softserve experience matters

# 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

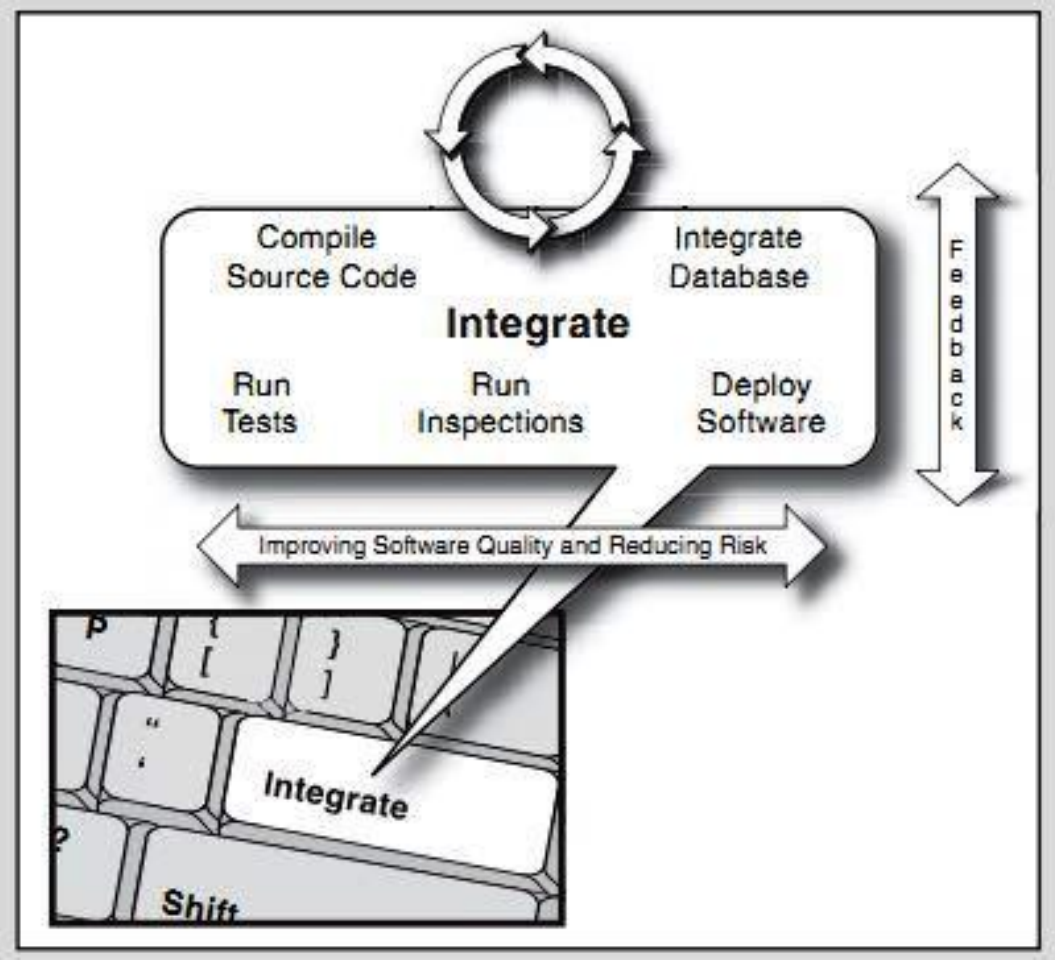
# 1. JS Tools and CI Overview

# *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*

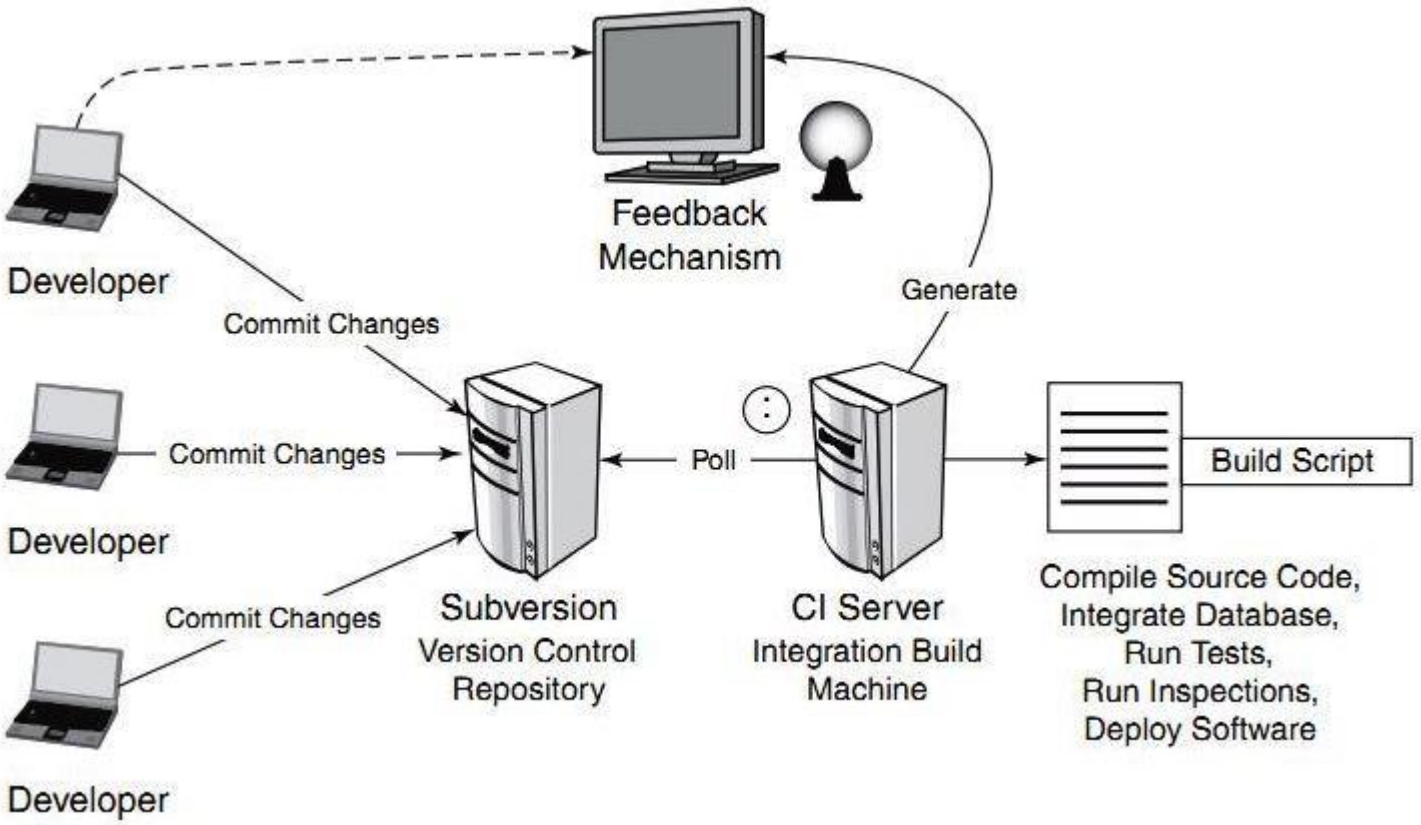
**Martin Fowler**

# The Integrate Button

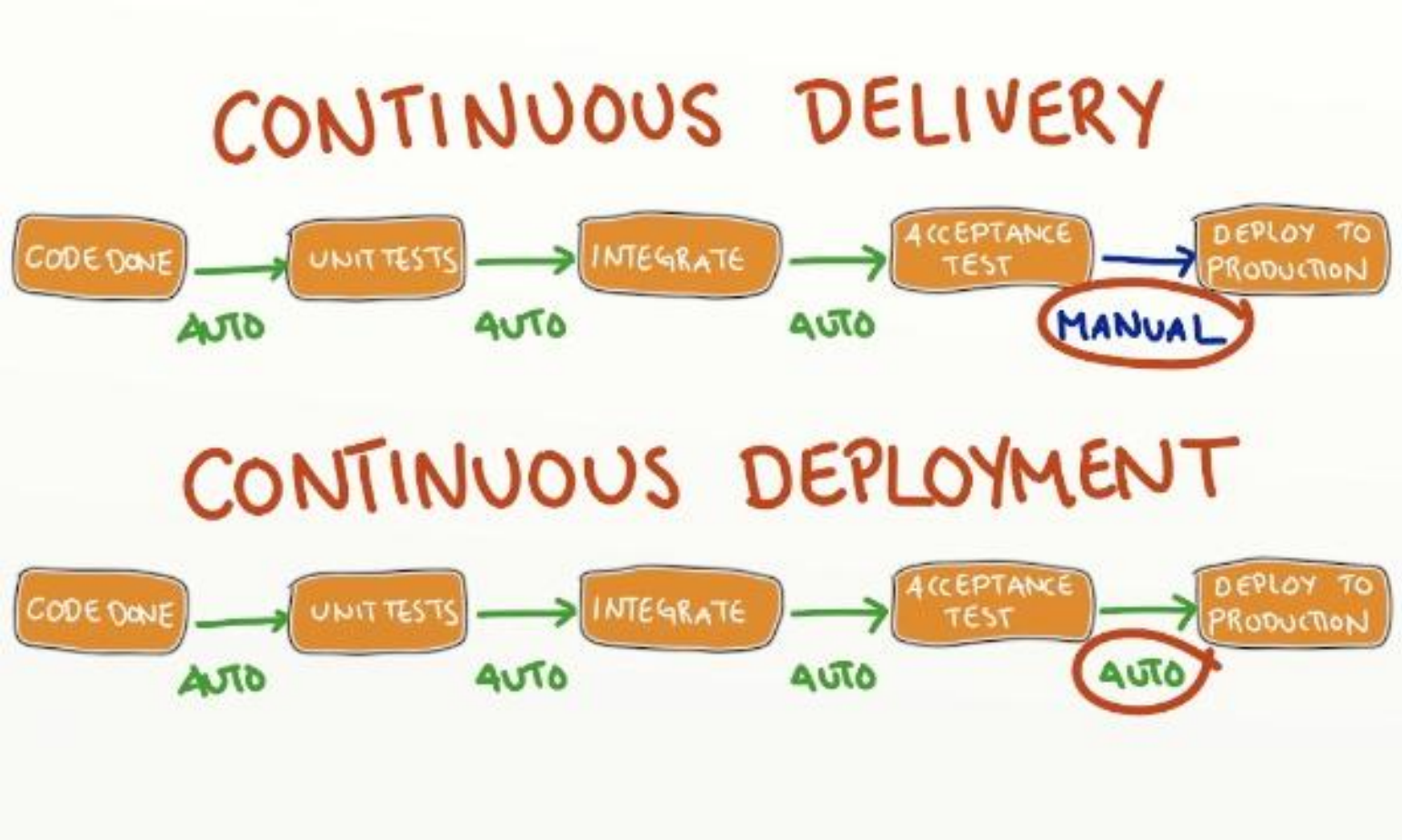



CI is a process that consists of **continuously** compiling, testing, inspecting and deploying source code

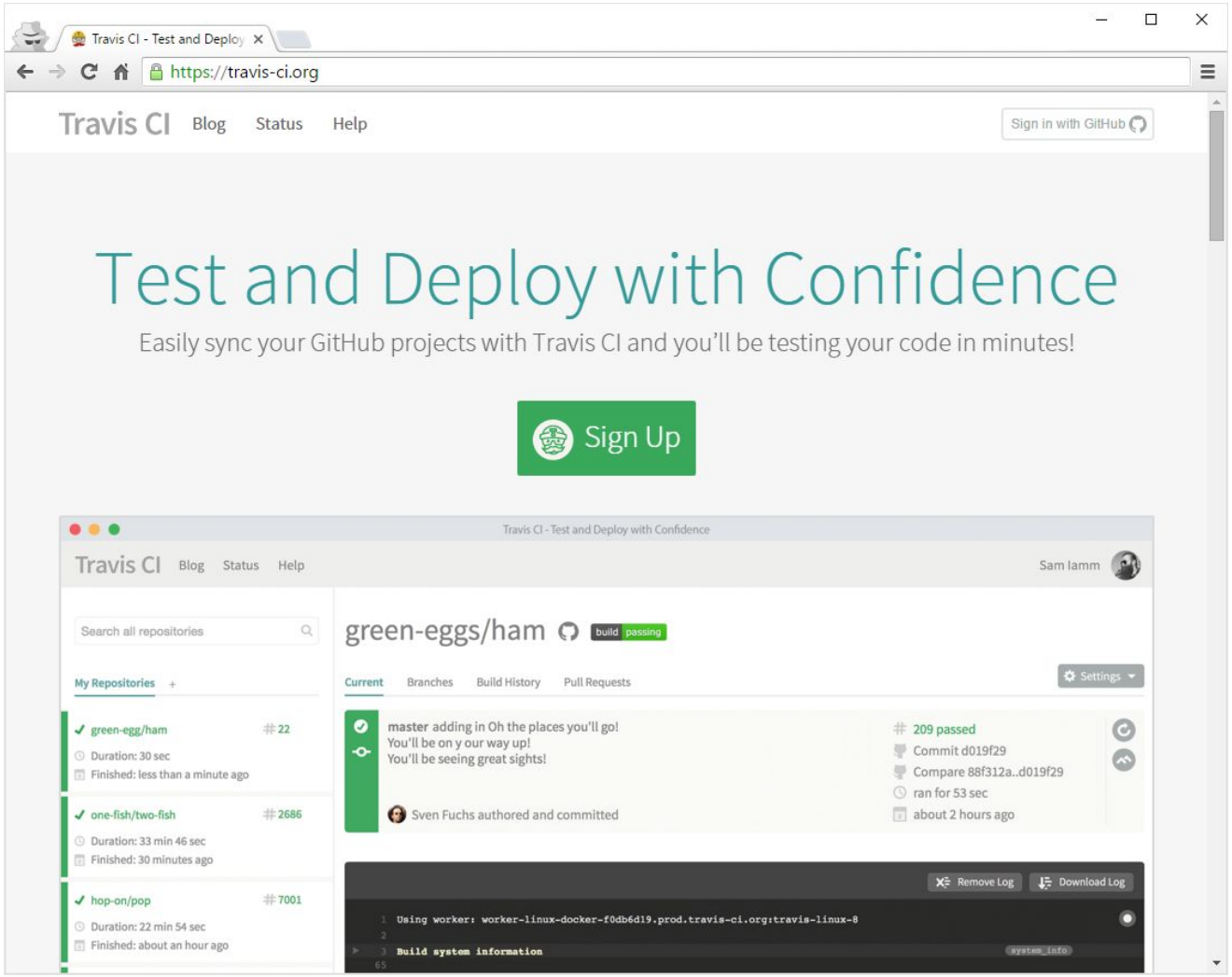
# CI Workflow


Feedback
Mechanism
Developer
Commit Changes
Generate
Commit Changes
Developer
Poll
Build Script
Commit Changes
Subversion
Version Control
Repository
CI Server
Integration Build
Machine
Compile Source Code,
Integrate Database,
Run Tests,
Run Inspections,
Deploy Software
Developer

# Continuous Delivery & Continuous Deployment



# Travis CI




softserve experience matters

# 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.*

softserve experience matters

# 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


Vyacheslav
JSCS: JavaScript Code Styl
jscs.info
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

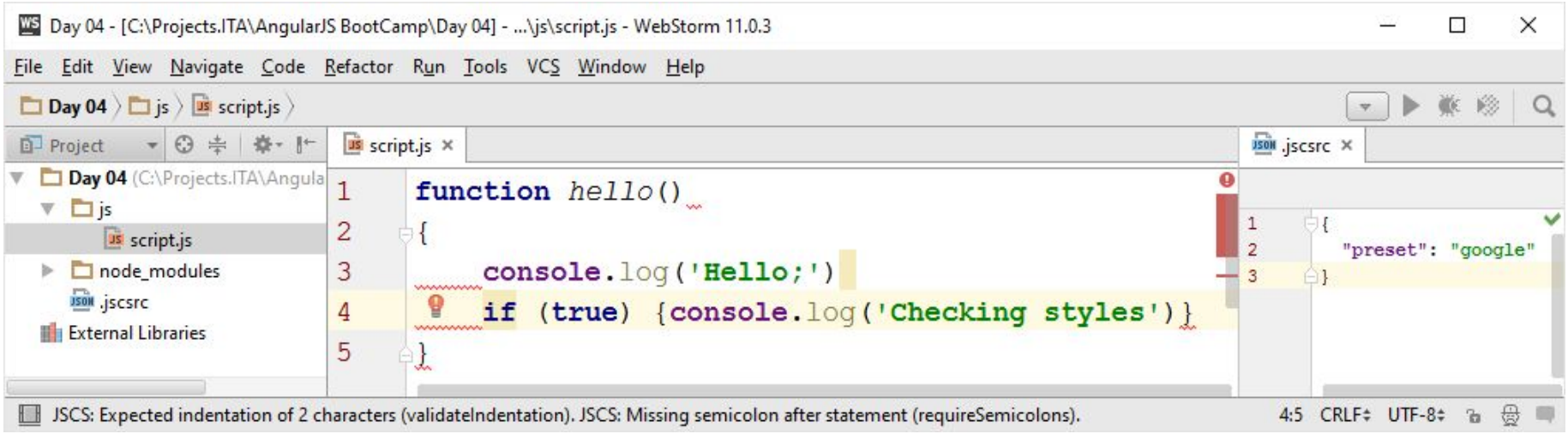
- Atom plugin: https://atom.io/packages/linter-jscs
- Brackets Extension: https://github.com/globexdesigns/brackets-jscs
- Grunt task: https://github.com/jscs-dev/grunt-jscs/
- Gulp task: https://github.com/jscs-dev/gulp-jscs/
- Overcommit Git pre-commit hook manager: https://github.com/brigade/overcommit/
- 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
}
```

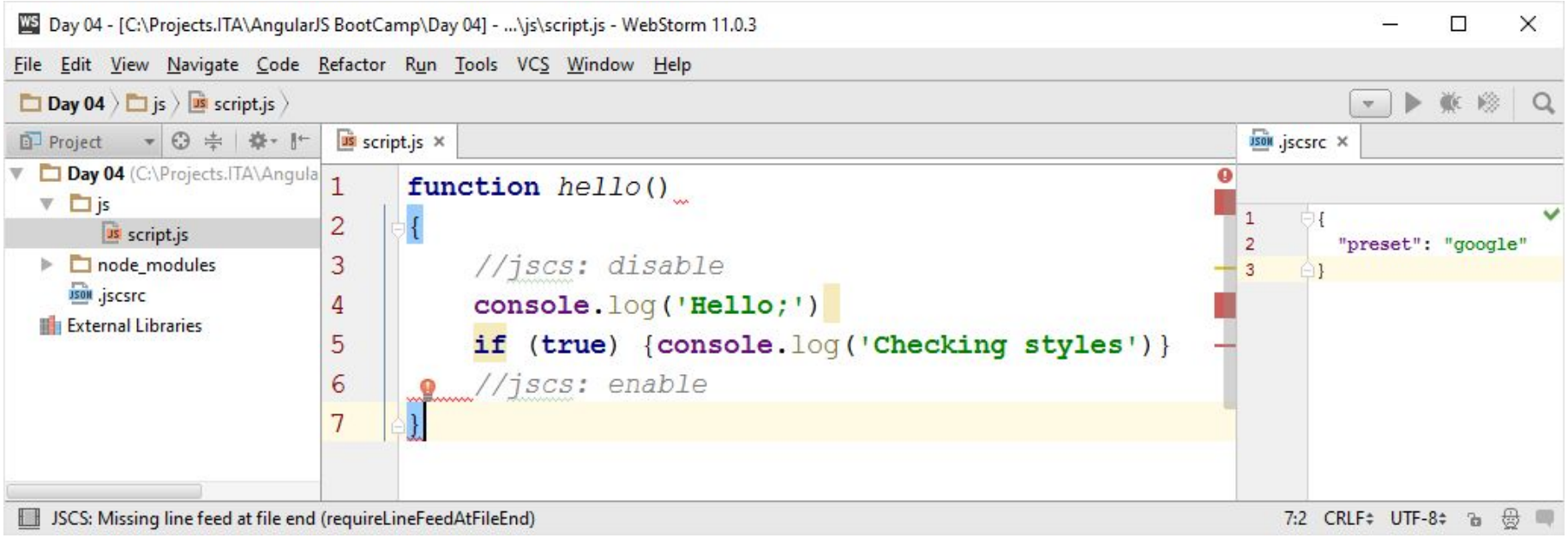
# WebStorm Sample




softserve experience matters

# WebStorm Sample

- Disabling/Enabling in Code




softserve experience matters

# A Comparison of JavaScript Linting Tools

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




softserve experience matters

# 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

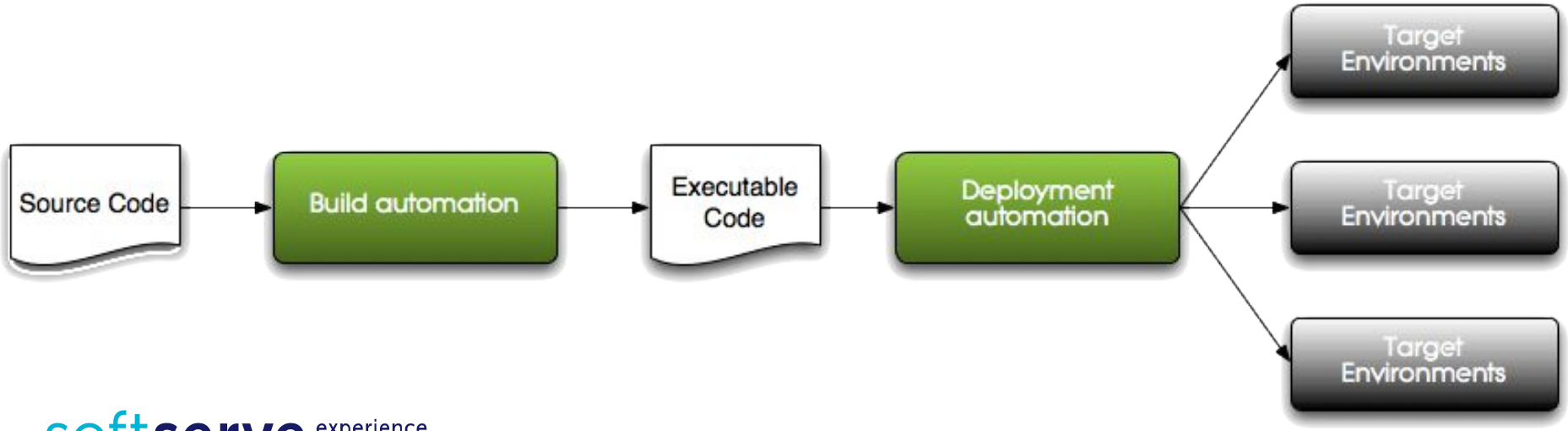
# GRUNT JS & Automation

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

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");
});
```

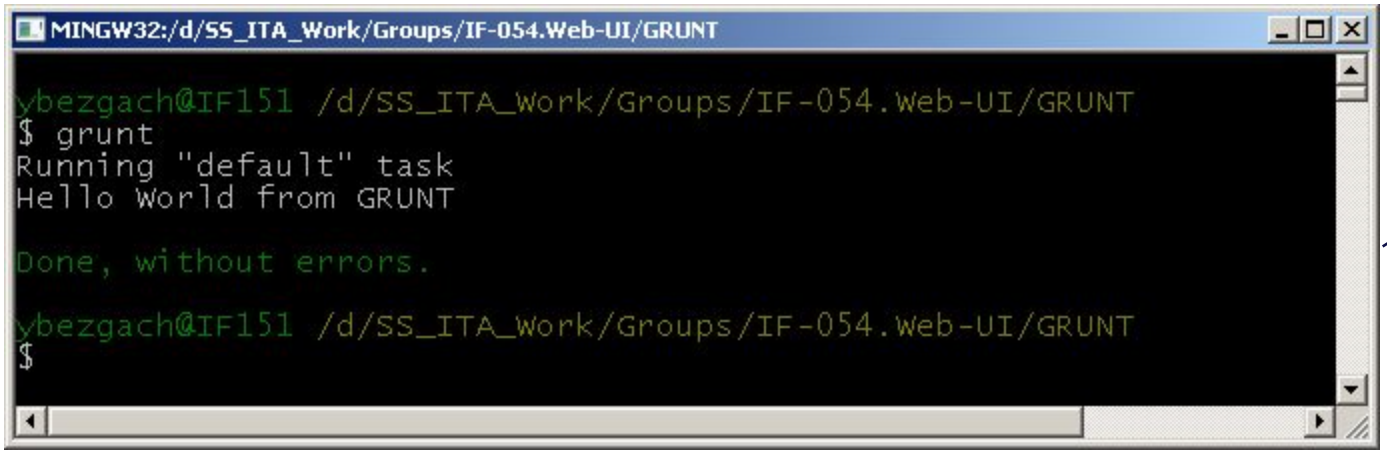
Save file ***gruntfile.js*** and run **grunt**

# 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




Output [Result]

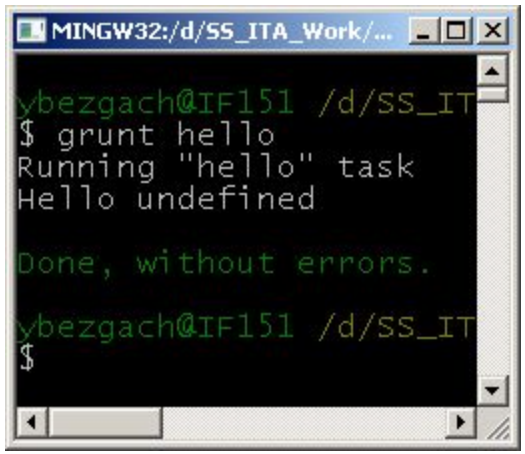
When we run **grunt** without parameters it will find the ***default*** task definition and run it

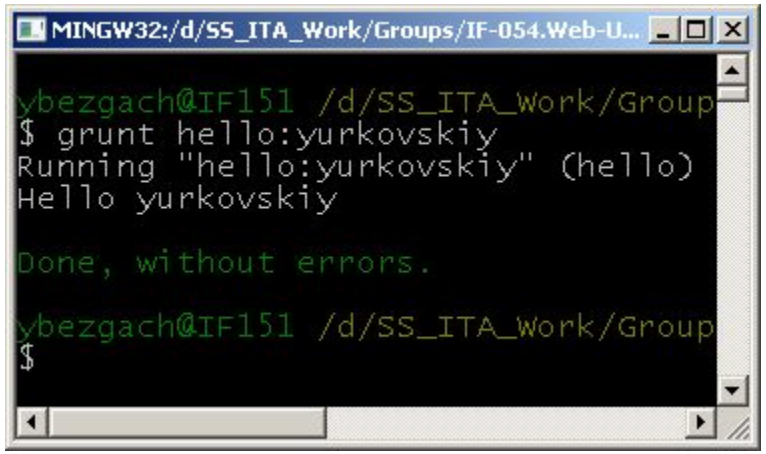
softserve experience matters

# Tasks with parameters

The ‘hello’ task is defined as:

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



# Check parameters

The ‘div’ task is defined as

gruntfile.js

```
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 |
| --- | --- |
| `grunt.file.mkdir` | grunt.file.defaultEncoding |
| `grunt.file.delete` | grunt.file.preserveBOM |
| `grunt.file.copy` | |
| `grunt.file.read` | |
| `grunt.file.readJSON` | |
| `grunt.file.write` | |

**more info on*
*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

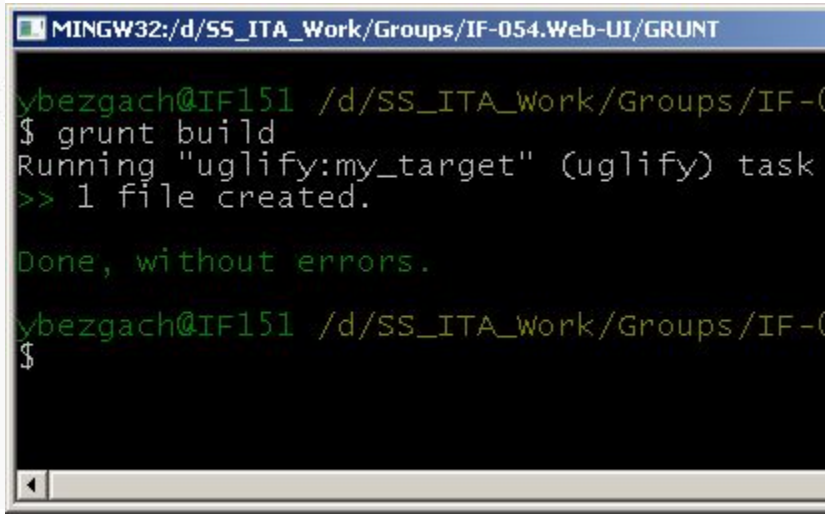
```
module.exports = function(grunt) {
    grunt.initConfig({
        uglify: {
            options: {
                mangle: true
            },
            my_target: {
                files: {
                    'dest/output.min.js': ['src/input.js']
                }
            }
        }
    });

    grunt.loadNpmTasks("grunt-contrib-uglify");


    grunt.registerTask("build", [
        "uglify",
        ]
    );
}
```

input.js

```
function add(number1, number2) {
    return (number1 + number2);
}
```




output.min.js

```
function add(a,b){return a+b}
```

softserve experience matters

# 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

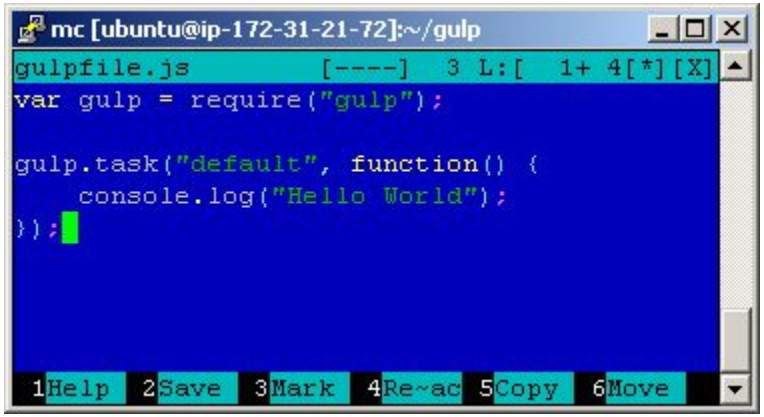
- `npm install -g gulp`
- In the project directory (root level):
  - create file **`package.json`** or use **`npm init`**
  - Install Gulp as dev dependency
    **`npm install gulp --save-dev`**
  - Create file **`gulpfile.js`**

Easy To Install


package.json

# Define a task

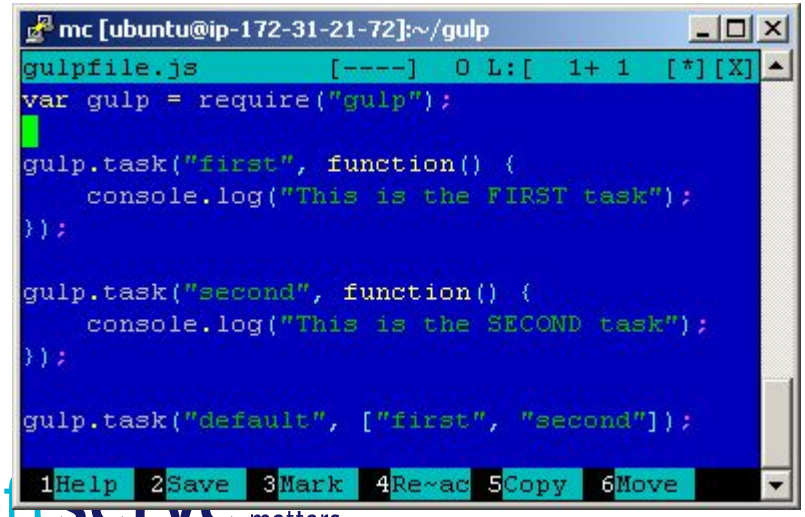
```
var gulp = require("gulp");
gulp.task("default", function() {
    // code for task
});
```

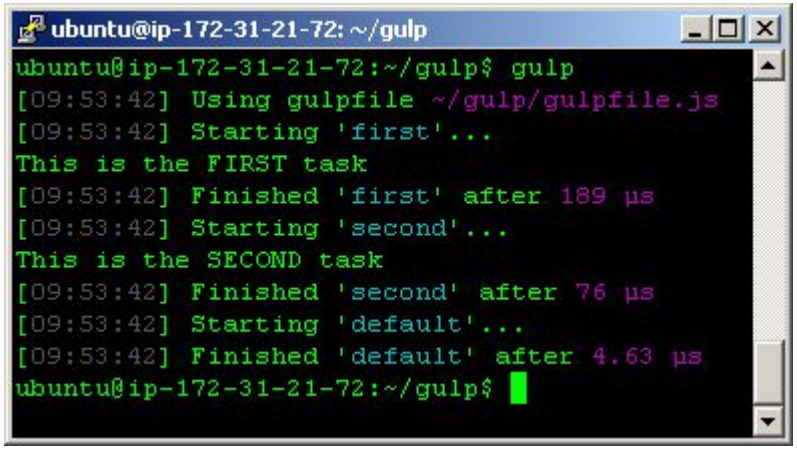



# Task series, dependency

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.



# 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

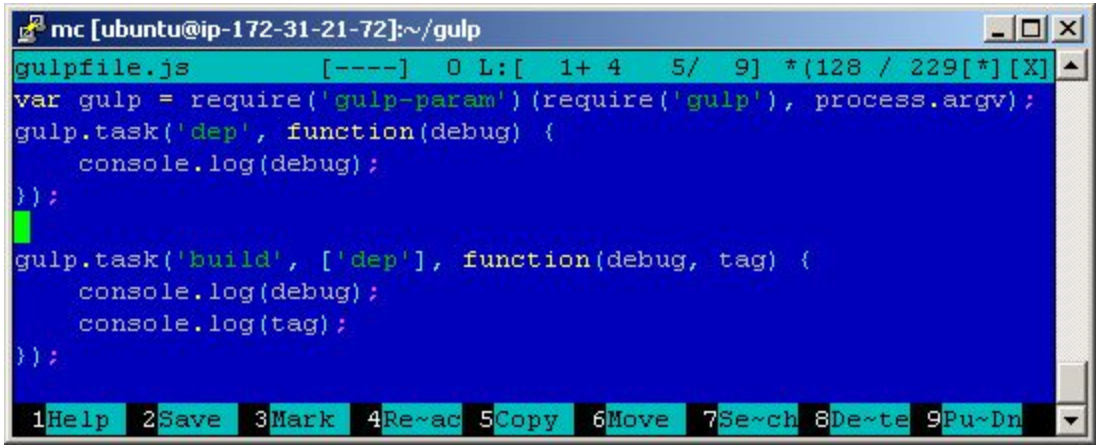
```
1  var gulp = require("gulp");
2  var uglify = require("gulp-uglify");
3  var concat = require("gulp-concat");
4
5  gulp.task("default", function() {
6      return gulp.src("src/*.js")
7      .pipe(uglify())
8      .pipe(concat("funcs.js"))
9      .pipe(gulp.dest("dist"));
10 });
```

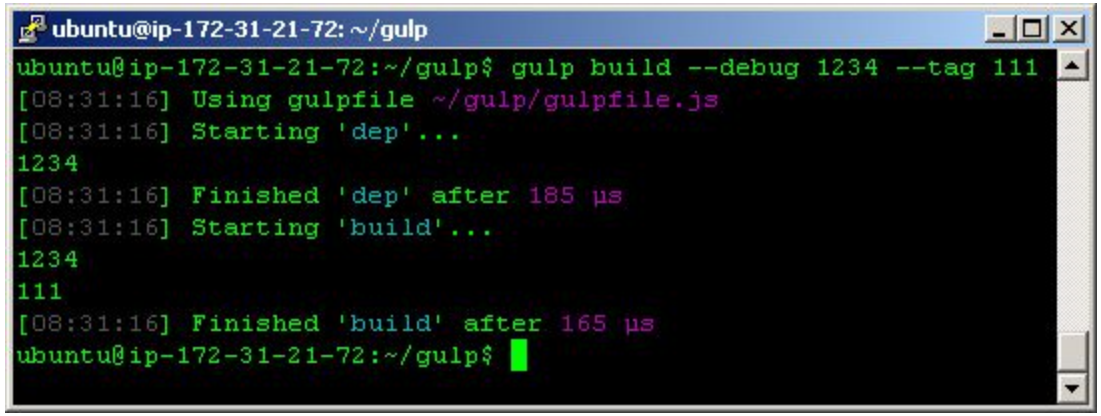
# Command line arguments

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

Plugins will help ☺

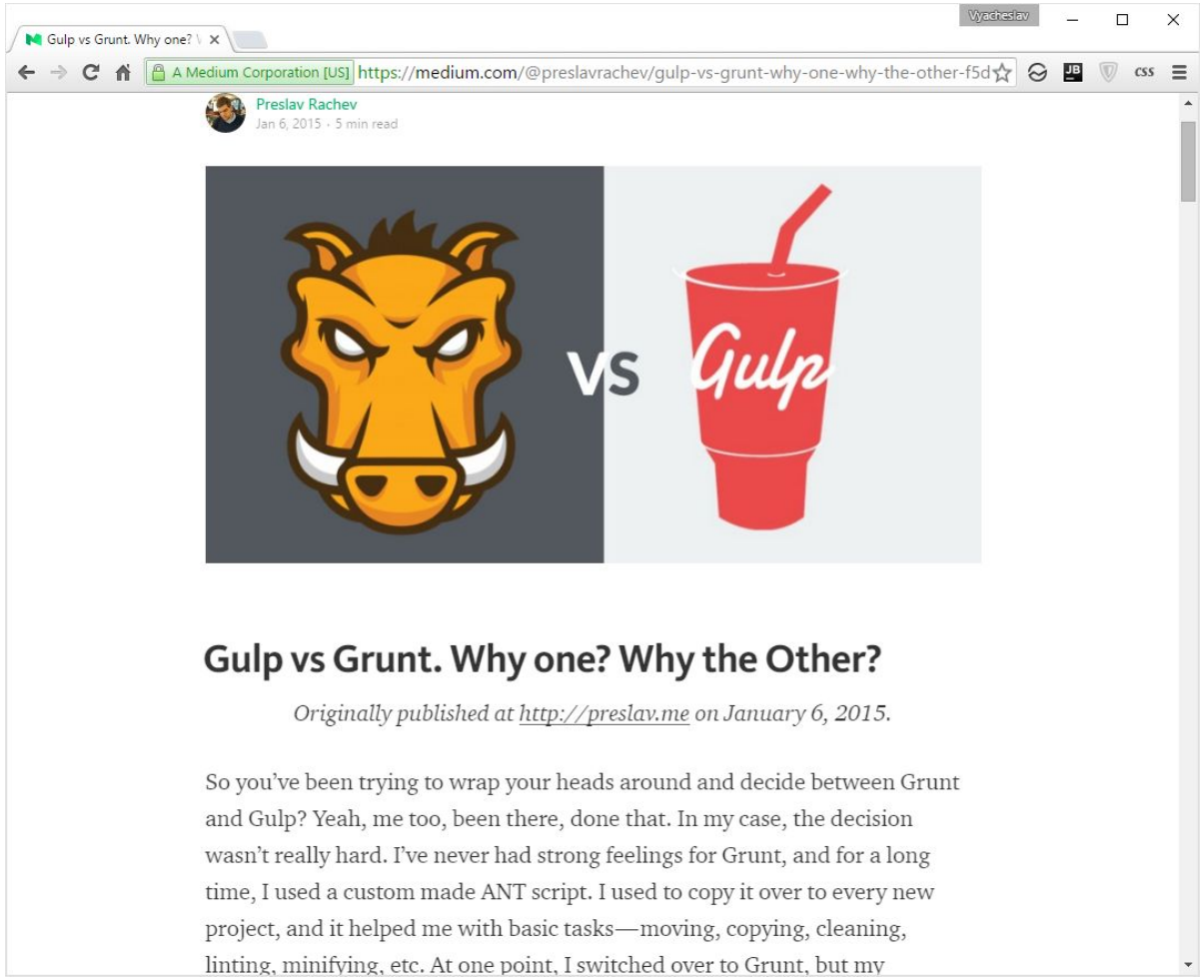
- yargs
- gulp-param




softserve experience matters

# Gulp vs Grunt

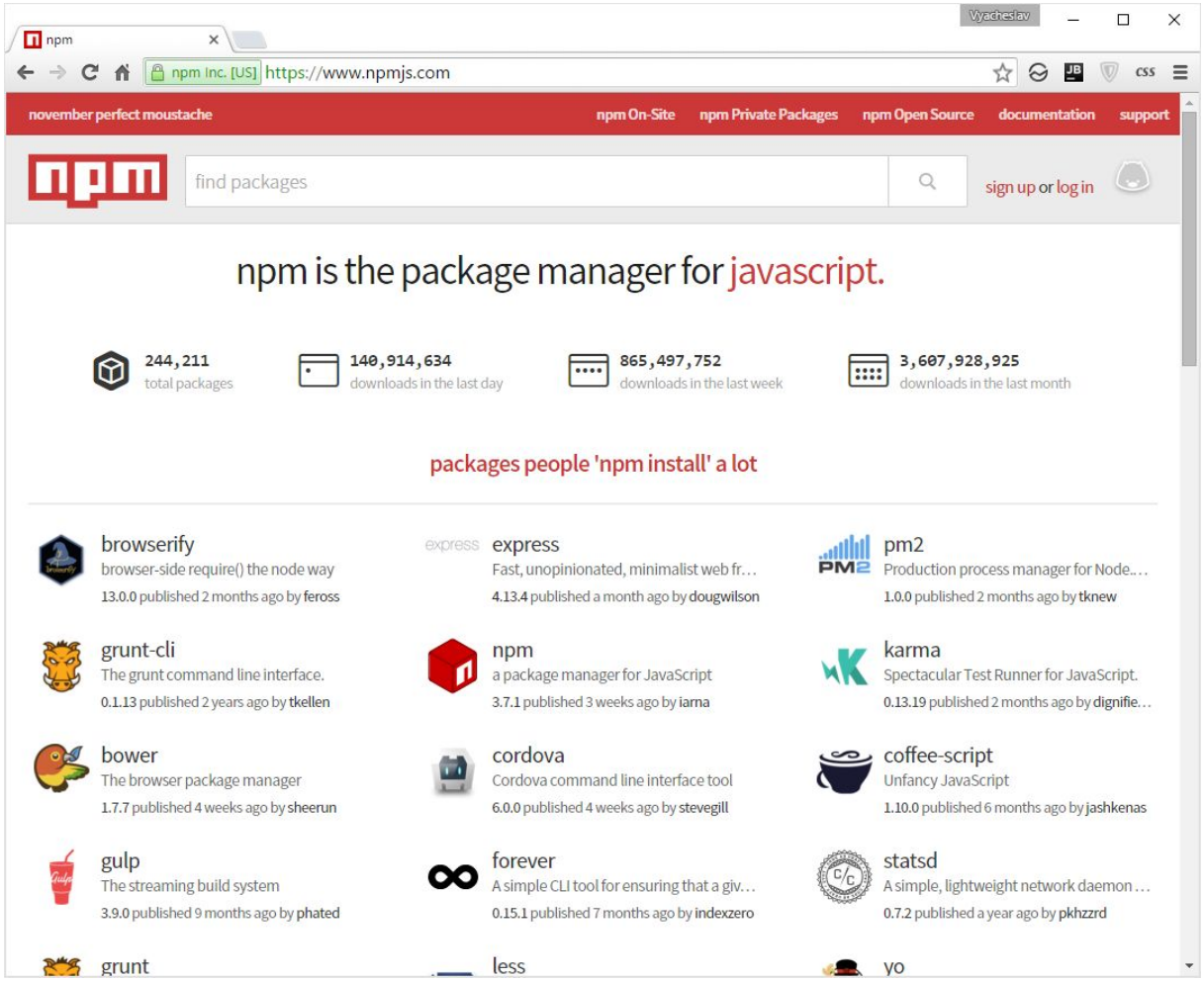



https://medium.com/@preslavrachev/gulp-vs-grunt-why-one-why-the-other-f5d3b398edc4#.jez2mtxgl

softserve experience matters

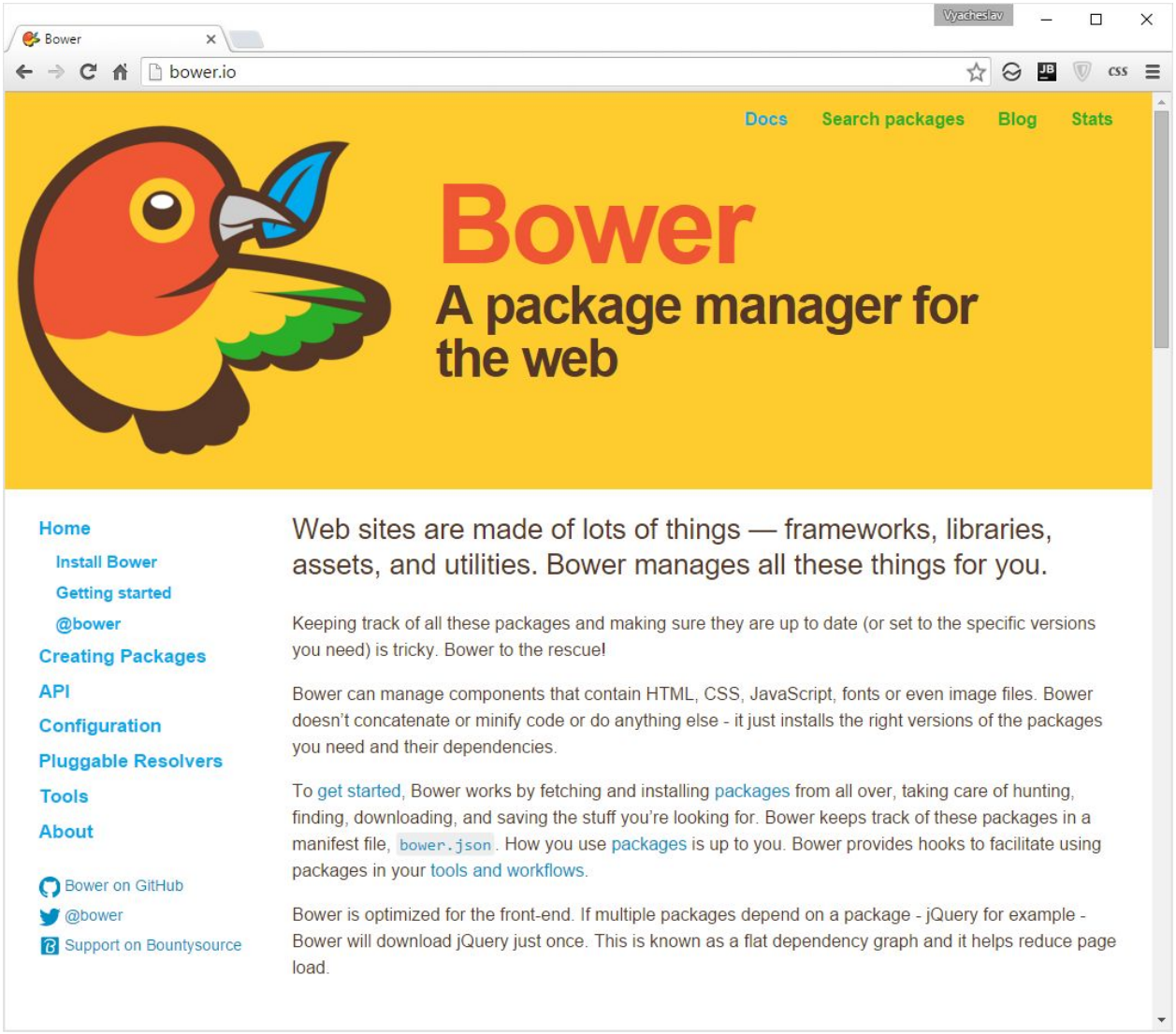
# 4. npm/bower

# Intro to npm




softserve experience matters

# bower


Bower
bower.io
Vyacheslav
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 paramters

  `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

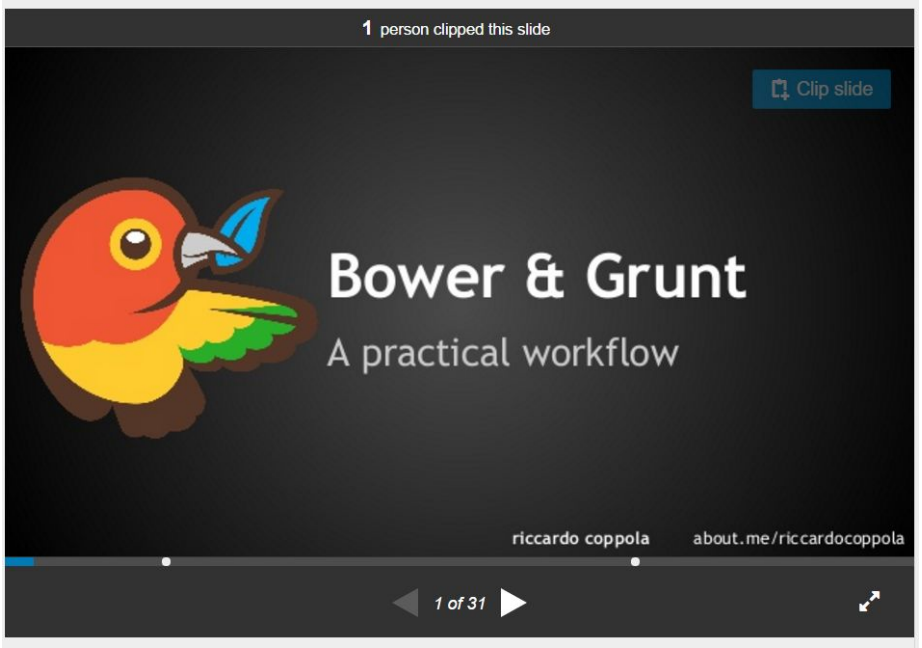
# npm vs bower

| | 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:

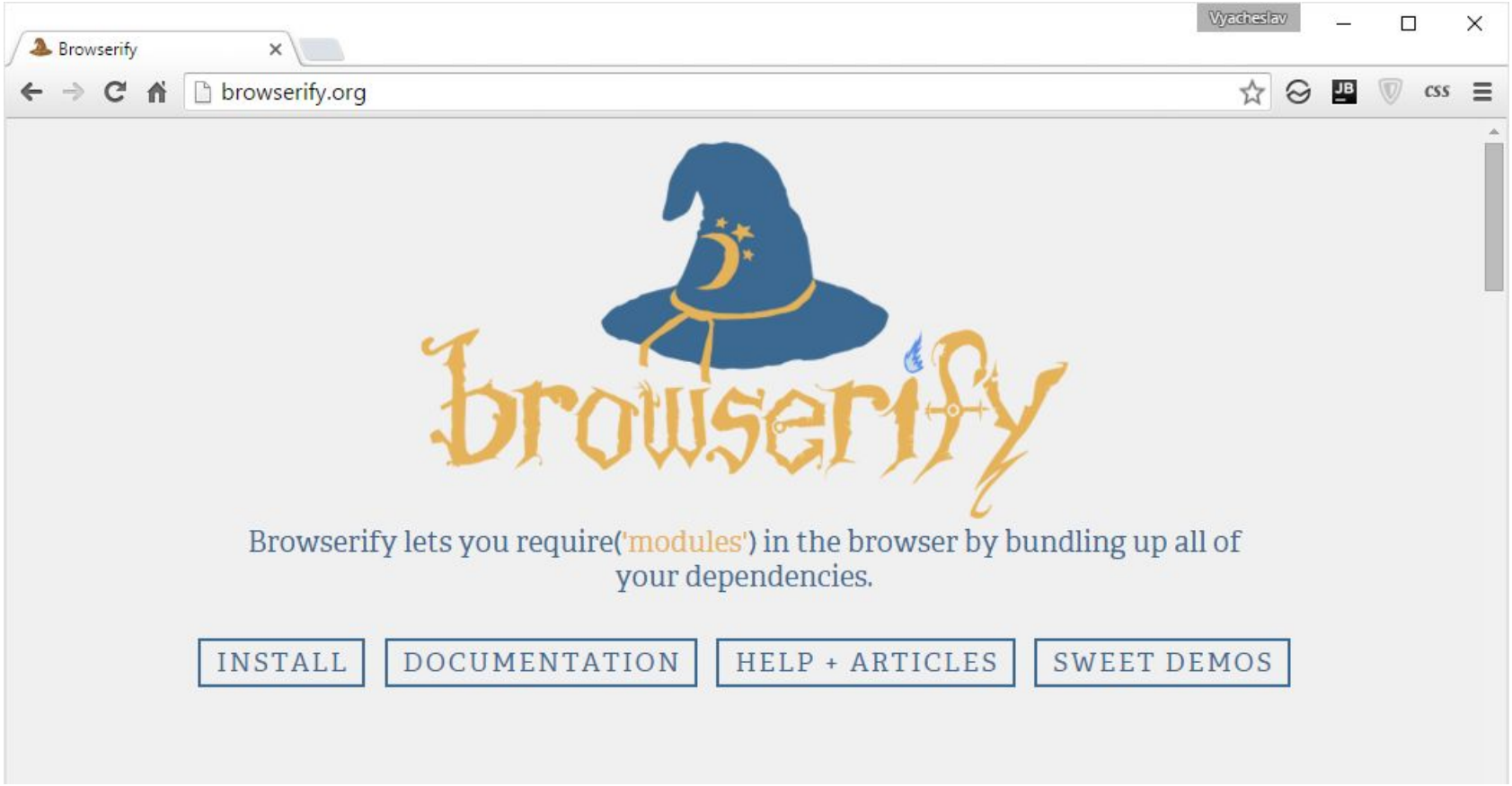
1. Different solutions (jQuery, Underscore, Knockout, Angular JS…)
2. Multiple versions (different versions of jQuery, Bootstrap…)
3. Pre-processing formats (less/sass/stylus, handlebars/jade/ejs, CoffeeScript/TypeScript/ES2015…)

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/

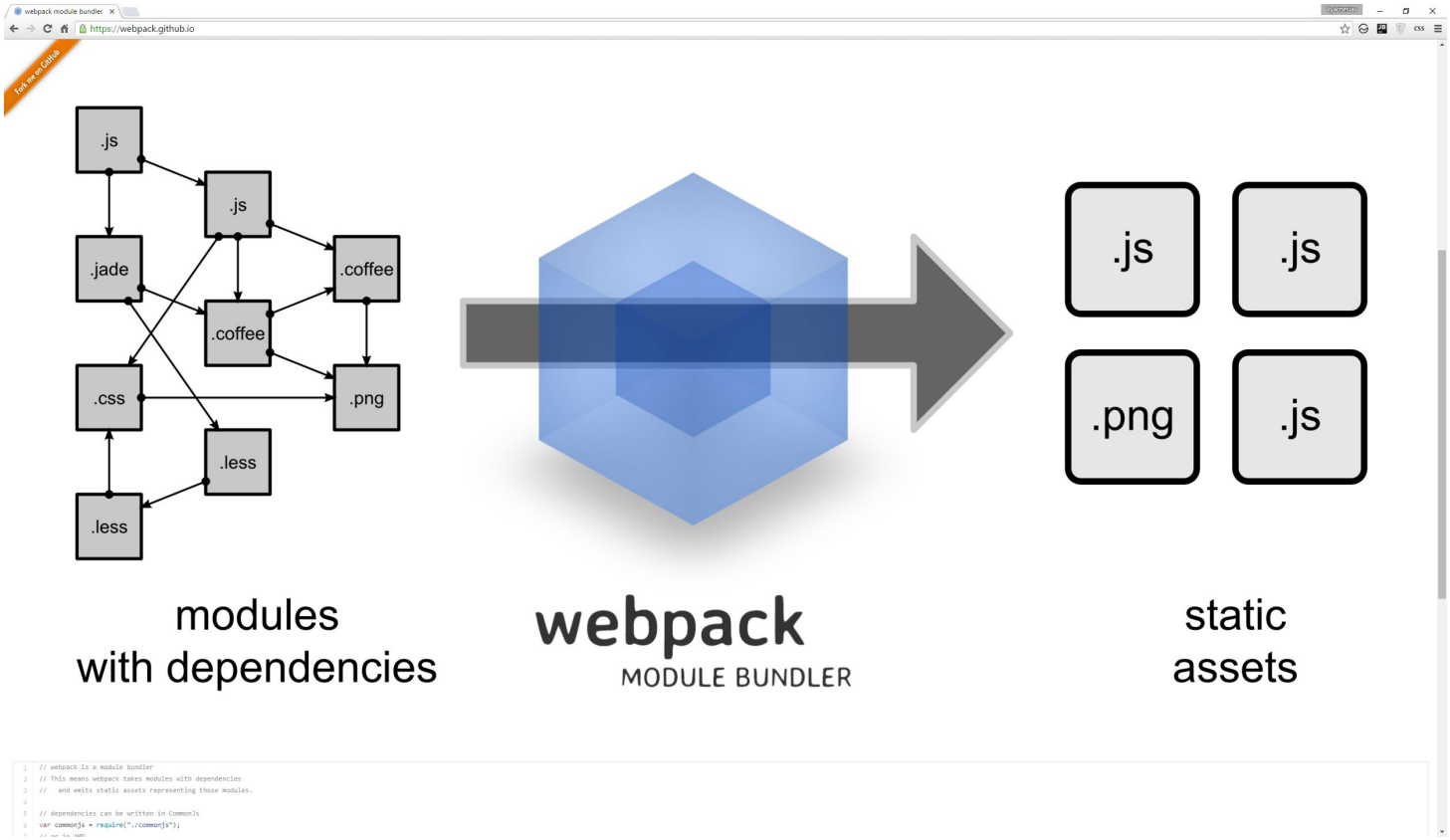


# Practice Task: Sample of Browserify Usage

```
// 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/




softserve experience matters

# 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/

softserve experience matters

# 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