Building a Frontend Pipeline

Nathaniel T. Schutta
@ntschutta
ntschutta.io

Ah JavaScript!

It used to be so simple right?

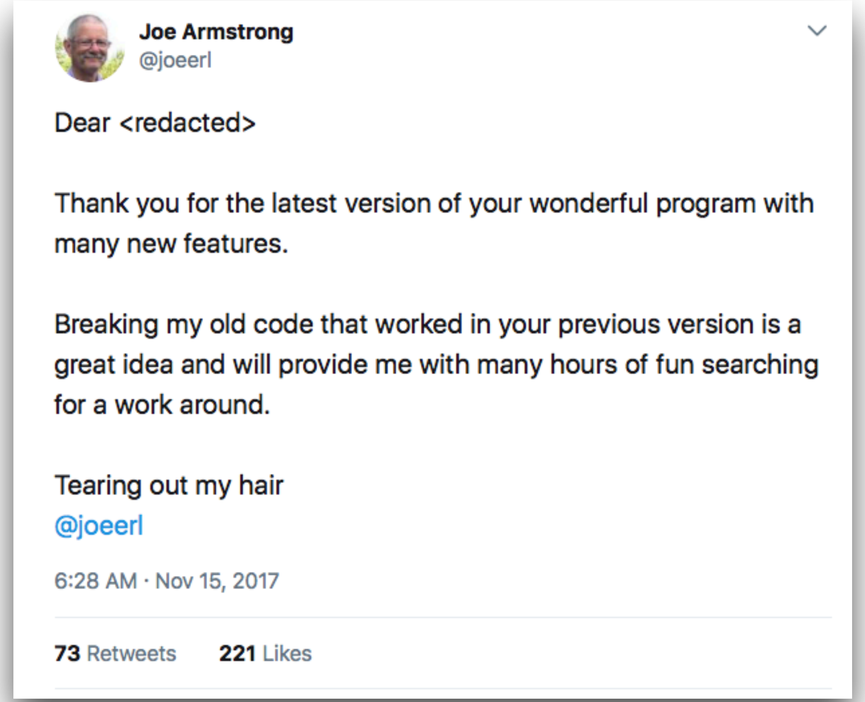
1,500 line **main.js**.

Our apps aren't like that anymore though are they?

JavaScript is a first class citizen now.

Modern applications are many thousands of lines of JavaScript.

We incorporate many libraries and frameworks.



Joe Armstrong
@joeerl

Dear <redacted>

Thank you for the latest version of your wonderful program with many new features.

Breaking my old code that worked in your previous version is a great idea and will provide me with many hours of fun searching for a work around.

Tearing out my hair
@joeerl

6:28 AM · Nov 15, 2017

73 Retweets **221** Likes

https://mobile.twitter.com/joeerl/status/930774515512201216

NPM all the things.

JavaScript itself is evolving.

Browsers are still an issue…

The struggle is real.

What do we do about it?

Why do we need a pipeline at all?

Not your older sister's web apps.

No longer a few thousands lines of JavaScript and jQuery.

Adhoc approach can’t keep up with the complexity of modern JS apps.

Apply the level of discipline to the frontend that we use on the backend.

Linters to ensure our code doesn't fall prey to common errors.

Unit tests to ensure our changes didn’t break anything.

Bundlers to minimize asynchronous requests to our servers.

Transpilers so we can leverage ES6 in older browsers.

Task runers to help tie it all together.

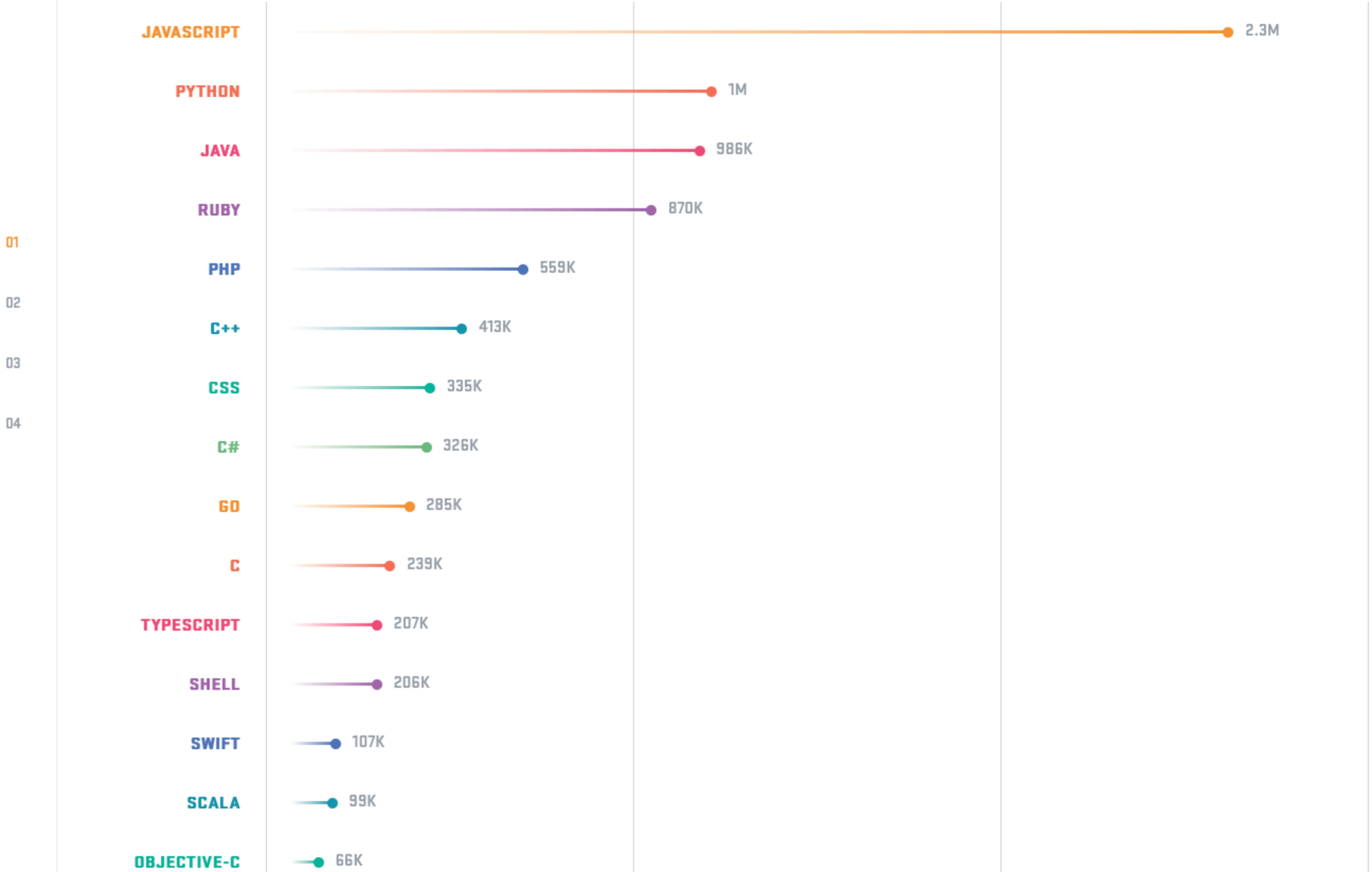
JavaScript usage has exploded.

Number one language on GitHub?

The fifteen most popular languages on GitHub

by opened pull request

GitHub is home to open source projects written in 337 unique programming languages—but especially JavaScript.



Yep.

Not a new phenomena either.

We cannot afford to treat JavaScript as a one off.

The same engineering discipline we use on “server side” code must apply.

Of course for many of us, JavaScript *is* our server side language too!

More than a few options.

Try and clear some of that up today.

Linters.

Certain mistakes can be caught automatically.

Static code analysis.

Find bad patterns.

Check style adherence.

Essentially looking for programer errors.

== instead of ===

That kind of thing.

Can (and should) wire linters into your IDE or text editor of choice.

Better to catch the errors as you go than wait for a build break.

As usual, multiple options.

jshint.com

```
// Hello.
//
// This is JSHint, a tool that helps to detect errors and potential
// problems in your JavaScript code.
//
// To start, simply enter some JavaScript anywhere on this page. Your
// report will appear on the right side.
//
// Additionally, you can toggle specific options in the Configure
// menu.

function main() {
  return 'Hello, World!';
}

main();
```

CONFIGURE

Metrics

There is only one function in this file.

It takes no arguments.

This function contains only one statement.

Cyclomatic complexity number for this function is 1.

JSHint

version 2.9.4

- About
- Documentation
- Install
- Contribute
- Blog

Akin to JSLint.

http://www.jslint.com

More configurable.

Finds common mistakes and defects.

Very few programs pass ;)

15% of programs listed on jshint.com pass. Just saying.

http://jshint.com/about/

Highly configurable.

Can configure inline, via file or in package.json.

Configuration is JSON.

Tune via ‘enforcing’ and ‘relaxing’ options.

Enforcing makes JSHint more strict...

Relaxing does the opposite.

Dozens and dozens of configuration options.

http://www.jshint.com/docs/options/

.jshintrc — ~/talks

.jshintrc

```
    // Environments
    "browser"       : true,     // Web Browser (window, document, etc)
    "browserify"    : false,    // Browserify (node.js code in the browser)
    "couch"         : false,    // CouchDB
    "devel"         : true,     // Development/debugging (alert, confirm, etc)
    "dojo"          : false,    // Dojo Toolkit
    "jasmine"       : false,    // Jasmine
    "jquery"        : false,    // jQuery
    "mocha"         : true,     // Mocha
    "mootools"      : false,    // MooTools
    "node"          : false,    // Node.js
    "nonstandard"   : false,    // Widely adopted globals (escape, unescape, etc)
    "phantom"       : false,    // PhantomJS
    "prototypejs"   : false,    // Prototype and Scriptaculous
    "qunit"         : false,    // QUnit
    "rhino"         : false,    // Rhino
    "shelljs"       : false,    // ShellJS
    "typed"         : false,    // Globals for typed array constructions
    "worker"        : false,    // Web Workers
    "wsh"           : false,    // Windows Scripting Host
    "yui"           : false,    // Yahoo User Interface

    // Custom Globals
    "globals"       : {}        // additional predefined global variables
}
```

.jshintrc 1:1 LF UTF-8 JSON 0 files

You can ask JSHint to ignore a chunk of code.

```
/* jshint ignore:start */
/* jshint ignore:end */
```

Be **very** careful with that…

You can hide individual warnings.

```
/* jshint -W034 */
```

Be careful with that…

You can also relax a number of rules.

Use with caution.

Can also tell JSHint about certain environments…

Browser, Node, etc.

This will define certain global variables.

Think console.

Defaults are pretty good.

But may want to turn a few more things on.

Yes, you can tell JSHint you are using ES6 syntax.

Installed via NPM.

Plugins for Atom, Sublime Text, Emacs, TextMate, VIM, etc.

JetBrains, NetBeans, Eclipse, Visual Studio and more.

Multiple other tools as well - pre commit hooks, JSX wrapper, Ant task…

MIT license.

Not your only linting option.

Some developers have moved on…

ESLint

The pluggable linting utility for JavaScript and JSX

Get Started »

Welcome

ESLint is an open source project originally created by Nicholas C. Zakas in June 2013. Its goal is to provide a pluggable linting utility for JavaScript.

Latest News

- ESLint v4.10.0 released 27 October 2017

About

Learn more about ESLint and why it came about and the general philosophy behind it.

Learn More »

Rules

ESLint comes with a bunch of default rules to get you started. This is the complete list.

See List »

Command Line Interface

ESLint is written to be used primarily on the command line. Learn about its usage here.

CLI Details »

Developer Guide

Love ESLint and want to help make it even awesomer? We've got all the details to get you started.

Start Hacking »

Pluggable linter.

Many, many rules you can tweak…

✔		no-class-assign	disallow reassigning class members
	🔧	no-confusing-arrow	disallow arrow functions where they could be confused with comparisons
✔		no-const-assign	disallow reassigning `const` variables
✔		no-dupe-class-members	disallow duplicate class members
		no-duplicate-imports	disallow duplicate module imports
✔		no-new-symbol	disallow `new` operators with the `Symbol` object
		no-restricted-imports	disallow specified modules when loaded by `import`
✔		no-this-before-super	disallow `this`/`super` before calling `super()` in constructors
	🔧	no-useless-computed-key	disallow unnecessary computed property keys in object literals
		no-useless-constructor	disallow unnecessary constructors
	🔧	no-useless-rename	disallow renaming import, export, and destructured assignments to the same name
	🔧	no-var	require `let` or `const` instead of `var`
	🔧	object-shorthand	require or disallow method and property shorthand syntax for object literals
	🔧	prefer-arrow-callback	require using arrow functions for callbacks
	🔧	prefer-const	require `const` declarations for variables that are never reassigned after declared
		prefer-destructuring	require destructuring from arrays and/or objects
	🔧	prefer-numeric-literals	disallow `parseInt()` and `Number.parseInt()` in favor of binary, octal, and hexadecimal literals
		prefer-rest-params	require rest parameters instead of `arguments`
	🔧	prefer-spread	require spread operators instead of `.apply()`
	🔧	prefer-template	require template literals instead of string concatenation
✔		require-yield	require generator functions to contain `yield`
	🔧	rest-spread-spacing	enforce spacing between rest and spread operators and their expressions
	🔧	sort-imports	enforce sorted import declarations within modules

Configuration is, wait for it, JSON. But you can also use JavaScript or YAML.

`eslint --init` gives you an `.eslintrc` file.

You get to decide the error level.

Off, warn or error.

The recommended rules are on by default - any rule in the list with a ✔.

Rules

Rules in ESLint are grouped by category to help you understand their purpose.

No rules are enabled by default. The `"extends": "eslint:recommended"` property in a configuration file enables rules that report common problems, which have a check mark ✔ below.

The `--fix` option on the command line automatically fixes problems (currently mostly whitespace) reported by rules which have a wrench 🔧 below.

Possible Errors

These rules relate to possible syntax or logic errors in JavaScript code:

		for-direction	enforce “for” loop update clause moving the counter in the right direction.
		getter-return	enforce `return` statements in getters
		no-await-in-loop	disallow `await` inside of loops
✔		no-compare-neg-zero	disallow comparing against -0
✔		no-cond-assign	disallow assignment operators in conditional expressions
✔		no-console	disallow the use of `console`
✔		no-constant-condition	disallow constant expressions in conditions
✔		no-control-regex	disallow control characters in regular expressions
✔	🔧	no-debugger	disallow the use of `debugger`
✔		no-dupe-args	disallow duplicate arguments in `function` definitions
✔		no-dupe-keys	disallow duplicate keys in object literals
✔		no-duplicate-case	disallow duplicate case labels
✔		no-empty	disallow empty block statements
✔		no-empty-character-class	disallow empty character classes in regular expressions
✔		no-ex-assign	disallow reassigning exceptions in `catch` clauses
✔	🔧	no-extra-boolean-cast	disallow unnecessary boolean casts

You can specify
environments and globals.

Various parser options.

Expects ES5 syntax but you can configure other variants.

Also supports JSX syntax.

Using React? Leverage

ESLint-plugin-React.

https://github.com/yannickcr/eslint-plugin-react

Espree is the default parser but you can swap that out if you wish.

ESLint supports 3rd party plugins as well.

You can ask ESLint to ignore a chunk of code.

```
/* eslint-disable */
/* eslint-enable */
```

Be **very** careful with that…

You can also disable a specific rule inline.

```
/* eslint-disable no-alert*/
/* eslint-enable no-alert*/
```

You could actually turn on *all* the rules.

```
"extends": "eslint:all"
```

Please don’t do that!

The rules change with every minor and major version.

Works with Atom, Emacs, Sublime Text, TextMate, VIM, etc.

JetBrains, Eclipse, Visual Studio and more.

Pre commit hooks, command line tools, Mocha integration…

Rules for Angular, Jasmine, React…

Stock configurations for Airbnb, Facebook, Google, Shopify, etc.

And you get an integration, and you get an integration!!!

Curated list - Awesome ESLint.

https://github.com/dustinspecker/awesome-eslint

Installed via NPM.

MIT license.

Use a linter! Catches common mistakes and errors.

Which one should you use?
Up to you!

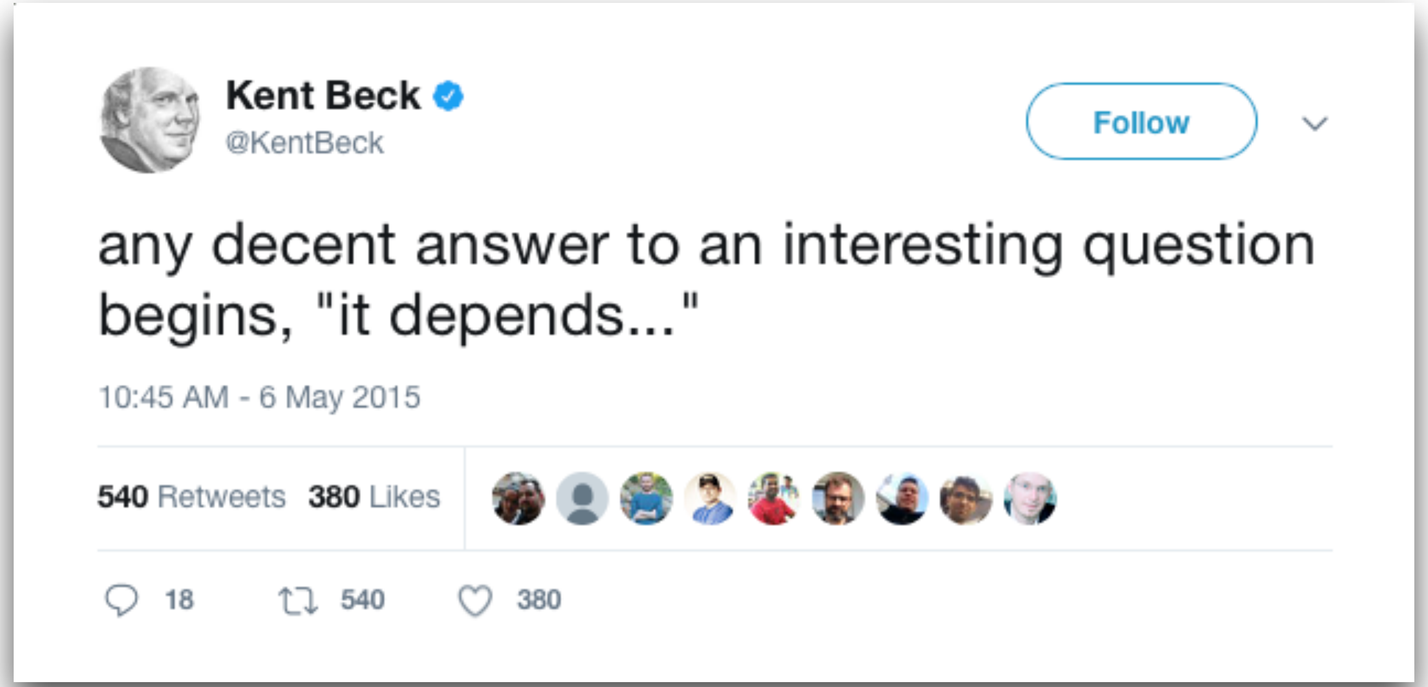
Feels like there is more energy behind ESLint.

More options than you probably know what to do with too!

Spend some time thinking about the configuration options.

There are a lot of options.

Which ones should you turn on?
Which ones should you relax?



https://twitter.com/KentBeck/status/596007846887628801

Is this an existing project?

Or greenfield?

Tempting to crank all the knobs up to 11.

PAZI - MINE
ПАЗИ - МИНЕ
UKLANJANJE OVOG ZNAKA
KRIVIČNO JE DJELO
УКЛАЊАЊЕ ОВОГ ЗНАКА
КРИВИЧНО ЈЕ ДЈЕЛО

Existing projects probably have some cruft.

Don’t overwhelm the team with warnings!

Easy to get discouraged.

“We have so many warnings…”

Fixing 10 or 20 hardly makes a dent.

Easy to miss when you add another one…

Start small.

Expect some…discussion around which rules to use.

Pick one or two.

Turn them on.

Should result in a
manageable set of warnings.

Fix them!

An iteration or two later,
add another rule or two.

Rinse, repeat.

“Ratchet up.”

Over time, you'll have a very complete set.

And a clean code base.

With a new code base, it is easier to start big.

Start clean, stay clean.

Don’t be afraid to tweak!

The warnings are
there to help, not hurt.

Testing.

We need to test our code.

That isn’t controversial.

Right?!?

We can have a debate about unit vs. integration tests vs. UI tests.

We need it all…up to you to determine how much of each.

Think about it like a pyramid…

An effective test automation strategy calls for automating tests at three different levels: unit, service and UI.

mountaingoatsoftware.com

The Forgotten Layer of the Test Automation Pyramid

by Mike Cohn • 46 Comments

Even before the ascendancy of agile methodologies like Scrum, we knew we should automate our tests. But we didn't. Automated tests were considered expensive to write and were often written months, or in some cases years, after a feature had been programmed. One reason teams found it difficult to write tests sooner was because they were automating at the wrong level. An effective test automation strategy calls for automating tests at three different levels, as shown in the figure below, which depicts the test automation pyramid.

As you might suspect, there are multiple testing tools at our disposal.

jasmine.github.io

Jasmine

Behavior-Driven JavaScript

Fork me on GitHub

Jasmine

GET STARTED DOCS SUPPORT RELEASES GITHUB

FAST

Low overhead, no external dependencies.

BATTERIES INCLUDED

Comes out of the box with everything you need to test your code.

NODE AND BROWSER

Run your browser tests and Node.js tests with the same framework.

```
describe("A suite is just a function", function() {
  var a;

  it("and so is a spec", function() {
    a = true;

    expect(a).toBe(true);
  });
});
```

Sample Code

Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests.

GET STARTED

Jasmine

OPEN SOURCE

PIVOTAL LABS

BDD JavaScript testing framework.

More readable?

Doesn’t require the
DOM or other libraries.

Suites with specs.

Set of matchers.

Can create custom matchers.

Group related specs with '*describe*'.

Describe blocks can be nested.

Includes beforeEach and afterEach for setup/teardown.

Specs and blocks can be disabled...

Don’t forget to turn them back on!

Includes spies aka test doubles.

Can mock out the JavaScript clock.

Includes a *waitFor* to test asynchronous cases.

Runs in any JavaScript environment.

Include Jasmine files,
page is now a runner.

Results read like a specification.

Runs across common browsers as well as Node.

Download or install via NPM.

MIT license.

Don’t like Jasmine? You have options!

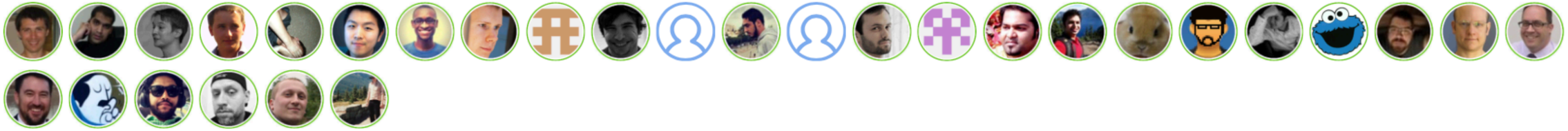
simple, flexible, fun

Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing *simple* and *fun*. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on GitHub.

gitter join chat backers 57 sponsors 10

BACKERS

Find Mocha helpful? Become a backer and support Mocha with a monthly donation.



SPONSORS

Use Mocha at Work? Ask your manager or marketing team if they'd help support our project. Your company's logo will also be displayed on npmjs.com and our GitHub repository.

Another very popular JavaScript testing tool.

Perhaps “meta” tool.

Can use different assert libraries.

Allows for BDD or
“assert” based testing.

Includes code coverage.

Highlights slow tests.

Detects globals.

Can run tests based on a regular expression...

Various reporting mechanisms.

Dot matrix, spec, TAP,
landing strip, list, JSON...

Also includes JSON and HTML coverage reports.

Interface system supports various testing DSLs.

should, expect, chai...

whatever you prefer using.

Often used with Chai.

Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework.

Download Chai

4.1.2 / 2017-08-31

for Node Another platform? Browser Rails

The **chai** package is available on npm.

```
$ npm install chai
```

View Node Guide

Issues | Fork on GitHub | Releases | Google Group | Build Status

Getting Started

Learn how to install and use Chai through a series of guided walkthroughs.

API Documentation

Explore the BDD & TDD language specifications for all available assertions.

Plugin Directory

Extend Chai's with additional assertions and vendor integration.

Chai has several interfaces that allow the developer to choose the most comfortable. The chain-capable BDD styles provide an expressive language & readable style, while the TDD assert style provides a more classical feel.

Should

```
chai.should();

foo.should.be.a('string');
foo.should.equal('bar');
foo.should.have.lengthOf(3);
tea.should.have.property('flavors')
  .with.lengthOf(3);
```

Visit Should Guide

Expect

```
var expect = chai.expect;

expect(foo).to.be.a('string');
expect(foo).to.equal('bar');
expect(foo).to.have.lengthOf(3);
expect(tea).to.have.property('flavors')
  .with.lengthOf(3);
```

Visit Expect Guide

Assert

```
var assert = chai.assert;

assert.typeOf(foo, 'string');
assert.equal(foo, 'bar');
assert.lengthOf(foo, 3)
assert.property(tea, 'flavors');
assert.lengthOf(tea.flavors, 3);
```

Visit Assert Guide

Plugins extend Chai's assertions to new contexts such as vendor integration & object construction. Developers can build their own plugins to share with community or use the plugin pattern to DRY up existing tests.

Just an assertion library.

Allows you to use BDD or TDD.

Supports asynchronous testing out of the box.

Can set test and suite level timeouts.

Mocha looks for /test/*.js.

Works with sinon.js for spies, stubs and mocks.

Headless testing via PhantomJS.

Basically, everything you expect out of a modern testing library!

Writing tests is more important than the tool you use to write them.

Doesn’t matter which one you use!

But use **something**!

Bundlers.

At build time, generates a bundle file.

One unified JavaScript file.

Your code, libraries, frameworks, all the dependencies - one file.

Saves the browser from downloading dozens (or hundreds) of files.

Why do we use bundlers?

First, a bit of history.

Common for developers
to leverage modules.

Or packages, namespaces, etc.

Nothing new really - just a pattern.

In some languages, modules are a first class concept.

In others it is very fuzzy.
Or lacking all together.

Just a way to encapsulate code.

Allows us to abstract functionality to a library or framework.

Simplifies reuse.

ES5 and earlier has no allowance for modules.

And that was fine - for a while.

But applications have grown.

Two common patterns in older JavaScript code…

Immediately Invoked Function Expression aka IIFE.

Wrap a function in parenthesis makes it a function expression…

Which we can then immediately call.

```
(function(){
  console.log('test');
})()
```

Code is encapsulated.

Variables inside the function stay within the function's closure.

Doesn’t help us with dependency management though.

IIFE wasn't the only approach. Some used the Revealing Module pattern.

Basically we assign a return value.

And we use that variable to access the “module’s API”.

```
var myModule = function(){
  function logTest() {
    console.log('test');
  }
  return {
    logTest: logTest
  }
}()
```

Basically gives us the same benefits and drawbacks as IIFE.

Unsurprisingly, clever developers yearned for more.

Started defining their own module formats for JavaScript.

Asynchronous Module Definition (AMD), CommonJS…

Universal Module Definition (UMD), System.register…

And of course ES6 has
a module format!

Module loaders execute at runtime.

Module loader looks at the module format and downloads required files.

Which often results in downloading dozens (or hundreds) of files.

"I only required calendar but 87 files were downloaded…"

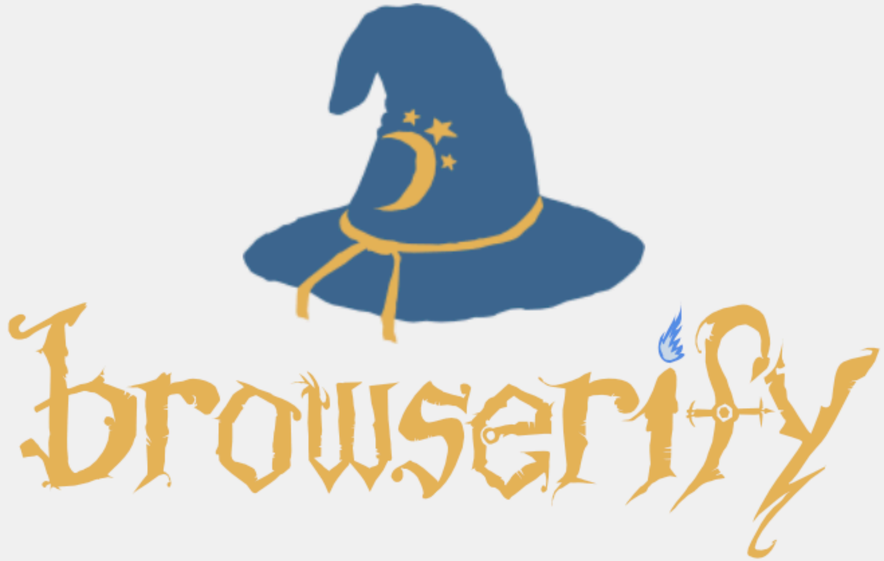
RequireJS and SystemJS are popular.

Module bundler replaces
a module loader.

Generates a bundle of code at build time. Browser fetches said bundle.

As you can guess, there are multiple bundler options to choose from!

browserify.org



Browserify lets you require('modules') in the browser by bundling up all of your dependencies.

INSTALL | DOCUMENTATION | HELP + ARTICLES | SWEET DEMOS

Install

USE BROWSERIFY FROM THE COMMAND LINE

First install node, which ships with npm. Then do:

Essentially brings Node's `require` to the browser.

CommonJS modules.

Allows you to use NPM packages in the browser.

Use `require()` to load code from files.

But we have to export something for it to be importable.

module.exports

Works for more than just functions, works for any kind of value.

Starts at your entry point and looks for `require()` calls in the AST.

Does this recursively throughout the dependency graph.

Concatenates those files together into one file that has everything.

My Java build doesn't concatenate the files, why should I in JavaScript?

Think about the typical
`<script>` tag approach…

Every new file must be added. To every HTML page. In the right order.

Order matters. And it hurts.

Also results in multiple round trip requests to the server.

One major downside to bundling though - edit/save/refresh breaks.

A full build of your JavaScript can severely hamper developer velocity.

Luckily, we don’t have to do that!

Some tools even support live reload saving you a browser refresh!

watchify.

https://www.npmjs.com/package/watchify

Change a file and the bundle is automatically recreated.

beefy.

https://www.npmjs.com/package/beefy

Local development server that supports live reload on change.

Browserify-HMR.

https://github.com/AgentME/browserify-hmr

Hot Module Replacement.

The list just goes on and on!

Bundling does lead you to add
something for developer productivity.

Still less time than wrangling
`<script>` tags though!

Installed via NPM.

MIT licensed but with some Joyent bits… ¯_(ツ)_/¯ IANAL.

Browserify is not our only option!

No one could have predicted this.

webpack.github.io

Fork me on GitHub

webpack

MODULE BUNDLER

webpack 1.x

Getting started by reading the **documentation**, which also contains an **introduction** and a **tutorial**.

webpack 2.x

Getting started by reading the **documentation**, which also contains an **introduction** and a **tutorial**.

G+ Tweet

Module bundler - but then you knew that.

Builds a dependency graph and pulls everything into one (or more) bundle.

Highly configurable.

Four core concepts: Entry, Output, Loaders and Plugins.

Entry - where to start the dependency graph.

Based on said entry point, what other modules do we depend on.

There can be multiple entry points.

Output - where do you want the bundle and what do you want to call it.

Also referred to as emitted, produced or discharged.

Loaders - extend webpack beyond JavaScript to other file types.

Essentially turns any file into a module that can be included.

A file that passes **this test** should be transformed with **this loader**.

Plugins - extensions to webpack.

Minify, optimize, define variables, etc.

Simply require the plugin, configure as necessary and away you go.

Multiple plugins built in, along with several others you can leverage.

https://webpack.js.org/plugins/

webpack does perform
a bit of transpiling.

Gives us **import** and **export** support.

Again, bundling breaks the edit/save/refresh lifecycle.

As you might expect, there are options in the webpack universe!

We can use source maps to, well map errors and warnings to the original file.

webpack includes a watch mode:
change a file, rebuild.

Just requires a browser refresh.

Violates the lazy developer ethos…

That which I do more than once should be automated away!

webpack-dev-server

gives us live reloading!

Make a change and once it is done building, it will be loaded!

webpack supports Hot Module Replacement (HMR) as well.

Check out the development guide for full setup information.

https://webpack.js.org/guides/development/

Installed via NPM or yarn.

MIT license.

Transpilers.

That term “transpile” has come up a few times - what does it mean?

Not a great word frankly…

And not a new idea as such.

Just a fancy way of saying
“source-to-source” compiling.

Prevalent in the frontend space.
For better or worse.

We can blame GWT and CoffeeScript.

Take Java and make it JavaScript.

Make my ES6 code run
in older browsers.

TypeScript, ClojureScript.
The list goes on and on!

Anything you can do I can do better…

Allows us to use modern approaches without worrying about browsers.

Remember how much fun the Netscape/IE wars were?!?

Too soon?

Have things gotten any better?

kangax.github.io

ECMAScript 5 6 2016+ next intl non-standard compatibility table

by kangax & webbedspace & zloirock Fork 537

Sort by Engine types | Show obsolete platforms | Show unstable platforms

V8 | SpiderMonkey | JavaScriptCore | Chakra | Carakan | KJS | Other

Minor difference (1 point) | Small feature (2 points) | Medium feature (4 points) | Large feature (8 points)

			Compilers/polyfills					Desktop browsers																Servers/runtimes			Mobile
Feature name	Current browser		Traceur	Babel + core-js[2]	Closure	Type-Script + core-js	es6-shim	Konq 4.14[3]	IE 11	Edge 15	Edge 16	FF 52 ESR	FF 56	CH 61, OP 48[1]	CH 62, OP 49[1]	SF 10.1	SF 11	PJS	XS6	JXA	Node 4[5]	Node >=6.5 <7[5]	Node >=8.7 <9[5]	DUK 1.8	DUK 2.2	iOS >=10.3 <11	iOS 11
Optimisation																											
proper tail calls (tail call optimisation)	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	0/2	2/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2		
Syntax																											
default function parameters	4/7	4/7	5/7	5/7	0/7	0/7	0/7	7/7	7/7	6/7	7/7	7/7	7/7	7/7	7/7	0/7	7/7	0/7	0/7	7/7	7/7	0/7	0/7	7/7	7/7		
rest parameters	4/5	3/5	2/5	4/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	5/5	0/5	0/5	5/5	5/5	0/5	0/5	5/5	5/5		
spread (...) operator	15/15	13/15	12/15	4/15	0/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	0/15	15/15	11/15	0/15	15/15	15/15	0/15	0/15	15/15	15/15		
object literal extensions	6/6	6/6	4/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	0/6	6/6	5/6	6/6	6/6	6/6	0/6	4/6	6/6	6/6		
for..of loops	9/9	9/9	6/9	3/9	0/9	0/9	0/9	9/9	9/9	7/9	9/9	9/9	9/9	9/9	9/9	0/9	9/9	8/9	7/9	9/9	9/9	0/9	0/9	9/9	9/9		
octal and binary literals	2/4	4/4	4/4	4/4	2/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	4/4	4/4	4/4	4/4	4/4	0/4	4/4	4/4	4/4		
template literals	4/5	4/5	3/5	3/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	5/5	5/5	5/5	5/5	5/5	0/5	0/5	5/5	5/5		
RegExp "y" and "u" flags	3/5	3/5	0/5	0/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	2/5	0/5	0/5	5/5	5/5	0/5	0/5	5/5	5/5		
destructuring, declarations	20/22	21/22	20/22	15/22	0/22	0/22	0/22	22/22	22/22	21/22	22/22	22/22	22/22	22/22	22/22	0/22	21/22	19/22	0/22	22/22	22/22	0/22	0/22	22/22	22/22		
destructuring, assignment	23/24	24/24	21/24	19/24	0/24	0/24	0/24	24/24	24/24	23/24	24/24	24/24	24/24	24/24	24/24	0/24	24/24	21/24	0/24	24/24	24/24	0/24	0/24	24/24	24/24		
destructuring, parameters	19/24	21/24	18/24	16/24	0/24	0/24	0/24	23/24	23/24	21/24	24/24	24/24	24/24	24/24	24/24	0/24	23/24	18/24	0/24	24/24	24/24	0/24	0/24	24/24	24/24		
Unicode code point escapes	1/2	1/2	1/2	1/2	0/2	0/2	0/2	2/2	2/2	1/2	2/2	2/2	2/2	2/2	2/2	0/2	2/2	2/2	2/2	2/2	2/2	0/2	2/2	2/2	2/2		
new.target	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	2/2	0/2	0/2	2/2	2/2	0/2	1/2	2/2	2/2		
Bindings																											
const	14/16	14/16	14/16	14/16	0/16	2/16	12/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	1/16	16/16	10/16	9/16	16/16	16/16	1/16	2/16	16/16	16/16		
let	10/12	10/12	10/12	10/12	0/12	0/12	10/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	0/12	12/12	0/12	6/12	12/12	12/12	0/12	0/12	12/12	12/12		
block-level function declaration[15]	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes		
Functions																											
arrow functions	11/13	9/13	10/13	9/13	0/13	0/13	0/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	0/13	12/13	0/13	9/13	13/13	13/13	0/13	0/13	13/13	13/13		
class	17/24	19/24	13/24	19/24	0/24	0/24	0/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	0/24	24/24	18/24	0/24	24/24	24/24	0/24	0/24	24/24	24/24		
super	7/8	4/8	6/8	7/8	0/8	0/8	0/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	0/8	8/8	7/8	0/8	8/8	8/8	0/8	0/8	8/8	8/8		
generators	24/27	24/27	16/27	0/27	0/27	0/27	0/27	27/27	27/27	25/27	27/27	27/27	27/27	27/27	27/27	0/27	27/27	0/27	20/27	27/27	27/27	0/27	0/27	27/27	27/27		
Built-ins																											
typed arrays	0/46	45/46	0/46	45/46	0/46	8/46	16/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	18/46	46/46	46/46	43/46	46/46	46/46	17/46	20/46	46/46	46/46		
Map	14/19	19/19	14/19	19/19	15/19	0/19	8/19	19/19	19/19	18/19	19/19	19/19	19/19	19/19	19/19	0/19	19/19	18/19	17/19	19/19	19/19	0/19	0/19	19/19	19/19		
Set	14/19	19/19	14/19	19/19	15/19	0/19	8/19	19/19	19/19	18/19	19/19	19/19	19/19	19/19	19/19	0/19	19/19	18/19	17/19	19/19	19/19	0/19	0/19	19/19	19/19		

Tomorrow's JavaScript today?

Babel is a JavaScript compiler.

Use next generation JavaScript, today.

Put in next-gen JavaScript	Get browser-compatible JavaScript out
`var [a,,`	

Check out our REPL to experiment more with Babel!

Latest From Our Blog: Babel Turns Three

Ready to get started?

Install the Babel CLI and a preset

Shell Copy

```
npm install --save-dev babel-cli babel-preset-env
```

Create a `.babelrc` file (or use your package.json)

JSON Copy

```
{
```

Wouldn’t it be nice to use all the new JavaScript hotness today?

Browser support isn't, and probably never will be, universal.

Babel gives us “syntax transformers”.

But it also supports polyfill, JSX and Flow.

Plugins allows you to piece together the transforms your project needs.

But how do I debug the transformed code? It isn’t what I wrote!

Source maps for the win.

What does this look like?

Use the REPL!

https://babeljs.io/repl/

babeljs.io

BABEL Learn ES2015 Docs Try it out Blog FAQ Team Donate Forum

Settings

- Evaluate
- Line Wrap
- Minify

Presets

Env Preset

```
"use strict";

class CoffeeDrink {

  constructor (size, name, shots) {
    this.size  = size;
    this.name = name;
    this.shots = shots || 2;
  }

  sayMyName () {
    console.log(`I have a drink for ${this.name}.`)
  }

}


{
    let name = "Tomorrow's JavaScript Today!";
}
```

```
"use strict";

var _createClass = function () { function defineProperties(target, props) { for (var i = 0; i < props.length; i++) { var descriptor = props[i]; descriptor.enumerable = descriptor.enumerable || false; descriptor.configurable = true; if ("value" in descriptor) descriptor.writable = true; Object.defineProperty(target, descriptor.key, descriptor); } } return function (Constructor, protoProps, staticProps) { if (protoProps) defineProperties(Constructor.prototype, protoProps); if (staticProps) defineProperties(Constructor, staticProps); return Constructor; }; }();

function _classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw new TypeError("Cannot call a class as a function"); } }

var CoffeeDrink = function () {
  function CoffeeDrink(size, name, shots) {
    _classCallCheck(this, CoffeeDrink);

    this.size = size;
    this.name = name;
    this.shots = shots || 2;
  }

  _createClass(CoffeeDrink, [{
    key: "sayMyName",
    value: function sayMyName() {
      console.log("I have a drink for " + this.name + ".");
    }
  }]);

  return CoffeeDrink;
}();

{
  var name = "Tomorrow's JavaScript Today!";
}
```

v6.26.0

But you'll want to use the CLI.

Along with your favorite build tool.

You name it, it is most likely supported.

Configure via `.babelrc` file.
Imagine that.

If you are doing React you are (probably) using Babel.

And the babel-preset-react.

Works with Atom, Sublime Text, VIM, Visual Studio, WebStorm, etc.

Installed via NPM.

MIT license.

Other options are largely other languages transforming to JavaScript.

Haste, Elm, Amber, ClojureScript, TypeScript, Opa, Dart, etc.

Task runners.

Obviously, you could manage these tools by hand. But you shouldn't.

Automation: #winning.

Ensures that we're doing the same thing. Every. Single. Time.

Run tests, minify, linting, bundle…

Not a new idea…

Akin to make, Rake, Ant, Gradle, etc.

Fertile ground. A veritable plethora of options in the build space.

Many people just use npm scripts.

Several supported “stages” plus the ability to run arbitrary scripts.

npm run foo

You can cover a lot of ground using nothing more than npm scripts.

And it is one less tool you have to install, update, learn etc.

Bit more of an…artisanal approach.

Of course there are full fledged task runner/build tools.

GRUNT

The JavaScript Task Runner

Latest Version

- Stable: v1.0.1 (npm)
- Development: master (GitHub)

Ads by Bocoup.

Latest News

Grunt 1.0.0 released
April 04, 2016

Grunt 1.0.0 rc1 released
February 11, 2016

Grunt 0.4.5 released
May 12, 2014

Getting Started

Read the Getting Started guide for instructions on using Grunt in your projects. Once you've done that, check out a Sample Gruntfile and learn more about Configuring Tasks.

Why use a task runner?

In one word: automation. The less work you have to do when performing repetitive tasks like minification, compilation, unit testing, linting, etc, the easier your job becomes. After you've configured it through a Gruntfile, a task runner can do most of that mundane work for you—and your team—with basically zero effort.

Why use Grunt?

The Grunt ecosystem is huge and it's growing every day. With literally hundreds of plugins to choose from, you can use Grunt to automate just about anything with a minimum of effort. If someone hasn't already built what you need, authoring and publishing your own Grunt plugin to npm is a breeze. See how to get started.

Available Grunt plugins

Many of the tasks you need are already available as Grunt Plugins, and new plugins are published every day. While the plugin listing is more complete, here's a few you may have heard of.

The first really popular task runner in the JavaScript space.

Huge plugin ecosystem.

Plugins for just about every task as well as common libraries/frameworks.

Free to create custom tasks as well.

Grunt is “ant like” in its approach.

Load a task, provide the configuration.

Good documentation,
easy to get started.

As you add more and more tasks, configuration can get out of hand.

Falling a bit out of favor.

Last commit…was months ago.

Last release…more than a year ago.

Installed via NPM.

MIT license.

Many teams have moved on…

gulpjs.com

Gulp

Docs Plugins Twitter Contribute

Automate and enhance your workflow

```
$ npm install gulp-cli -g
$ npm install gulp -D
$ touch gulpfile.js
$ gulp --help
```

gulp is a toolkit for automating painful or time-consuming tasks in your development workflow, so you can stop messing around and build something.

GET STARTED

Came along well after Grunt.

Alternative approach to automation.

Code over configuration.

More Gradle like.

Large plugin ecosystem (though ultimately smaller than Grunt's).

“The streaming build system.”

Leverages Node streams.

Node streams can be…challenging for developers to understand.

“Streams are Node’s best and most misunderstood idea.”

https://medium.freecodecamp.org/node-js-streams-everything-you-need-to-know-c9141306be93

Just a collection of data - and it may not all fit into memory.

But data is only part of the picture.

Think composing Unix commands via pipes.

Streams let us do the same thing.

Many Node modules implement the streaming interface: HTTP, TCP, zlib.

Gulp takes advantage of streams - leads to faster, more efficient builds.

But the learning curve can be steep.

At least for those lacking a solid Node background.

Documentation isn’t stellar.

MIT license.

Don’t like any of those options?

No problem!

And you get a build tool and you get a build tool and you get a build tool…

broccolijs.com

HOME GETTING STARTED DOCS PLUGINS GITHUB

Broccoli.js – The asset pipeline for ambitious applications.

```
$ npm install -g broccoli-cli
```

GET STARTED

```
/* Brocfile.js */

var compileSass = require('broccoli-sass');
var filterCoffeeScript = require('broccoli-coffee');
var mergeTrees = require('broccoli-merge-trees');

var sassDir = 'scss';
var coffeeDir = 'coffeescript';

var styles = compileSass([sassDir], 'app.scss', 'app.css');
var scripts = filterCoffeeScript(coffeeDir);

module.exports = mergeTrees([styles, scripts]);
```

Blazing fast

Less (code) is more

brunch.io

Brunch

Home Docs Plugins Skeletons In Production Community

Seeing your build tool in nightmares? Try Brunch!

Brunch lets you focus on what matters most — solving real problems instead of messing around with the glue.

Get Started

Small configs

By being opinionated about your build pipeline, Brunch is able to provide a smooth and fast experience, and makes your config files take a drastic cut.

Three simple commands

It doesn't take much to get around with brunch:

- `brunch new` to create a new project
- `brunch build` to build
- `brunch watch` to live-compile

Productivity and happiness

- NPM support
- source maps out-of-the-box
- fast from-zero builds
- incremental builds
- and more

Yes, the configs are **much** simpler

TYPICAL GULP CONFIG – VS – TYPICAL BRUNCH CONFIG

```
var app, base, concat, directory, gulp, gutil, hostname, path, refresh, sass, uglify, del, connect, autoprefixer, babel;

var autoPrefixBrowserList = ['last 2 version', 'safari 5', 'ie 8', 'ie 9', 'opera 12.1', 'ios 6', 'android 4'];

gulp        = require('gulp');
gutil       = require('gulp-util');
concat      = require('gulp-concat');
uglify      = require('gulp-uglify');
sass        = require('gulp-sass');
connect     = require('gulp-connect');
del         = require('del');
autoprefixer = require('gulp-autoprefixer');
babel        = require('gulp-babel');
```

```
exports.files = {
  javascripts: {
    joinTo: {
      'vendor.js': /^(?!app)/,
      'app.js': /^app/
```

There are others though more than a few appear abandoned…

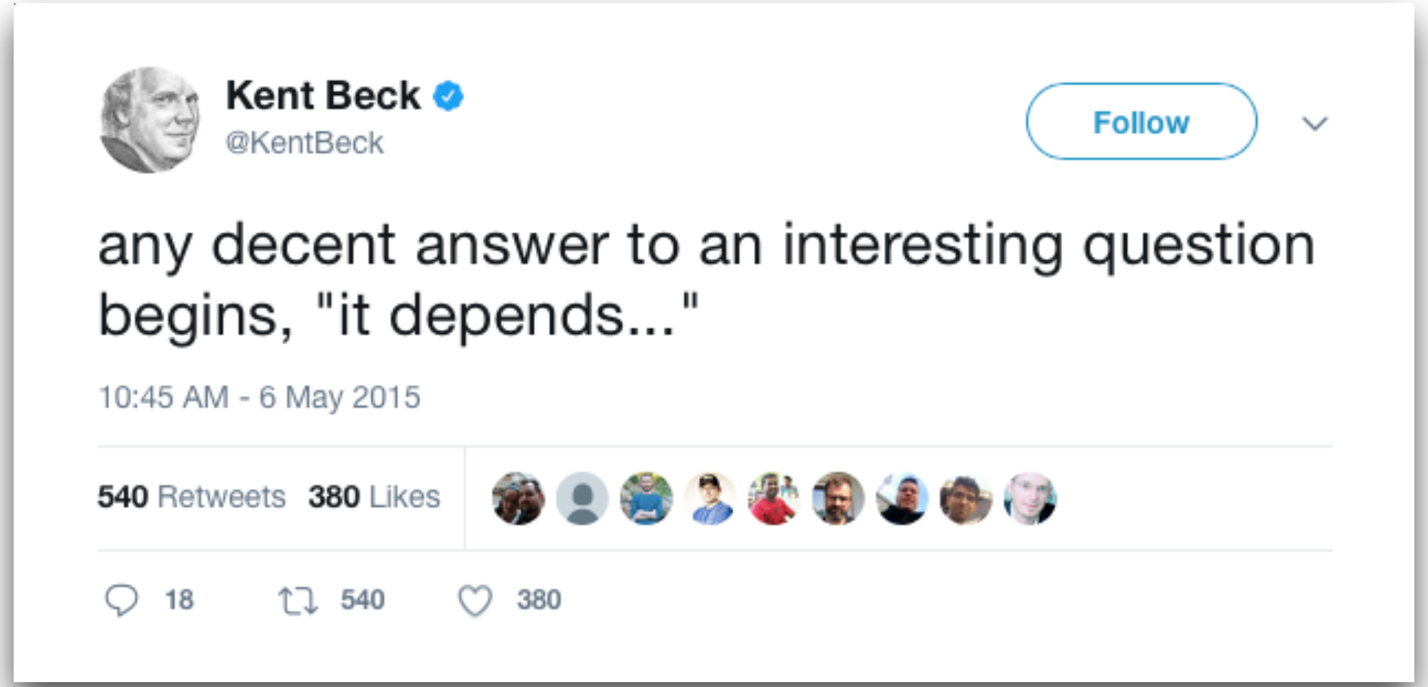
Anyone publish one today?

Putting it all together.

We've touched on a number of tools!

Leads to a very natural question…

Which one should *I* use?



https://twitter.com/KentBeck/status/596007846887628801

Don’t like what I’ve shown you?

Great, use what works for your team!

Not sure what that is?

Perform a time boxed eval.

But use something!

Many of these tools are very new.

Expect some volatility.

Can’t treat JavaScript like
a toy language today.

You don’t need to adopt every one of these today.

Pick your pain point.

Introduce one or two tools.

Ratchet up.

After a few weeks or months, add another.

Rinse repeat.

Take the time to wire these into your editor or IDE of choice.

Make the right thing to do,
the easy thing to do.

Some projects have tried to simplify.

“Plain vanilla JavaScript.”

It can be done.

But it may not support your needs.

Do what is right.

We've come a long way!

JavaScript is a first class citizen.

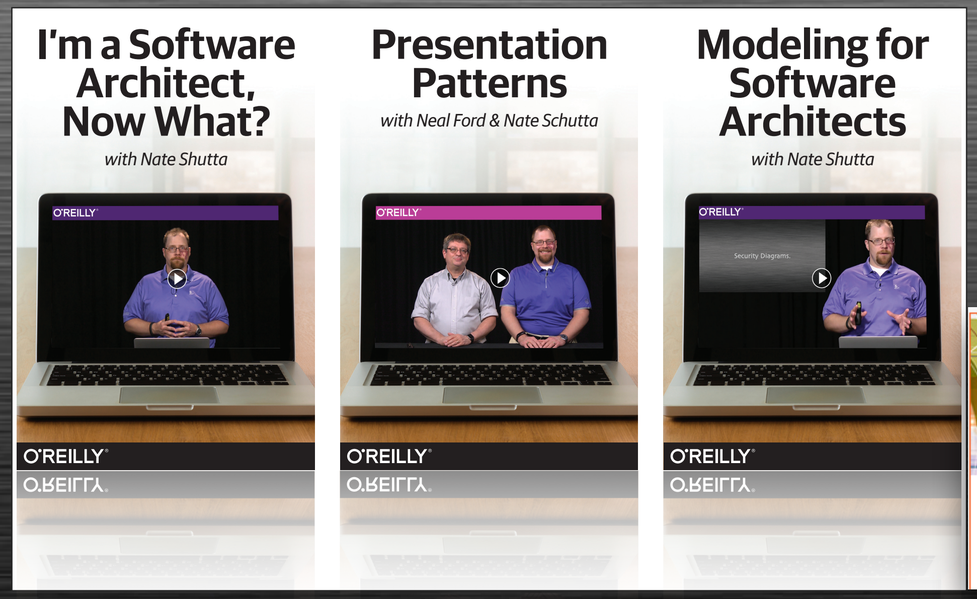
We have the tools to prove it.

No excuses.

Good luck!

Questions?

Thanks!



Nathaniel T. Schutta
@ntschutta
ntschutta.io