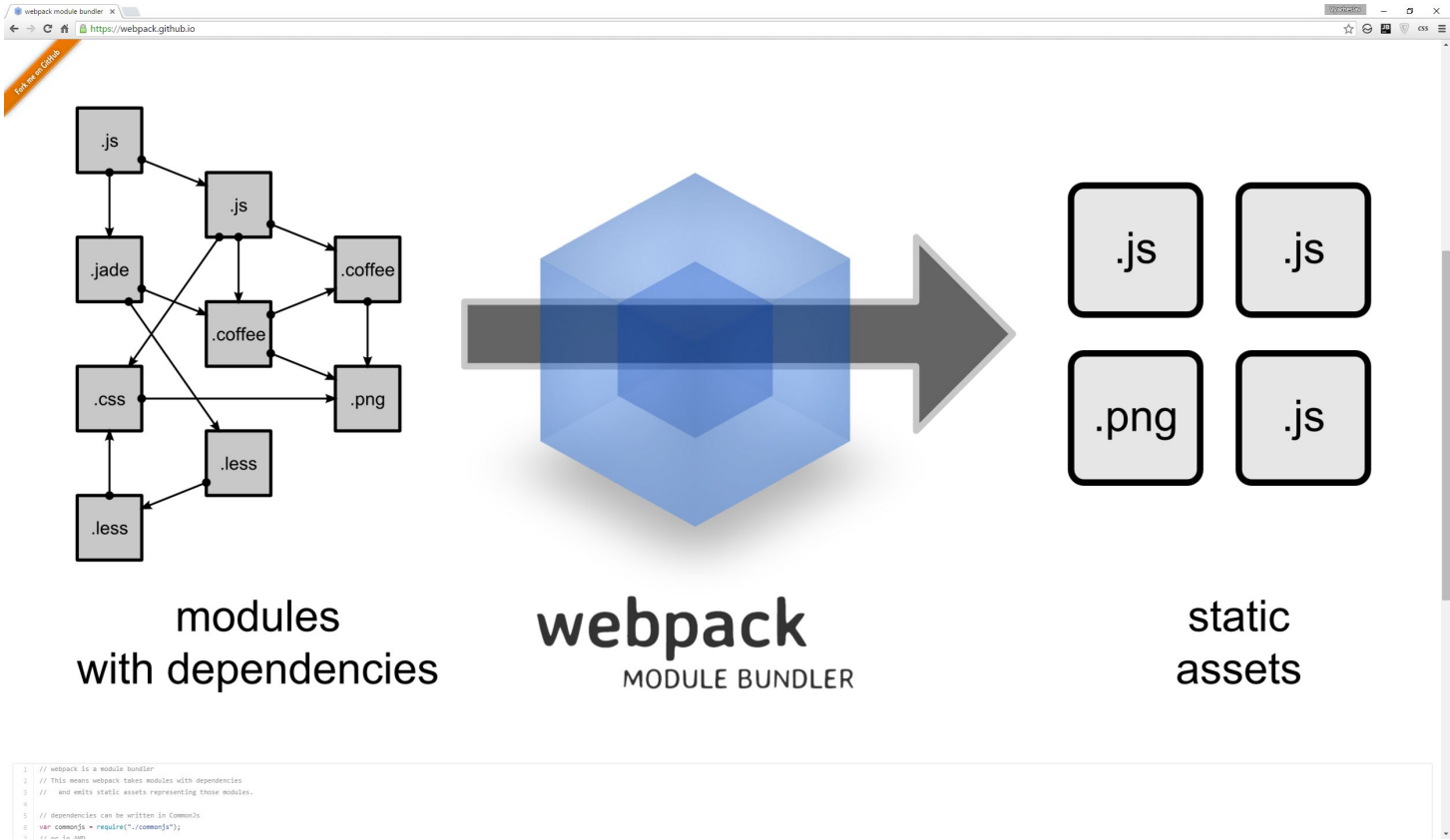


Practice Task: Sample of Browserify Usage

```
// create main.js  
var unique = require('uniq');  
var data = [1, 2, 2, 3, 4, 5, 5, 5, 6];  
console.log(unique(data));  
// install uniq module  
npm install uniq  
// bundle modules into one file  
browserify main.js -o bundle.js  
// link one file to the html  
<script src="bundle.js"></script>
```

webpack

<https://webpack.github.io/>



How is webpack Different?

- Existing module bundlers are not well suited for big projects (big single page applications). The most pressing reason for developing another module bundler was Code Splitting and that static assets should fit seamlessly together through modularization.
- **Code Splitting**: webpack has two types of dependencies in its dependency tree: sync and async. Async dependencies act as split points and form a new chunk. After the chunk tree is optimized, a file is emitted for each chunk.
- **Loaders**: webpack can only process JavaScript natively, but loaders are used to transform other resources into JavaScript. By doing so, every resource forms a module.
- **Clever parsing**: webpack has a clever parser that can process nearly every 3rd party library. It even allows expressions in dependencies like `sorequire("./templates/" + name + ".jade")`. It handles the most common module styles: CommonJs and AMD.
- **Plugin system**: webpack features a rich plugin system. Most internal features are based on this plugin system. This allows you to customize webpack for your needs and distribute common plugins as open source.

details: <http://webpack.github.io/docs/what-is-webpack.html>

Practice task

Complete tutorial from webpack official website:

<http://webpack.github.io/docs/tutorials/getting-started/>

Thank you!

USA HQ

Toll Free: 866-687-3588

Tel: +1-512-516-8880

Ukraine HQ

Tel: +380-32-240-9090

Bulgaria

Tel: +359-2-902-3760