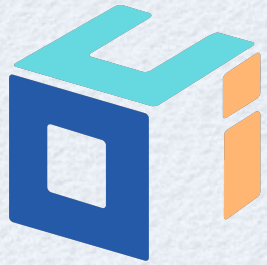
Modern JavaScript Tools

slides at https://github.com/mvolkmann/talks

R. Mark Volkmann
Object Computing, Inc.
http://objectcomputing.com
Email: mark@objectcomputing.com
Twitter: @mark_volkmann
GitHub: mvolkmann



OCI | TRAINING

Table of Contents

The Plan

For each topic we will cover ...

1) purpose

2) alternatives

3) how to configure and use

4) demo

Demo App

- "Hello World" web app
  - very simple to avoid distracting from focus on tools
  - doesn't use any framework to avoid bias toward one
  - available at https://github.com/mvolkmann/modern-js-tools
- Notable files
  - **`index.html`**
  - **`demo.css`**
  - **`src/index.js`** - uses ES6 imports
  - **`src/demo.js`** - uses ES6 exports
  - **`src/demo.test.js`** - tests for **`demo.js`**
  - **`package.json`**
  - plus several tool configuration files discussed later

npm Overview

https://www.npmjs.com/

- Purpose
  - installs Node packages
  - manages dependencies in `package.json` and `package-lock.json`
  - scripts common tasks
- Alternatives
  - for installing and managing dependencies - **yarn**
  - for scripting tasks - **gulp** and **Grunt**

npm Details

- Automatically installed when Node.js is installed
  - can install separately, but why?
- Initially an acronym for Node Package Manager
  - they are trying hard to convince us that it's not an acronym now ... not sure why
- Common commands
  - **`npm init`** - asks questions and creates **`package.json`**
  - **`npm install`** ***`name`*** - installs a specified package as a runtime dependency
  - **`npm install -D`** ***`name`*** - installs a specified package as a development dependency
  - **`npm install`** - installs all dependencies listed in **`package.json`**
    - and creates **`package-lock.json`** file
  - **`npm run`** ***`script-name`*** runs an npm script can omit **`run`** for "special" scripts
  - **`npx`** ***`command`*** - see https://www.npmjs.com/package/npx

npm Scripts

- Defined in **`package.json`**
- Can write in a way that works on Windows and *nix platforms
  - **shx** - "Portable Shell Commands for Node"
    - https://github.com/shelljs/shx
  - **cross-env** - "Run scripts that set and use environment variables across platforms"
    - https://github.com/kentcdodds/cross-env
- Examples

also see **shelljs** at https://shelljs.org

also see **cross-run** at https://github.com/sheerun/cross-run

```
"build": "npm-run-all verify bundle",
"bundle": "webpack",
"clean": "rm -rf build coverage",
"cover": "open coverage/lcov-report/index.html",
"flow": "flow",
"format": "prettier-eslint --write 'src/**/*.js'",
"lint": "eslint --quiet src --ext .js",
"parcel": "parcel index.html",
"prepush": "npm run verify",
"sync": "browser-sync start --server --files 'index.html' 'build/bundle.js'",
"test": "jest --watch src",
"verify": "npm-run-all lint flow test"
```

`npm install -D npm-run-all`

!Windows; can use shx

!Windows, consider https://www.npmjs.com/package/opener

git hook processed by Husky

ESLint Overview

http://eslint.org/

- Purpose
  - "The pluggable linting utility for JavaScript and JSX"
  - can report many syntax errors and potential run-time errors
  - can report deviations from specified coding guidelines
- Alternatives
  - **JSLint** - from Douglas Crockford
  - **JSHint** - a more configurable, less opinionated version of JSLint
  - **TSLint** - for TypeScript

ESLint Details

- Error messages identify violated rules, making it easy to adjust them if you disagree
- Has `--fix` mode that can fix violations of many rules
  - modifies source files
- To install, `npm install -D eslint babel-eslint`
- To use from an npm script, add following to `package.json`

```
"lint": "eslint --quiet src --ext .js",
```

`--quiet` only reports errors

"You only need to use **babel-eslint** if you are using **types** (Flow) or **experimental features** not supported in ESLint itself yet."

- Editor/IDE integrations available
  - Atom, Eclipse, emacs, Intellij IDEA, Sublime, VS Code, Vim, WebStorm

may also want `eslint-plugin-flowtype`, `eslint-plugin-html`, and `eslint-plugin-react`

`eslint-plugin-html` lints JS in HTML files

ESLint Rules

- No rules are enforced by default
- Desired rules must be configured
- See list of current rules at http://eslint.org/docs/rules/
- Configuration file formats supported
  - JSON - **`.eslintrc.json`**; can include JavaScript comments; most popular
  - JavaScript - **`.eslintrc.js`**
  - YAML - **`.eslintrc.yaml`**

  see mine at https://github.com/mvolkmann/MyUnixEnv/blob/master/**.eslintrc.json**

  - inside **`package.json`** using **`eslintConfig`** property
  - use of **`.eslintrc`** containing JSON or YAML is **deprecated**
- Searches upward from current directory for these files
  - combines settings in all configuration files found with settings in closest taking precedence
  - configuration file in home directory is only used if no other configuration files are found

ESLint Demo

- See **`lint`** script in **`package.json`**
- Modify **`src/demo.js`**
  - remove an "**`e`**" from name of **`handleGreet`** function
- **`npm run lint`**

Prettier Overview

https://github.com/prettier/prettier

- Purpose
  - "An opinionated JavaScript formatter ...
    with advanced support for language features
    from ES2017, JSX, Flow, TypeScript, CSS, LESS, and SCSS"
  - "Parses your JavaScript into an AST and pretty-prints the AST,
    completely ignoring any of the original formatting"
    - "Well actually, **some original styling is preserved** when practical -
      see empty lines and multi-line objects."
  - can also format JSON, Markdown, and more
- Alternatives
  - **Standard** - https://standardjs.com/
    - no semicolons
    - space after function names before left paren in definitions
      ex. `function foo (bar, baz) {`

Prettier Details

- To install, **`npm install -D prettier`**
- To use from an npm script, add following to **`package.json`**

```
"format": "prettier --no-bracket-spacing --single-quote --write 'src/**/*.{css,js}'",
```

  - to format all matching files under **`src`** directory, enter **`npm run format`**
  - **`--write`** option overwrites existing files with formatted versions

Must have quotes around glob path!
(see https://prettier.io/docs/en/cli.html)

in Windows use escaped double-quotes

```
--write \"src/**/*.{css,js}\"",
```

- Can also configure in **`.prettierrc`** file

```
{
  "bracketSpacing": false,
  "singleQuote": true
}
```

- Doesn't run on files under **`node_modules`** by default
- Editor/IDE integrations available
  - Atom, Emacs, JetBrains, Sublime, Vim, VS Code

Create **`.prettierignore`** file and add **`package.json`** to it to avoid thrashing because npm install rewrites this file.

Prettier Options

- **`--jsx-bracket-same-line`**
  - puts closing > of JSX start tags on last line instead of on new line

```
<something
  prop1="value1"
  prop2="value1"
  prop3="value1"
  prop4="value1"
>
  content
</something>
```

VS.

```
<something
  prop1="value1"
  prop2="value1"
  prop3="value1"
  prop4="value1">
  content
</something>
```

- ★ **`--no-bracket-spacing`**
  - omits spaces between brackets in object literals

```
{ foo='1' bar=true }
```

VS.

```
{foo='1' bar=true}
```

- **`--no-semi`** - omits semicolons
- **`--parser`** - for use with TypeScript, set to **`"typescript"`**
- **`--print-width n`** - defaults to 80
- ★ **`--single-quote`**

This may become the default. See https://github.com/prettier/prettier/issues/4102

  - uses single quotes instead of double quotes for string delimiters
- **`--tab-width n`** - defaults to 2
- **`--trailing-comma`**
  - adds trailing commas wherever possible; defaults to none
- **`--use-tabs`** - uses tabs instead of spaces for indentation
- and more lesser used options

Prettier Demo

- See **`format`** script in **`package.json`**
- Modify **`src/demo.js`**
  - remove several semicolons
  - mess up lots of indentation
  - put **`onLoad`** parameter on a separate line
- **`npm run format`** and reload file in editor to see changes
  or trigger from editor/IDE plugin

Babel Overview

- Purpose
  - transpiles JavaScript code to different JavaScript code
  - can use newer JS features in environments that don't support them yet
    - reads modern JS code and generates new JS code that runs in older environments
    - ex. **ES modules**
  - can use JS features not yet finalized by ECMAScript (via plugins)
    - ex. `String` `trimStart` and `trimEnd` methods stage 3 proposal
  - can use features that may never be part of ECMAScript
    - ex. **Flow** for type checking
- Alternatives
  - **TypeScript** - also adds types and support for custom syntax

Babel Details

- To install, **`npm install -D babel-cli babel-preset-env`** described on next slide
- To use from an npm script, add following to **`package.json`**

```
"babel": "babel src -d dist"
```

  - not needed if using **`webpack`** and **`babel-loader`**

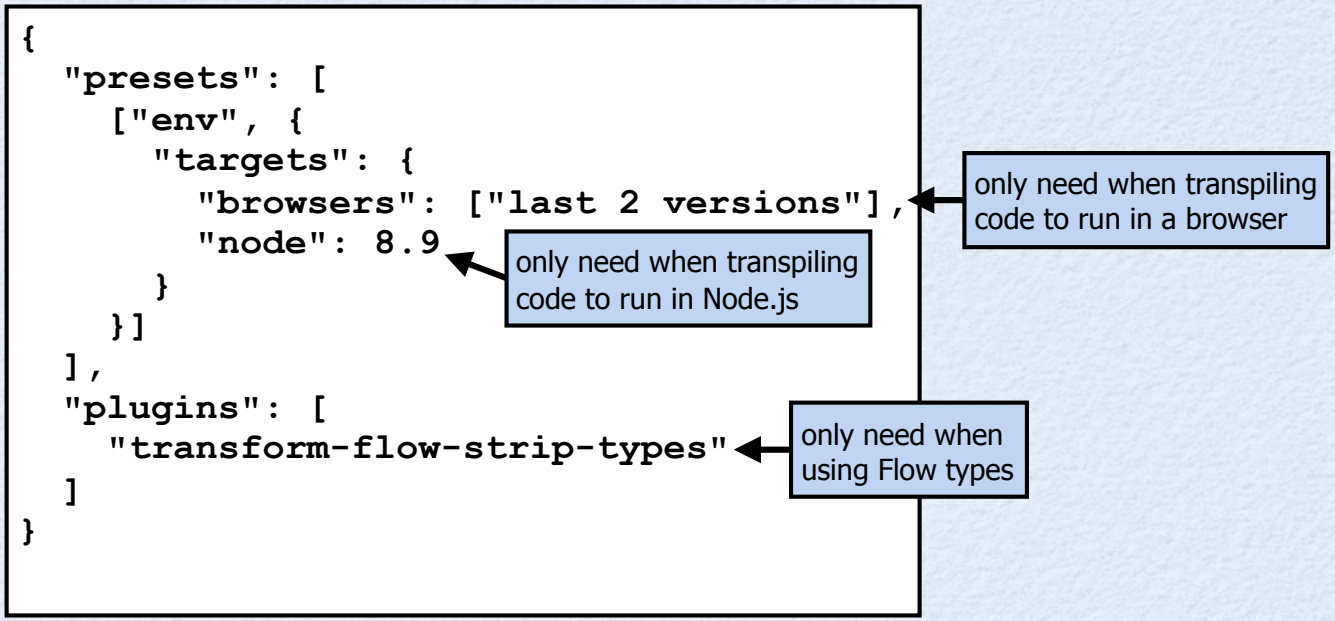
Babel Plugins

- Recommended plugins
  - **babel-preset-env**
    - "automatically determines the Babel plugins you need based on your supported environments"
    - can target specific browser versions and Node.js versions
    - https://babeljs.io/docs/plugins/preset-env/
  - **babel-plugin-transform-flow-strip-types**
    - removes Flow type declarations from `.js` files
    - https://babeljs.io/docs/plugins/transform-flow-strip-types/
- To use a plugin
  - install with npm as a dev dependency
  - configure in `.babelrc` (see next slide)

environments are specified in `.babelrc`

Babel Configuration

- In **`.babelrc`** file
- Example



Babel Demo

- See **`babel`** script in **`package.json`**
  - only there for this demo since webpack (covered later) will also run babel
- **`rm -rf dist`**
- **`npm run babel`**
- Note files in **`dist`** directory

Why Use Types?

- Can find type errors before runtime
  - more convenient than waiting until runtime
- Types document expectations about code
  - types of variables, object properties, function parameters, and function return types
  - comments can be used instead, but those
    - are more verbose
    - tend to be applied inconsistently
    - easily go out of date when code is updated
- Increases refactoring confidence
  - don't have to wonder what assumptions callers made about supported types
- Removes need to write ...
  - error checking code for type violations
  - type-related unit tests
- Editor/IDE plugins can use types to highlight issues and provide code completion

Why Avoid Types?

- Takes time to ...
  - learn type syntax
  - master applying them
- Makes code more verbose
- Can hamper prototyping and rapid development
  - developers can lose focus when distracted by having to satisfy a compiler or type checker

When to Use Types

- Use types when
  - application is large, complex, or critical
  - expected lifetime of code is long and refactoring is likely
  - code will be written and maintained by a team of developers
- Avoid types when
  - the conditions above are not present

Flow Overview

https://flow.org/

- Purpose
  - "A static type checker, designed to find type errors in JavaScript programs"
  - catches many errors without types
    using **type inference** and **flow analysis**
  - "precisely tracks the types of variables as they flow through the program"
  - can gradually add types
- Alternatives
  - **TypeScript** - https://www.typescriptlang.org/

Flow Details

- Open source tool from Facebook
- Most ES6+ features are supported
  - for a list, see https://github.com/facebook/flow/issues/560
- Supports React and JSX
- To install, **`npm install -D`** ***`name`***
  - where ***`name`*** is **`babel-cli`**, **`babel-eslint`**, **`babel-plugin-transform-flow-strip-types`**, **`eslint-plugin-flowtype`**, and **`flow-bin`**
- Editor/IDE integrations available
  - Atom, emacs, Sublime, VS Code, Vim, WebStorm
- Too much to say about this
  - see slides at https://github.com/mvolkmann/talks/blob/master/flow.key.pdf and talk video at https://www.youtube.com/watch?v=5kt3urZOg4g

To install type definitions for all dependencies in **`package.json`**,
`npm install -g flow-typed`
`flow-typed install`
`flow init` (creates **`.flowconfig`**)
edit **`.flowconfig`** and
add **`flow-typed`** after **`[libs]`**

Reasons to Prefer Flow Over TS

- Catches more errors without adding types
  - via better flow analysis
- Strict null checking is the default
  - also true for new TS projects that use "`tsc --init`"
- Uses nominal rather than structural type checking for classes
  - the right thing to do
- Just does type checking, not transpiling,
  so Babel can be used for transpiling
  - can tell TS to target ES6 and then run that output through Babel,
    but that feels awkward
- Just adds types
  - TS extends the language with features that may never be added to JavaScript

Flow Demo

- See **`flow`** script in **`package.json`**
- Modify **`src/demo.js`**
  - change type of **`getGreeting`** **`name`** parameter to **`number`**
  - change type of **`handleGreet`** **`messageDiv`** parameter to **`HTMLInputElement`**
- **`npm run flow`**
  or see errors provided by editor/IDE plugin

Jest Overview

https://facebook.github.io/jest/

- Purpose
  - a JavaScript test framework "built on top of Jasmine"
  - "runs your tests with a fake DOM implementation (via jsdom) so that your tests can run on the command line"
  - can watch source and test files and have tests automatically rerun when they change
    - can run all tests or only those that failed in last run
- Alternatives
  - **Mocha** - https://mochajs.org/
  - **Jasmine** - https://jasmine.github.io/
  - **AVA** - https://github.com/avajs/ava
  - **Tape** - https://github.com/substack/tape

Jest Details

- To install, **`npm install -D jest`**
- To use from an npm script, add following to **`package.json`**

```
"test": "jest",
```

- Has good support for mocking and spies
  - create a mock function with `jest.fn()`
  - add a mock implementation with `jest.fn.mockImplementation(someFn)`
  - test that a mock function was called with
    `expect(mockFn).toBeCalled()` or
    `expect(mockFn).toBeCalledWith(arg1, arg2, ...)`
  - for more see https://facebook.github.io/jest/docs/en/mock-function-api.html
- Can use to test React components
  - but isn't specific to React
  - supports "snapshot tests" for React components (more on next slide)
  - default test framework of apps created with **create-react-app**

Jest Snapshot Tests

- Snapshot tests assert that ...
  - a component will render same content as last successful test
- The first time snapshot tests are run ...
  - `toMatchSnapshot` matchers save a representation of the rendered output in a subdirectory of the test file named `__snapshots__`
- In subsequent runs ...
  - the same representation is generated again and compared to what was saved in last successful run
- When snapshot tests fail ...
  - scroll back to review differences in rendered output
  - if changes are correct, press "u" to accept them
    - overwrites previous snapshot files with new ones
  - if changes are incorrect, fix code and run tests again
- Requires react-test-renderer
  - `npm install -D react-test-renderer`

`__snapshot__` directories should be checked into version control

Jest Watch Mode

- To enable, add **`--watch`** option to **`jest`** command
- Can iteratively change code being tested and tests and have tests rerun automatically on save from any editor/IDE
- Can filter tests to run
  - to filter on file name, press **p** and enter a regex pattern
  - to filter on test name, press **t** and enter a regex pattern
  - to return to running all tests, press **a**
  - to quit watch mode, press **q**
  - to show usage help for additional options, press **w**
- Add **`.only`** and/or **`.skip`** to **`describe`**, **`test`**, and **`it`** function names to focus testing

Enzyme Overview

http://airbnb.io/enzyme/

- Purpose
  - tests interactions with React components
    by finding elements and simulating events on them
- Alternatives
  - React **test-utils** - https://reactjs.org/docs/test-utils.html

Enzyme Details

- To install, **`npm install -D enzyme`**
- Steps
  - render a component with **`mount`**, **`render`**, or **`shallow`**
    - these return a wrapper object representing what was rendered
  - **`find`** an input element whose interaction will be tested
    - by calling **`find`** on wrapper object
    - supports a subset of CSS selectors
  - **`simulate`** an event on it
    - by calling **`simulate`** on wrapper returned by **`find`**
  - make assertions about changes that should occur
    - can use **`expect`** from Jest

`render` performs static rendering.
This generates static HTML.
Assertions can only test
what is rendered.

`shallow` performs shallow rendering.
The component and its
top-level children are rendered,
but not their descendants.
Assertions can test what
the parent renders and can
simulate events on those elements.

`mount` performs full rendering.
The top component and
all its ancestors are rendered.
Assertions can test everything
that is rendered and
simulate events on everything.

Jest Demo

- See **`test`** script in **`package.json`**
- **`npm t`**
  - runs tests in watch mode
  - initially all tests pass
- Modify **`src/demo.js`**
  - remove comma from string returned by **`getGreeting`**
  - change `===` to `!==` in **`handleNameChange`**
  - note errors when tests run automatically
  - fix errors one at a time
  - press **`w`** to see options
  - press **`q`** to quit

Istanbul Overview

https://istanbul.js.org/

- Purpose
  - collects and reports code coverage statistics
- Alternatives
  - nothing notable

Istanbul Details

- Ships with Jest
  - Jest reports code coverage of tests using Istanbul
- If not using Jest, **`npm install -D istanbul`**
- To use from an npm script, add following to **`package.json`**

```
"cover": "open coverage/lcov-report/index.html",
```

!Windows, consider https://www.npmjs.com/package/opener

  - assumes Jest is configured to collect test coverage data
- To exclude code from coverage statistics, use special comment
  `// istanbul ignore word`
  where ***`word`*** is **`next`**, **`if`**, or **`else`**
  - good for code that should never be executed or is very difficult to execute from a test
  - is using this considered cheating?

Istanbul Configuration

- Can configure to fail if coverage is below specified thresholds
- When using Jest, can configure in **`package.json`** or **`jest.json`**
- Example **`jest.json`**

```
{
  "collectCoverage": true,
  "collectCoverageFrom": [
    "src/**/*.js",
    "!src/index.js"
  ],
  "coverageThreshold": {
    "global": {
      "branches": 100,
      "functions": 100,
      "lines": 100,
      "statements": 100
    }
  }
}
```

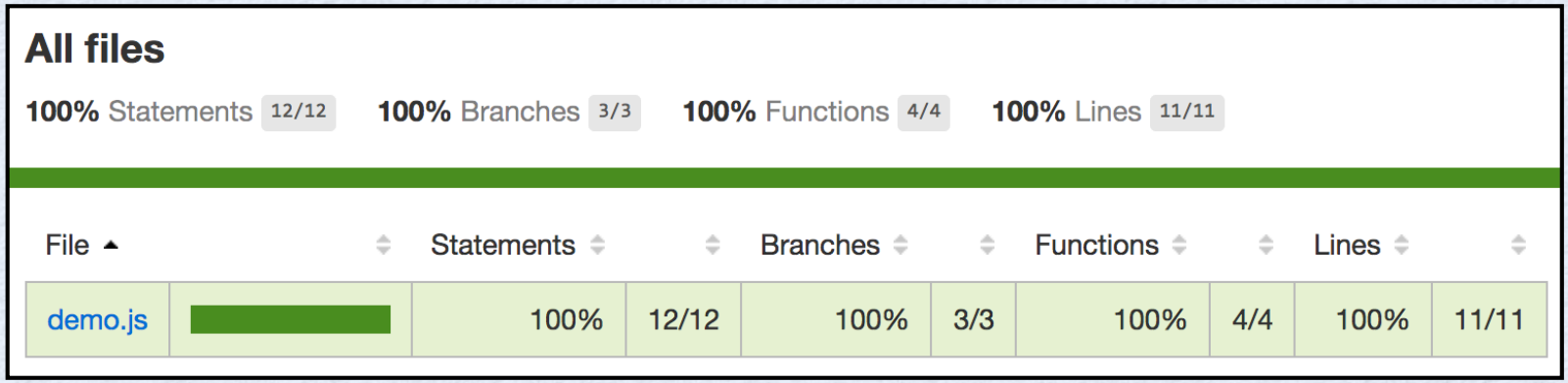
nothing interesting to test in this file

ex. parts of a ternary

can have more than one statement on same line

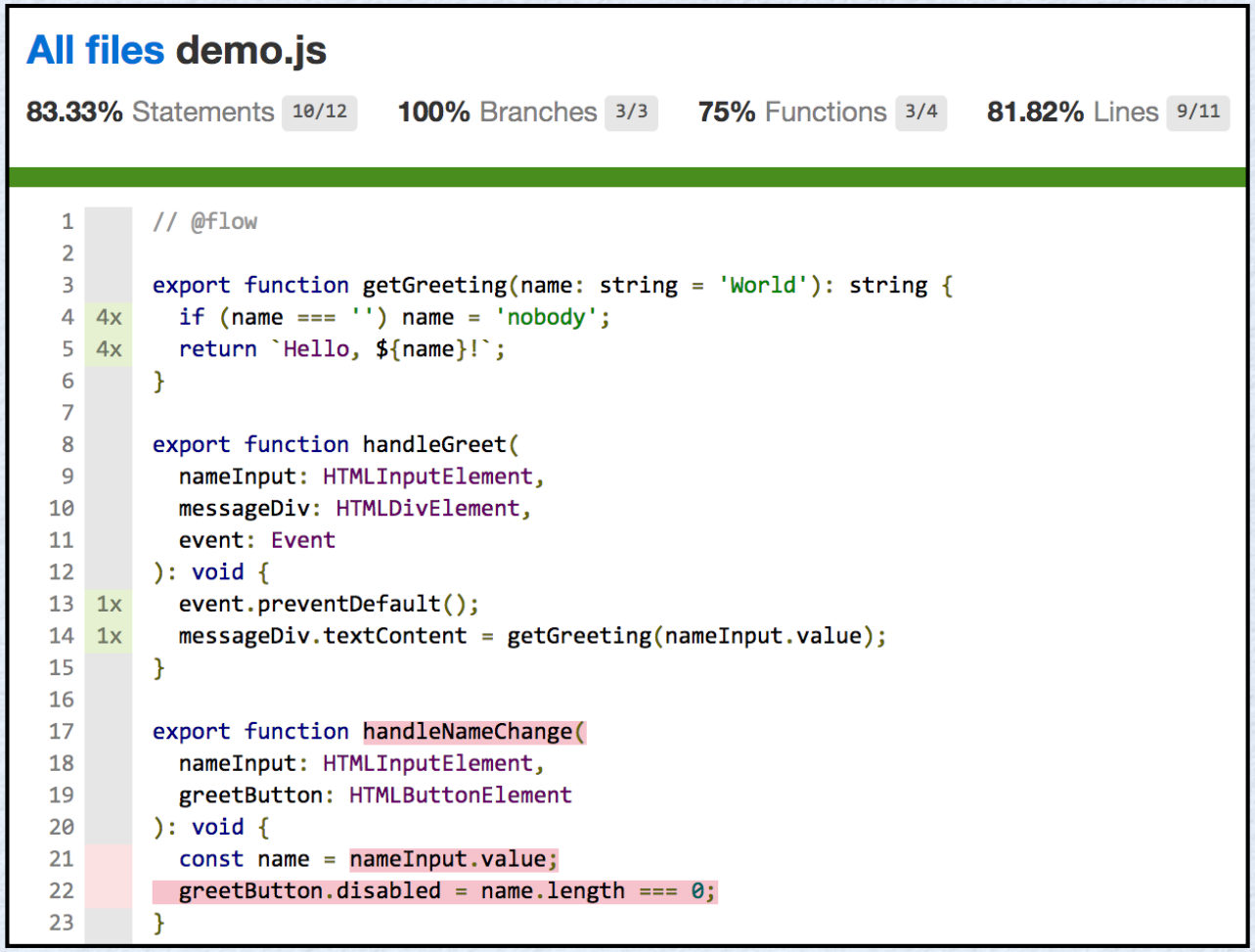
Istanbul Demo ...

- See these scripts in **`package.json`**
  - **`build`** calls **`verify`**
  - **`verify`** calls **`test`**
  - **`test`** runs tests and records coverage in files under **`coverage`** directory
  - **`cover`** opens coverage report in **`coverage/lcov-report/index.html`**
- **`npm run test`**
- **`npm run cover`**



… Istanbul Demo

- Modify **`demo.test.js`**
  - change **`test`** for **`handleNameChange`** to **`test.skip`**
- **`npm run test`**
- Refresh browser
- Click **`demo.js`** to see detail
- Note uncovered code paths



webpack 4 Overview

https://webpack.js.org/

- Purpose
  - bundles JavaScript files and other resources into a single JavaScript file to load into browsers faster
  - enables use of ES6 **`import`**/**`export`** syntax rather than adding a **script** tag for each **`.js`** file in main HTML
- Alternatives
  - **Rollup** - https://rollupjs.org/
  - **Browserify** - http://browserify.org/
  - **Parcel** - https://parceljs.org/

Webpack 4 supports "**zero configuration**" in some cases, but not when using Flow.
Parcel does support this with no configuration!

webpack Loaders

- Optionally uses "loaders" to convert non-JS files into JS files so they can be bundled
- Examples of loaders see https://webpack.js.org/loaders/
  - **babel-loader** - "allows transpiling JavaScript files using Babel"
  - **css-loader** - "interprets `@import` and `url()` like `import/require()` and will resolve them"
  - **sass-loader** - "loads a SASS/SCSS file and compiles it to CSS"
  - **style-loader** - "adds CSS to the DOM by injecting a `<style>` tag"

"**Loaders** enable webpack to process more than just JavaScript files (webpack itself only understands JavaScript). They give you the ability to leverage webpack's bundling capabilities for all kinds of files by converting them to valid modules that webpack can process."

webpack Details

- To install, **`npm install -D webpack webpack-cli`**
- To install any loaders used
  - **`npm install -D babel-loader css-loader style-loader`**
  - and for Babel, **`npm install -D babel-core babel-preset-env`**
- To use from an npm script, add following to **`package.json`**

```
"bundle": "webpack --mode development",
```

defaults to **`production`**

  - development mode optimizes for developer error messages
  - production mode optimizes for size
- Has watch mode to automatically create a new bundle when files change
  - not on by default, add "**`watch: true`**" to configuration
  - but doesn't provide live reload, so manually refresh browser
- Also consider webpack-dev-server
  - "Uses webpack with a development server that provides live reloading."
  - "for development only"

webpack Configuration

- A common complaint about webpack is that it is difficult to configure
- It can be, but here is a simple **`webpack.config.js`**
- **TRY WITHOUT THIS FILE!**

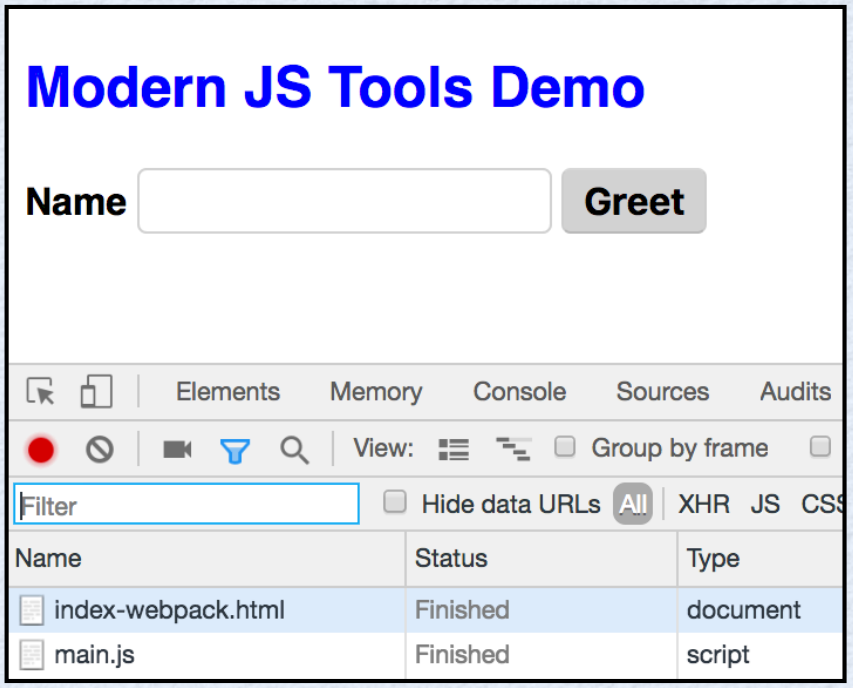
```
module.exports = {
  entry: './src/index.js',
  output: {
    path: __dirname,
    filename: 'dist/main.js'
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: 'babel-loader'
      },
      {
        test: /\.css$/,
        exclude: /node_modules/,
        use: [
          {loader: 'style-loader'},
          {loader: 'css-loader'}
        ]
      },
    ]
  },
  watch: true
};
```

current directory

run in opposite of specified order

webpack Demo

- **`npm run clean`**
- **`npm run bundle`**
  - will continue running due to watch mode
- Note contents of **`list`** directory
  - only **`main.js`**
- Open **`index-webpack.html`** in browser
- Note first two files loaded in devtools Network tab
  - CSS is on line 96 in **`main.js`**



Parcel Overview

https://parceljs.org/

- Purpose
  - transpiles using Babel
  - bundles JavaScript files and other resources into a single JavaScript file to load into browsers faster
  - enables use of ES6 `import`/`export` syntax rather than adding a **script** tag for each `.js` file in main HTML
- From web site
  - "offers blazing fast performance utilizing multicore processing, and requires **zero configuration**"
  - "can take any type of file as an entry point, but an HTML or JavaScript file is a good place to start"
  - "has a development server built in, which **will automatically rebuild your app as you change files** and supports **hot module replacement** for fast development"

live reload

- Alternatives
  - **Rollup** - https://rollupjs.org/
  - **Webpack** - https://webpack.js.org/

Parcel Details

- To install, **`npm install -D parcel-bundler`**
- To process Sass files, **`npm install -D node-sass`**
- To use from an npm script, add following to **`package.json`**

```
"parcel": "parcel index.html",
```

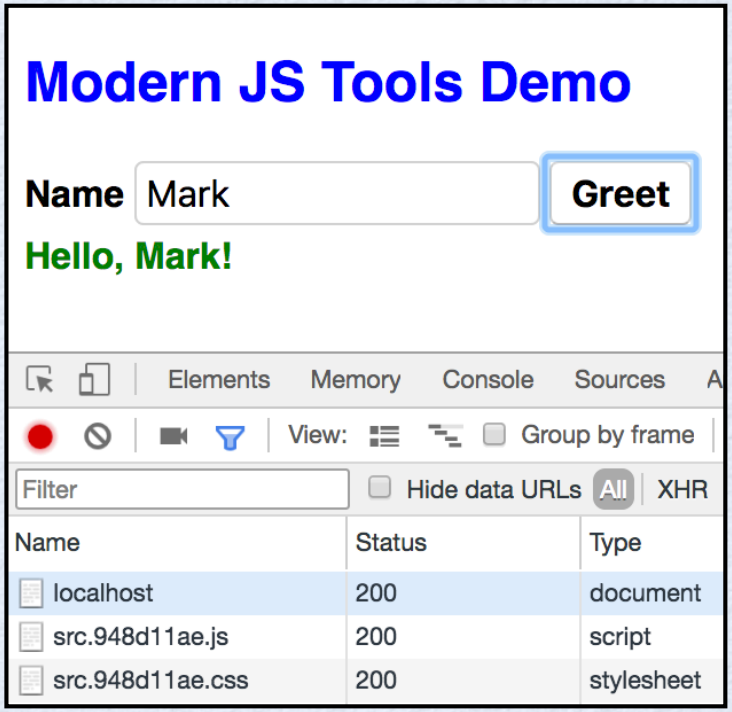
- Writes files to **`dist`** directory,
  starts local server,
  and watches for changes
  - browse localhost:1234
- To build without starting server and without watch

```
"parcel": "parcel build index.html",
```

- For more options, see https://parceljs.org/cli.html

Parcel Demo

- **`npm run clean`**
- **`npm run parcel`**
  - will continue running with watch mode
- Note contents of **`dist`** directory
  - **`index.html`**
  - **`src.something.js`**
  - **`src.something.css`**
- Browse localhost:1234
- Note first **`.js`** and **`.css`** files loaded in devtools Network tab



Browsersync

https://www.browsersync.io/

- Purpose
  - provides live reload of browser for development testing
- Alternatives
  - **livereload** in npm - more complicated setup
  - several commercial tools
- To install,
  `npm install -D browser-sync`
- To start from an npm script, add following to **`package.json`**

```
"sync": "browser-sync start --server --files 'index.html' 'build/bundle.js'",
```

runs a local server

another option for `--files`

```
'**/*.css, **/*.html, **/*.js'
```

- Many more options!

Browsersync Demo

- Modify **`index.html`**
  - change button text **`Greet`** to **`Howdy`**
- Modify **`demo.js`**
  - change **`Hello`** to **`Howdy`** in **`getGreeting`** function
- Note how browser picks up changes without manually rebuilding and refreshing

Husky Overview

https://github.com/typicode/husky

- Purpose
  - "Git hooks made easy"
  - can specify in **`package.json`** instead of in separate files in **`.git/hooks`** directory
  - one use is to configure a Git hook for **`prepush`** that runs ESLint, Prettier, and tests and doesn't push if any of those fail
- Alternatives
  - manually create in **`.git/hooks`** directory (just using Git)
  - pre-commit (in npm)
  - ghooks (in npm)

Husky Detail

- To install, **`npm install -D husky`**
- In **`package.json`**

```
"scripts": {
  ...
  "prepush": "npm run verify",
  "cover": "jest --coverage",
  "verify": "npm-run-all lint format test",
  ...
 }
```

- Can bypass
  - **`git push --no-verify`**
  - mostly useful to push to own branch rather than **`master`**

```
alias pushn='git push --no-verify origin `git rev-parse --abbrev-ref HEAD`'
```

gets name of current branch

Husky Demo

- Modify **`demo.test.js`**
  - change **`test`** for **`handleNameChange`** to **`test.skip`**
  - will cause test coverage to fall below goals
- Commit change
  - **`git commit -av`**
- Attempt to push changes
  - **`git push`**
  - runs the **`lint`**, **`format`**, and **`test`** scripts
  - if any of these fail, the push is not performed

Wrap Up

- Configuring tools requires a bit of work,
  but the information and automation they provide
  is well worth the effort
- Tools reduce time spent performing tedious tasks
  - like finding bugs, formatting code, and running tests
- Go forth and automate!

Bonus Tool: Emmet
an editor plugin for quickly entering HTML and CSS;
see article at https://objectcomputing.com/resources/publications/sett/march-2018-emmet-editor-plugin