SpiderMonkey Parser API:

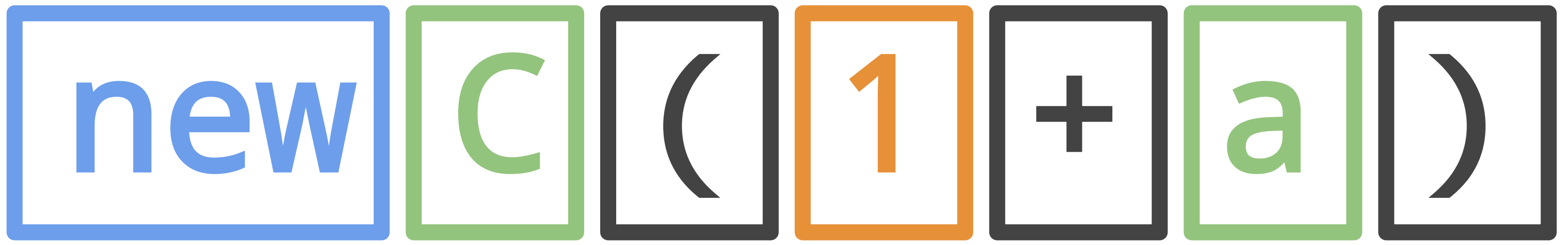
A Standard For Structured JS Representations

Michael Ficarra

A JavaScript Program

```
new C(1 + a)
```

Typical Tokenisation



▨ PUNCTUATOR

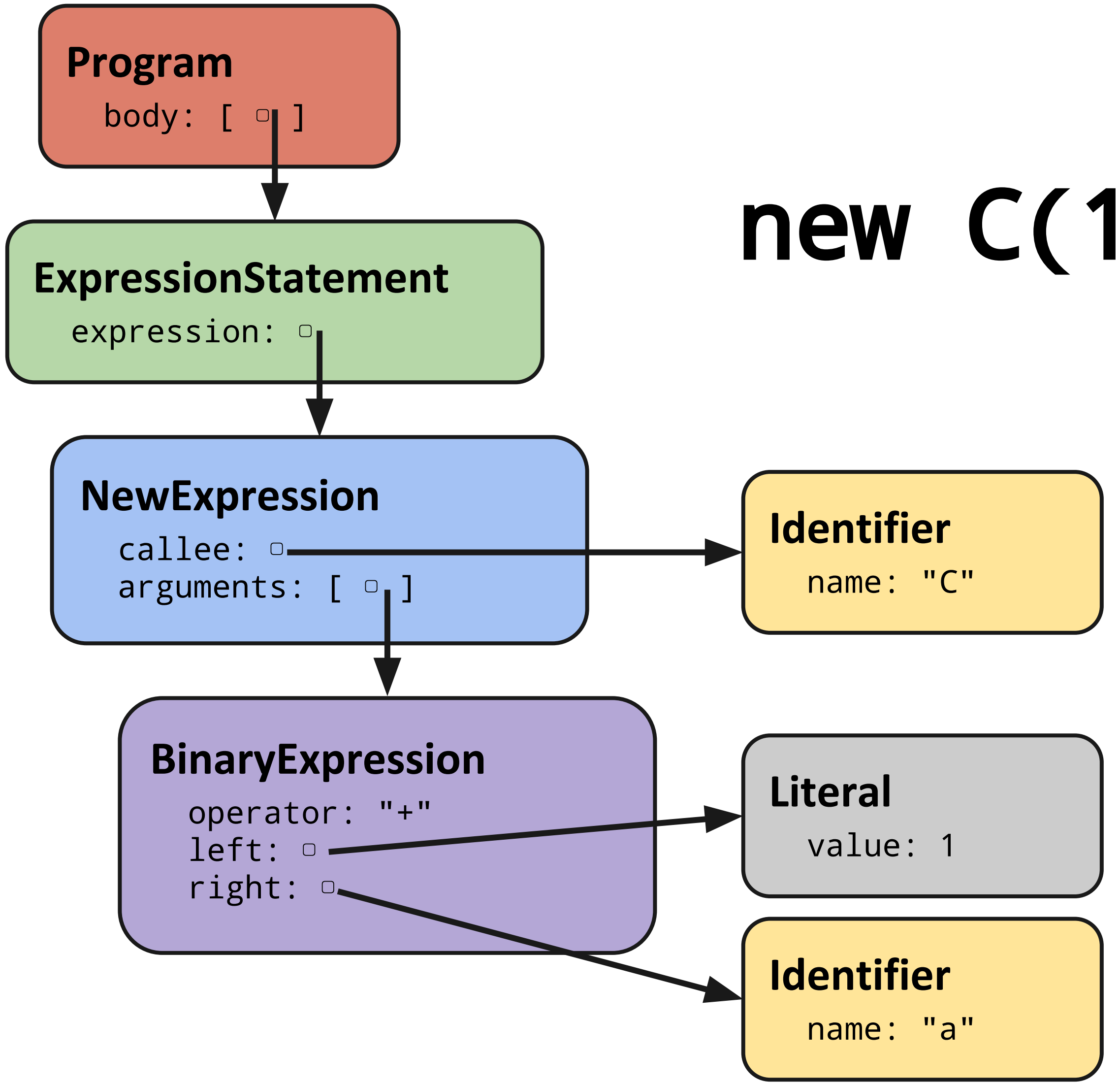
▨ KEYWORD

▨ IDENTIFIER

▨ NUMERIC

PARSER
MAGIC

Structured Representation (AST)



```
new C(1 + a)
```

← Module scoping and linking Zaphod: A Browser Language Lab for JS →

An API for parsing JavaScript

Posted on August 25, 2010 | 13 Comments

In new builds of the SpiderMonkey shell we're introducing an experimental API for parsing JavaScript source code, which landed this week. For now, you have to download and build SpiderMonkey from source to use it, but hopefully we'll include it in future versions of Firefox.

The parser API provides a single function:

```
Reflect.parse(src[, filename=null[, lineno=1]])
```

`Reflect.parse` takes a source string (and optionally, a filename and starting line number for source location metadata), and produces a JavaScript object representing the abstract syntax tree of the parsed source code, using the built-in parser of SpiderMonkey itself. Straightforward enough, but behind this simple entry point is a thorough API that covers the entirety of SpiderMonkey's abstract syntax. In short, *anything that SpiderMonkey can parse, you can parse, too*. Developer tools

Parser API

by 22 contributors:

Recent builds of the [standalone SpiderMonkey shell] include a reflection of the SpiderMonkey parser, made available as a JavaScript API. This makes it easier to write tools in JavaScript that manipulate JavaScript source programs, such as syntax highlighters, static analyses, translators, compilers, obfuscators, etc.

NOTE: Several projects are using this specification. Please do not make changes to it without consulting with the authors of Esprima, Escodegen, and Acorn.

Example:

```
> var expr = Reflect.parse("obj.foo + 42").body[0].expression
> expr.left.property
({loc:null, type:"Identifier", name:"foo"})
> expr.right
({loc:{source:null, start:{line:1, column:10}, end:{line:1, column:12}}, type:"Literal", value:42})
```

It is also available since Firefox 7; it can be imported into the global object via:

```
Components.utils.import("resource://gre/modules/reflect.jsm")
```

Structured Representation (AST)

```
{ type: "Program"
, body: [
  { type: "ExpressionStatement"
  , expression:
    { type: "NewExpression"
    , callee: {type: "Identifier", name: "C"}
    , arguments: [
      { type: "BinaryExpression"
      , operator: "+"
      , left: {type: "Literal", value: 1}
      , right: {type: "Identifier", name: "a"}
      }
    ]}
  }
]}
```

`Reflect.parse(src[, options])`

Coerces **`src`** to a string and parses the result as a JavaScript program. By default, the parsing returns a Program object (see below) representing the parsed abstract syntax tree (AST).

Additional options may be provided via the **`options`** object, which can include any of the following properties:

`loc`	Boolean	Default: `true`
When **`loc`** is `true`, the parser includes source location information in the returned AST nodes.		
`source`	String	Default: `null`
A description of the input source; typically a filename, path, or URL. This string is not meaningful to the parsing process, but is produced as part of the source location information in the returned AST nodes.		
`line`	Number	Default: `1`
The initial line number to use for source location information.		
`builder`	Builder	Default: `null`
A builder object, which can be used to produce AST nodes in custom data formats. The expected callback methods are described under Builder Objects.		

If parsing fails due to a syntax error, an instance of `SyntaxError` is thrown. The syntax error object thrown by `Reflect.parse()` has the same `message` property as the syntax error that would be thrown by `eval(src)`. The `lineNumber` and `fileName` properties of the syntax error object indicate the source location of the syntax error.

```
interface Node {
    type: string;
    loc: SourceLocation | null;
}
```

The `type` field is a string representing the AST variant type. Each subtype of Node is documented below with the specific string of its `type` field. You can use this field to determine which interface a node implements.

The `loc` field represents the source location information of the node. If the parser produced no information about the node's source location, the field is `null`; otherwise it is an object consisting of a start position (the position of the first character of the parsed source region) and an end position (the position of the first character *after* the parsed source region):

```
interface SourceLocation {
    source: string | null;
    start: Position;
    end: Position;
}
```

Each `Position` object consists of a `line` number (1-indexed) and a `column` number (0-indexed):

```
interface Position {
    line: uint32 >= 1;
    column: uint32 >= 0;
}
```

Programs

```
interface Program <: Node {
    type: "Program";
    body: [ Statement ];
}
```

A complete program source tree.

Functions

```
interface Function <: Node {
    id: Identifier | null;
    params: [ Pattern ];
    defaults: [ Expression ];
    rest: Identifier | null;
    body: BlockStatement | Expression;
    generator: boolean;
    expression: boolean;
}
```

A function declaration or expression. The `body` of the function may be a block statement, or in the case of an expression closure, an expression.

***Note:** Expression closures are SpiderMonkey-specific.*

```
interface Statement <: Node { }
```

Any statement.

```
interface EmptyStatement <: Statement {
    type: "EmptyStatement";
}
```

An empty statement, i.e., a solitary semicolon.

```
interface BlockStatement <: Statement {
    type: "BlockStatement";
    body: [ Statement ];
}
```

A block statement, i.e., a sequence of statements surrounded by braces.

```
interface ExpressionStatement <: Statement {
    type: "ExpressionStatement";
    expression: Expression;
}
```

An expression statement, i.e., a statement consisting of a single expression.

Properties of a Good AST Format

1. each node tagged with its type(s)
2. nodes have no state or knowledge of context
3. disallows construction of invalid program
4. similar syntactic productions are meaningfully grouped

```
interface BinaryExpression <: Expression {
    type: "BinaryExpression";
    operator: BinaryOperator;
    left: Expression;
    right: Expression;
}
```

A binary operator expression.

```
interface AssignmentExpression <: Expression {
    type: "AssignmentExpression";
    operator: AssignmentOperator;
    left: Expression;
    right: Expression;
}
```

An assignment operator expression.

```
interface LogicalExpression <: Expression {
    type: "LogicalExpression";
    operator: LogicalOperator;
    left: Expression;
    right: Expression;
}
```

A logical operator expression.

```
interface UnaryExpression <: Expression {
    type: "UnaryExpression";
    operator: UnaryOperator;
    prefix: boolean;
    argument: Expression;
}
```

A unary operator expression.

```
interface UpdateExpression <: Expression {
    type: "UpdateExpression";
    operator: UpdateOperator;
    argument: Expression;
    prefix: boolean;
}
```

An update (increment or decrement) operator expression.

```
interface Identifier <: Node, Expression, Pattern {
    type: "Identifier";
    name: string;
}
```

An identifier. Note that an identifier may be an expression or a destructuring pattern.

```
interface Literal <: Node, Expression {
    type: "Literal";
    value: string | boolean | null | number | RegExp;
}
```

A literal token. Note that a literal can be an expression.

```
interface MemberExpression <: Expression {
    type: "MemberExpression";
    object: Expression;
    property: Identifier | Expression;
    computed: boolean;
}
```

A member expression. If `computed === true`, the node corresponds to a computed `e1[e2]` expression and property is an `Expression`. If `computed === false`, the node corresponds to a static `e1.x` expression and property is an `Identifier`.

```
interface ArrayExpression <: Expression {
    type: "ArrayExpression";
    elements: [ Expression | null ];
}
```

An array expression.

```
interface ObjectExpression <: Expression {
    type: "ObjectExpression";
    properties: [ { key: Literal | Identifier,
                    value: Expression,
                    kind: "init" | "get" | "set" } ];
}
```

An object expression. A literal property in an object expression can have either a string or number as its `value`. Ordinary property initializers have a `kind` value `"init"`; getters and setters have the `kind` values `"get"` and `"set"`, respectively.

Overly Permissive: Structures

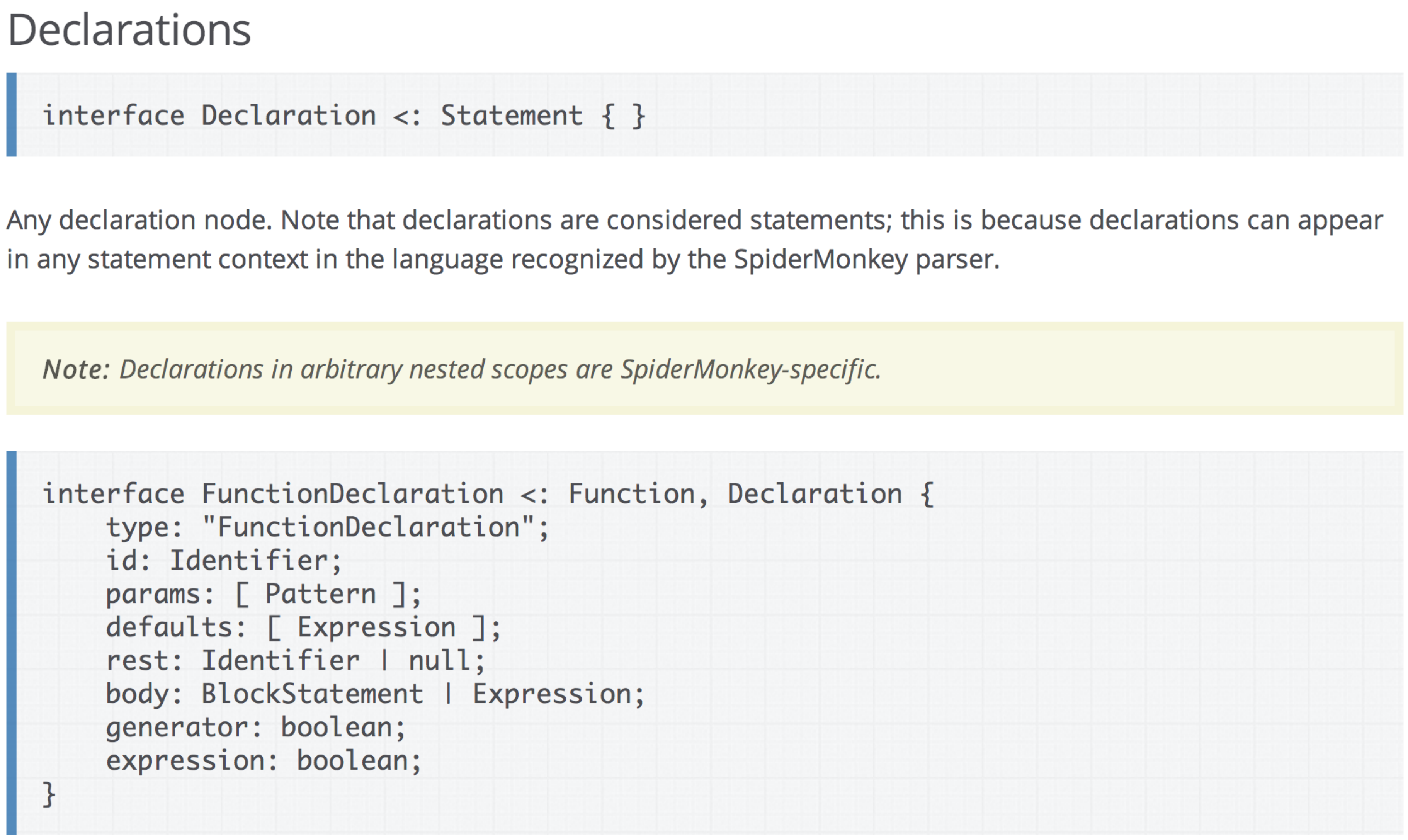
```
{
  type: "IfStatement",
  test: (...),
  consequent: {
    type: "IfStatement",
    test: (...),
    consequent: (...),
    alternate: null
  },
  alternate: (...)
}
```

```
{
    type: "TryStatement",
    block: (...),
    handler: null,
    guardedHandlers: [],
    finalizer: null
}
```

```
if(test)
  if(test) a();
else b();
```

```
try { a() }
```

Overly Permissive: Decl. Position



Declarations

```
interface Declaration <: Statement { }
```

Any declaration node. Note that declarations are considered statements; this is because declarations can appear in any statement context in the language recognized by the SpiderMonkey parser.

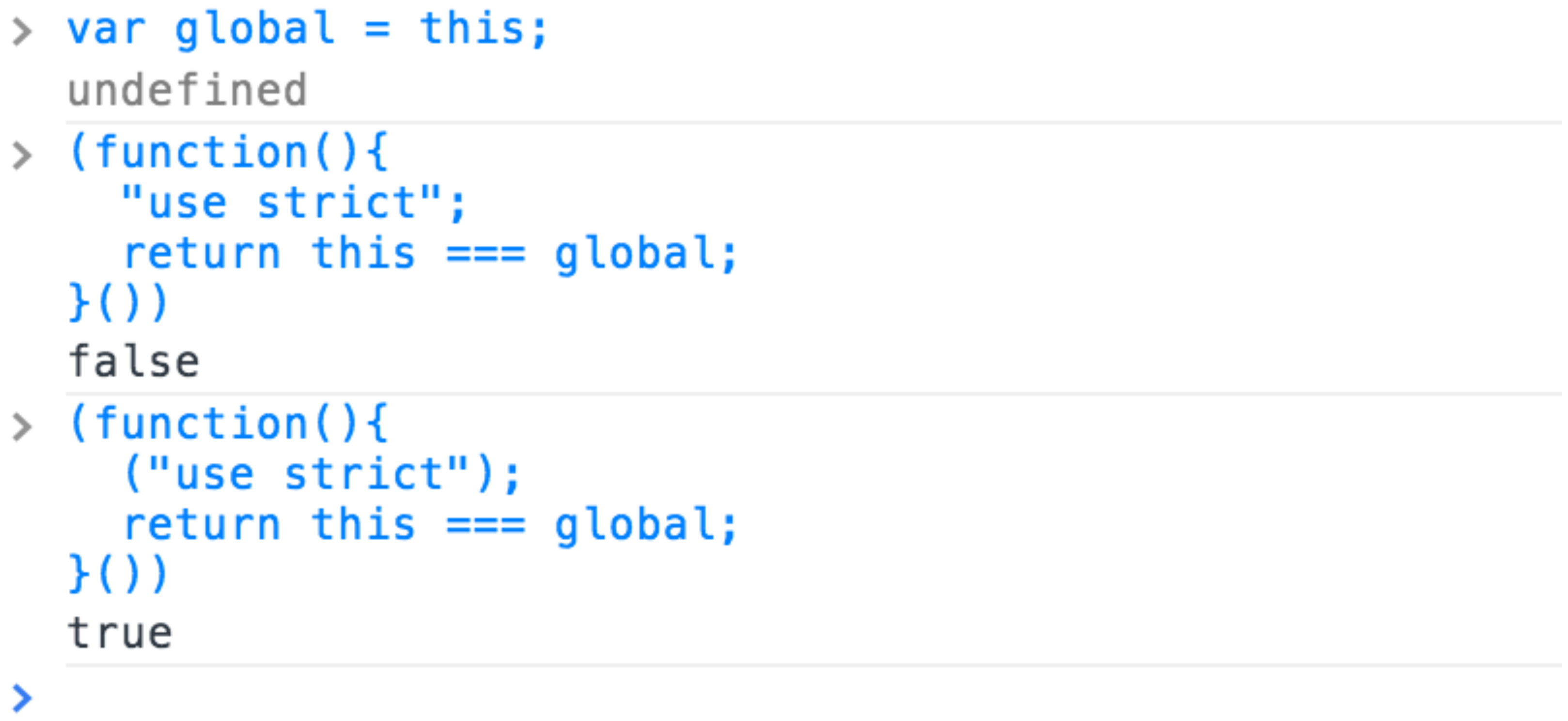
> ***Note:** Declarations in arbitrary nested scopes are SpiderMonkey-specific.*

```
interface FunctionDeclaration <: Function, Declaration {
    type: "FunctionDeclaration";
    id: Identifier;
    params: [ Pattern ];
    defaults: [ Expression ];
    rest: Identifier | null;
    body: BlockStatement | Expression;
    generator: boolean;
    expression: boolean;
}
```

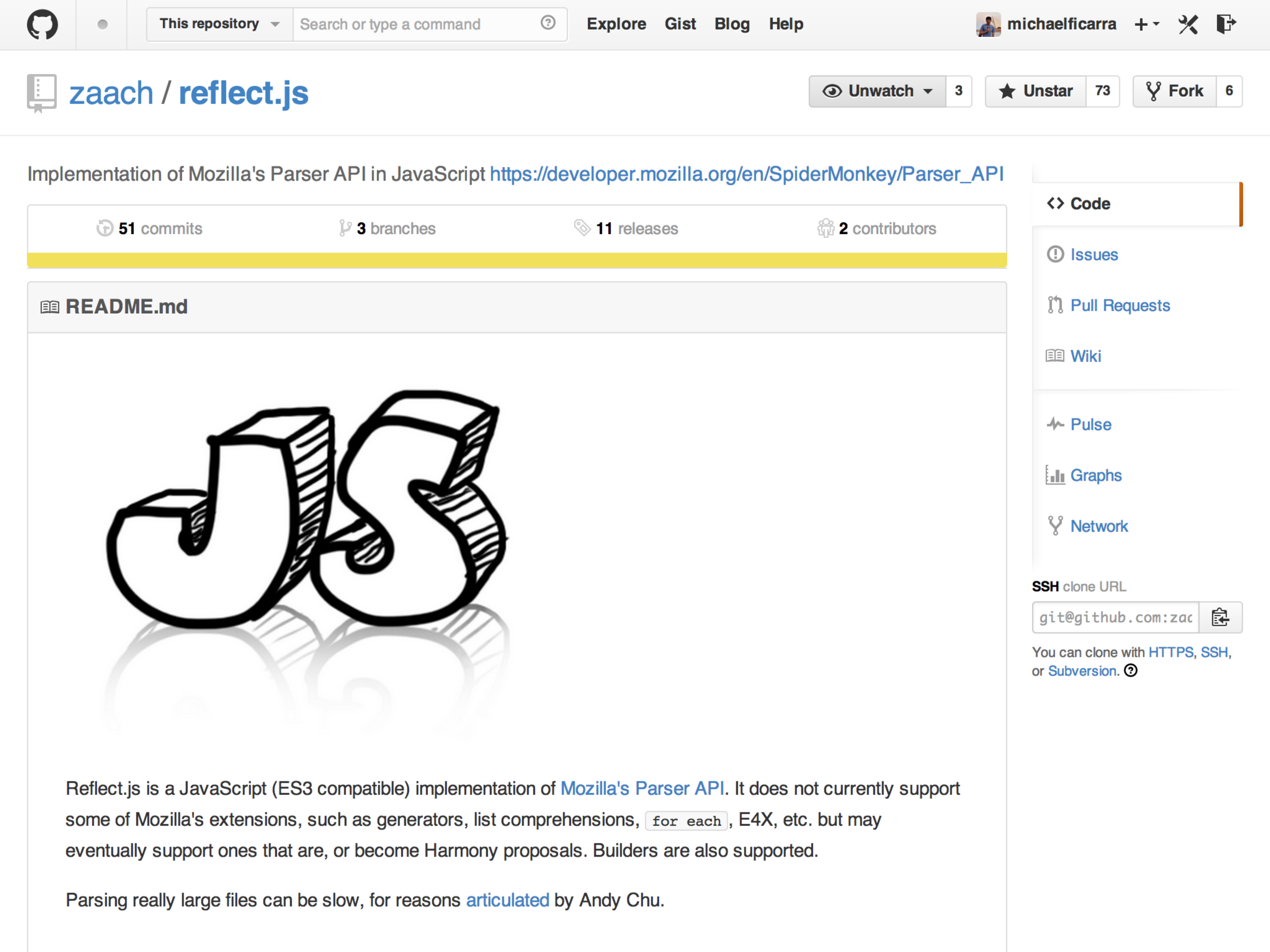
Overly Permissive: List Properties

```
0, // needs to sequence at least 2 expressions

var // needs at least one declarator

switch(0) { } // needs at least one case/default

// cannot contain more than one default
switch(0) {
   default: 0
   default: 0
}
```

No DirectiveStatement Node (yet)



```
> var global = this;
  undefined
> (function(){
    "use strict";
    return this === global;
  }())
  false
> (function(){
    ("use strict");
    return this === global;
  }())
  true
>
```



zaach / reflect.js

Implementation of Mozilla's Parser API in JavaScript https://developer.mozilla.org/en/SpiderMonkey/Parser_API

51 commits **3** branches **11** releases **2** contributors

README.md

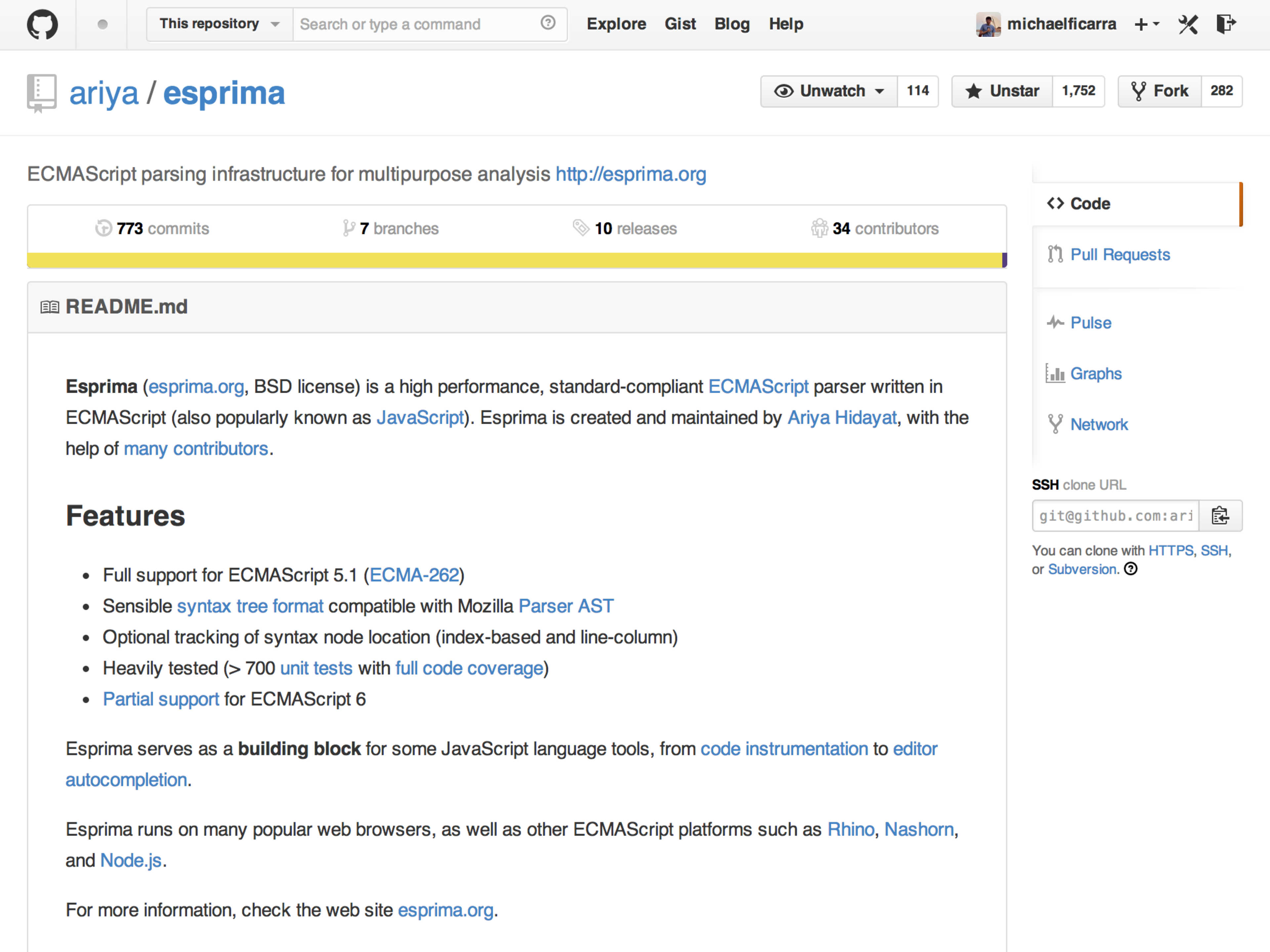
Reflect.js is a JavaScript (ES3 compatible) implementation of Mozilla's Parser API. It does not currently support some of Mozilla's extensions, such as generators, list comprehensions, `for each`, E4X, etc. but may eventually support ones that are, or become Harmony proposals. Builders are also supported.

Parsing really large files can be slow, for reasons articulated by Andy Chu.

SSH clone URL

`git@github.com:zac`

You can clone with HTTPS, SSH, or Subversion.



ariya / esprima

ECMAScript parsing infrastructure for multipurpose analysis http://esprima.org

README.md

Esprima (esprima.org, BSD license) is a high performance, standard-compliant ECMAScript parser written in ECMAScript (also popularly known as JavaScript). Esprima is created and maintained by Ariya Hidayat, with the help of many contributors.

Features

- Full support for ECMAScript 5.1 (ECMA-262)
- Sensible syntax tree format compatible with Mozilla Parser AST
- Optional tracking of syntax node location (index-based and line-column)
- Heavily tested (> 700 unit tests with full code coverage)
- Partial support for ECMAScript 6

Esprima serves as a **building block** for some JavaScript language tools, from code instrumentation to editor autocompletion.

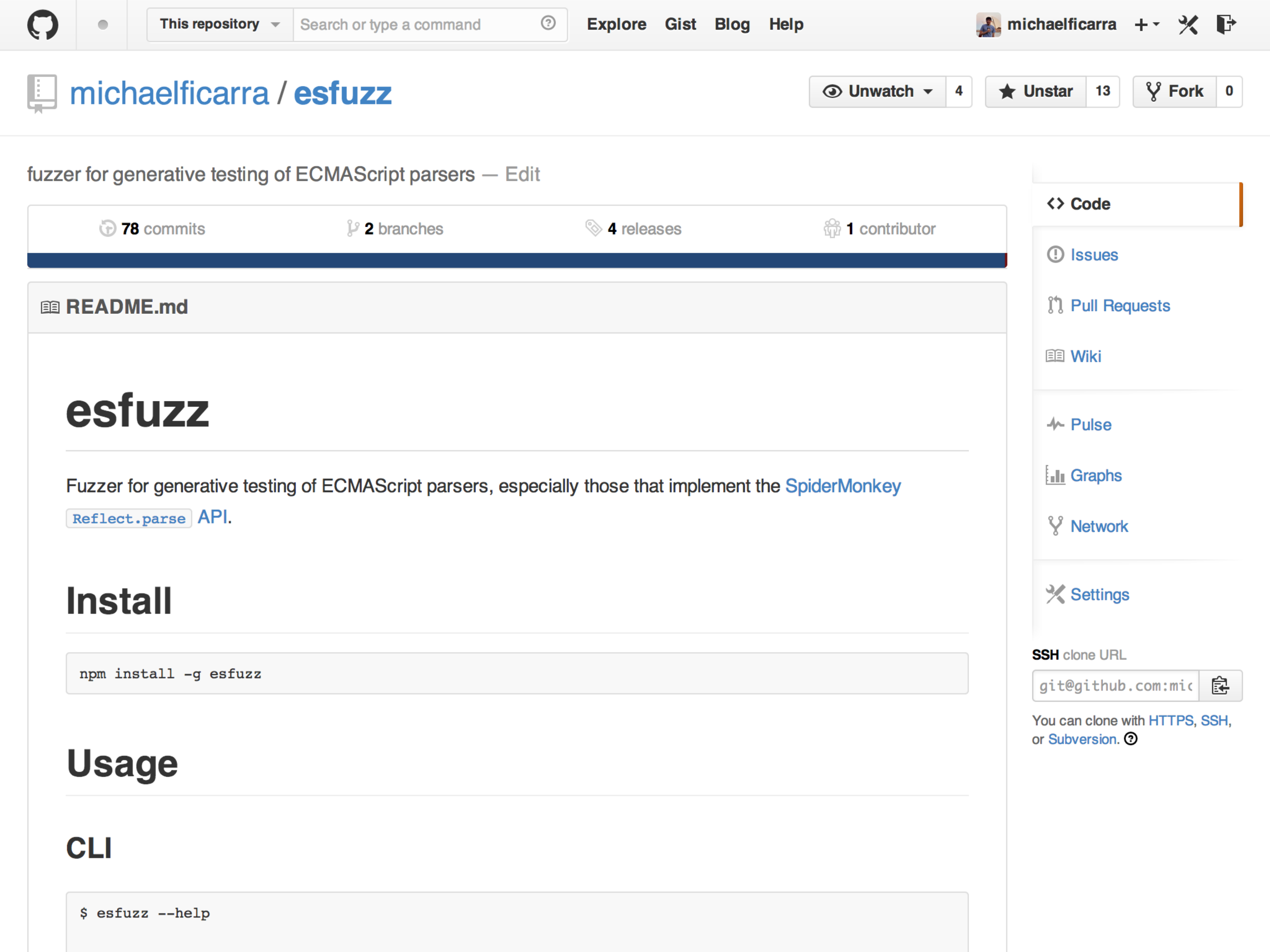
Esprima runs on many popular web browsers, as well as other ECMAScript platforms such as Rhino, Nashorn, and Node.js.

For more information, check the web site esprima.org.

SSH clone URL

`git@github.com:ari`

You can clone with HTTPS, SSH, or Subversion.



michaelficarra / esfuzz

fuzzer for generative testing of ECMAScript parsers — Edit

78 commits | 2 branches | 4 releases | 1 contributor

README.md

esfuzz

Fuzzer for generative testing of ECMAScript parsers, especially those that implement the SpiderMonkey `Reflect.parse` API.

Install

```
npm install -g esfuzz
```

Usage

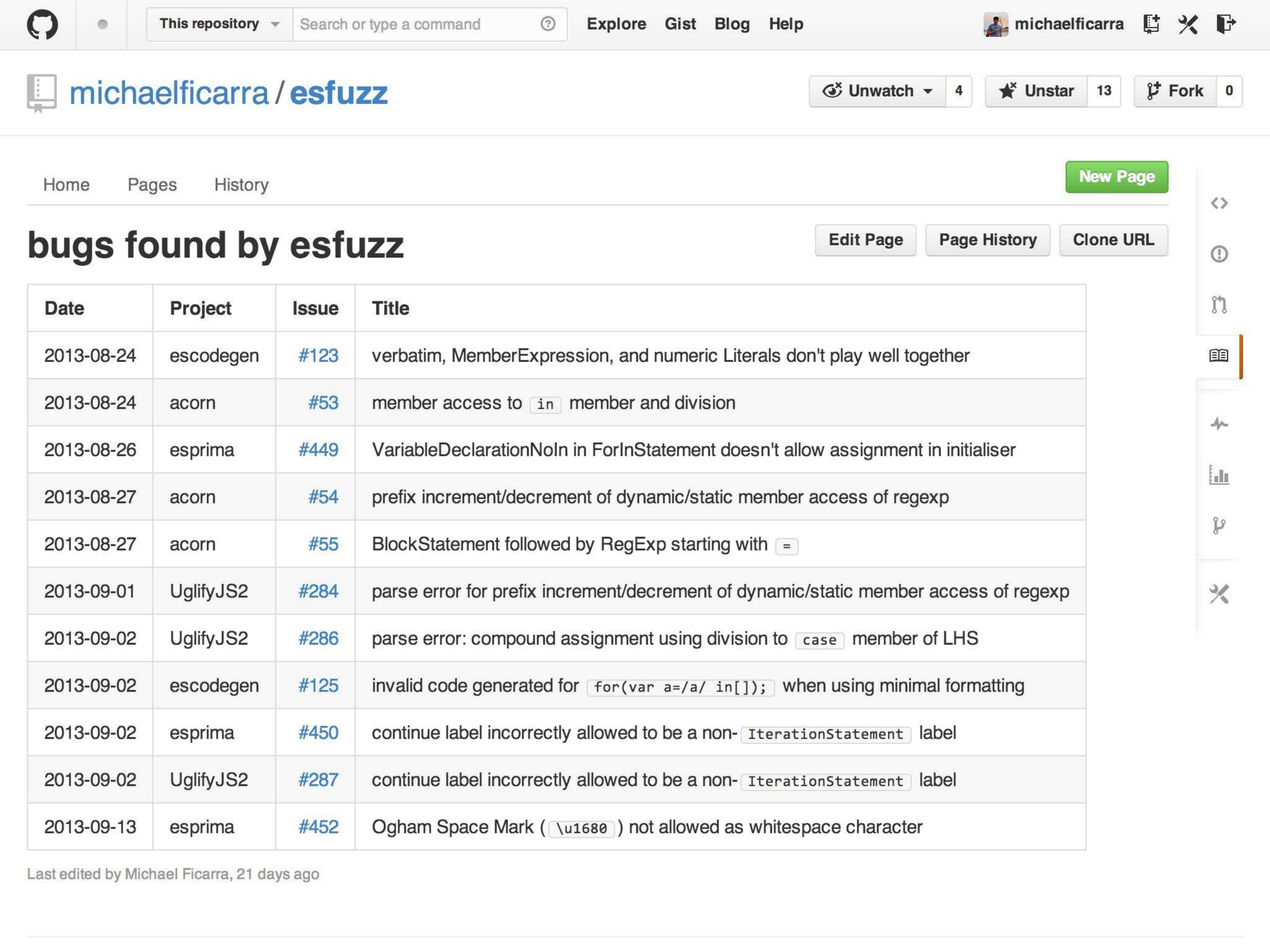
CLI

```
$ esfuzz --help
```

SSH clone URL

`git@github.com:mic`

You can clone with HTTPS, SSH, or Subversion.

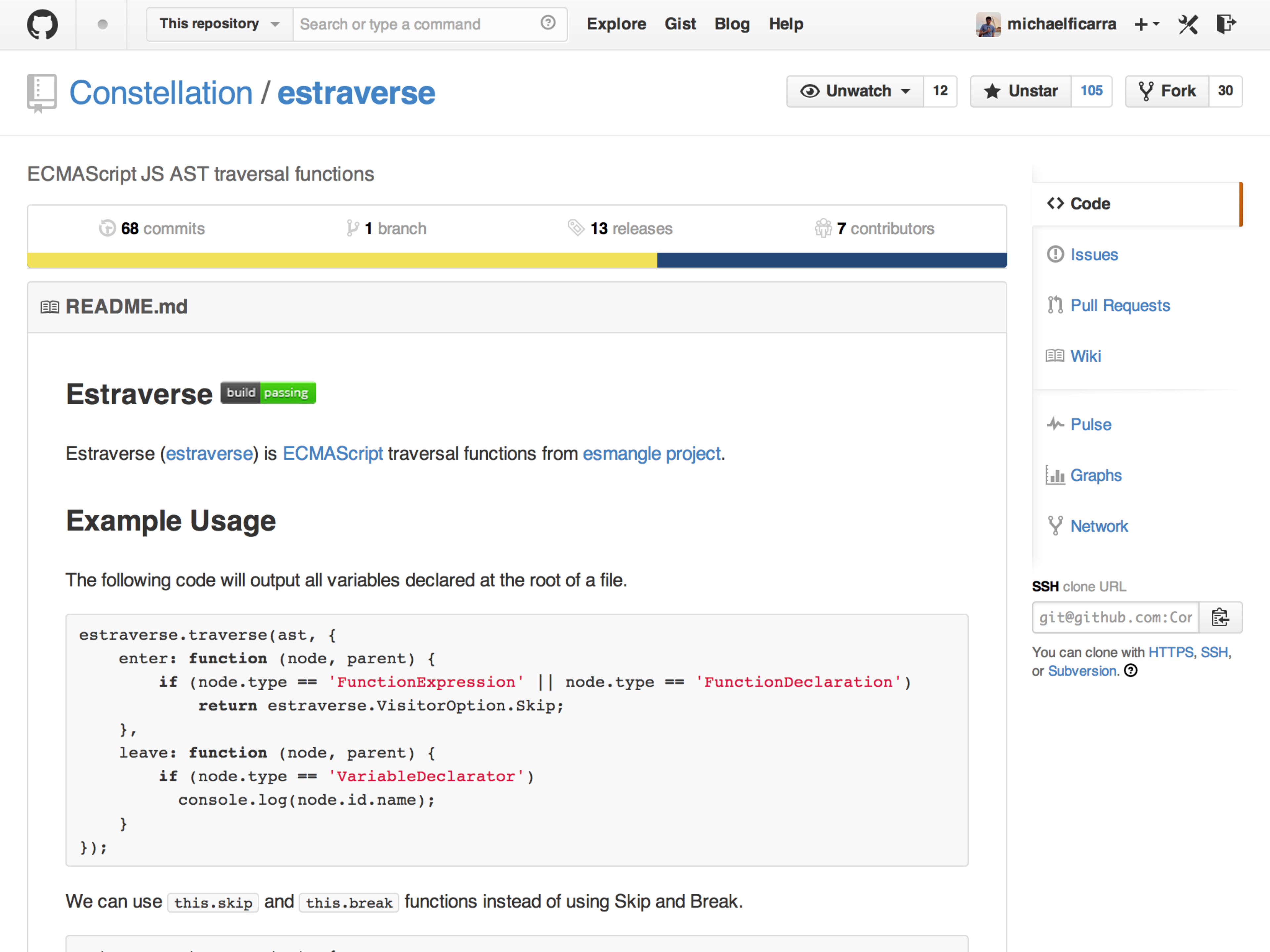


bugs found by esfuzz

Date	Project	Issue	Title
2013-08-24	escodegen	#123	verbatim, MemberExpression, and numeric Literals don't play well together
2013-08-24	acorn	#53	member access to `in` member and division
2013-08-26	esprima	#449	VariableDeclarationNoIn in ForInStatement doesn't allow assignment in initialiser
2013-08-27	acorn	#54	prefix increment/decrement of dynamic/static member access of regexp
2013-08-27	acorn	#55	BlockStatement followed by RegExp starting with `=`
2013-09-01	UglifyJS2	#284	parse error for prefix increment/decrement of dynamic/static member access of regexp
2013-09-02	UglifyJS2	#286	parse error: compound assignment using division to `case` member of LHS
2013-09-02	escodegen	#125	invalid code generated for `for(var a=/a/ in[]);` when using minimal formatting
2013-09-02	esprima	#450	continue label incorrectly allowed to be a non-`IterationStatement` label
2013-09-02	UglifyJS2	#287	continue label incorrectly allowed to be a non-`IterationStatement` label
2013-09-13	esprima	#452	Ogham Space Mark (`\u1680`) not allowed as whitespace character

Last edited by Michael Ficarra, 21 days ago

Basic Tooling



This repository | Search or type a command | Explore | Gist | Blog | Help | michaelficarra

Constellation / estraverse

Unwatch 12 | Unstar 105 | Fork 30

ECMAScript JS AST traversal functions

68 commits | 1 branch | 13 releases | 7 contributors

Code | Issues | Pull Requests | Wiki | Pulse | Graphs | Network

SSH clone URL

git@github.com:Cor

You can clone with HTTPS, SSH, or Subversion.

README.md

Estraverse build passing

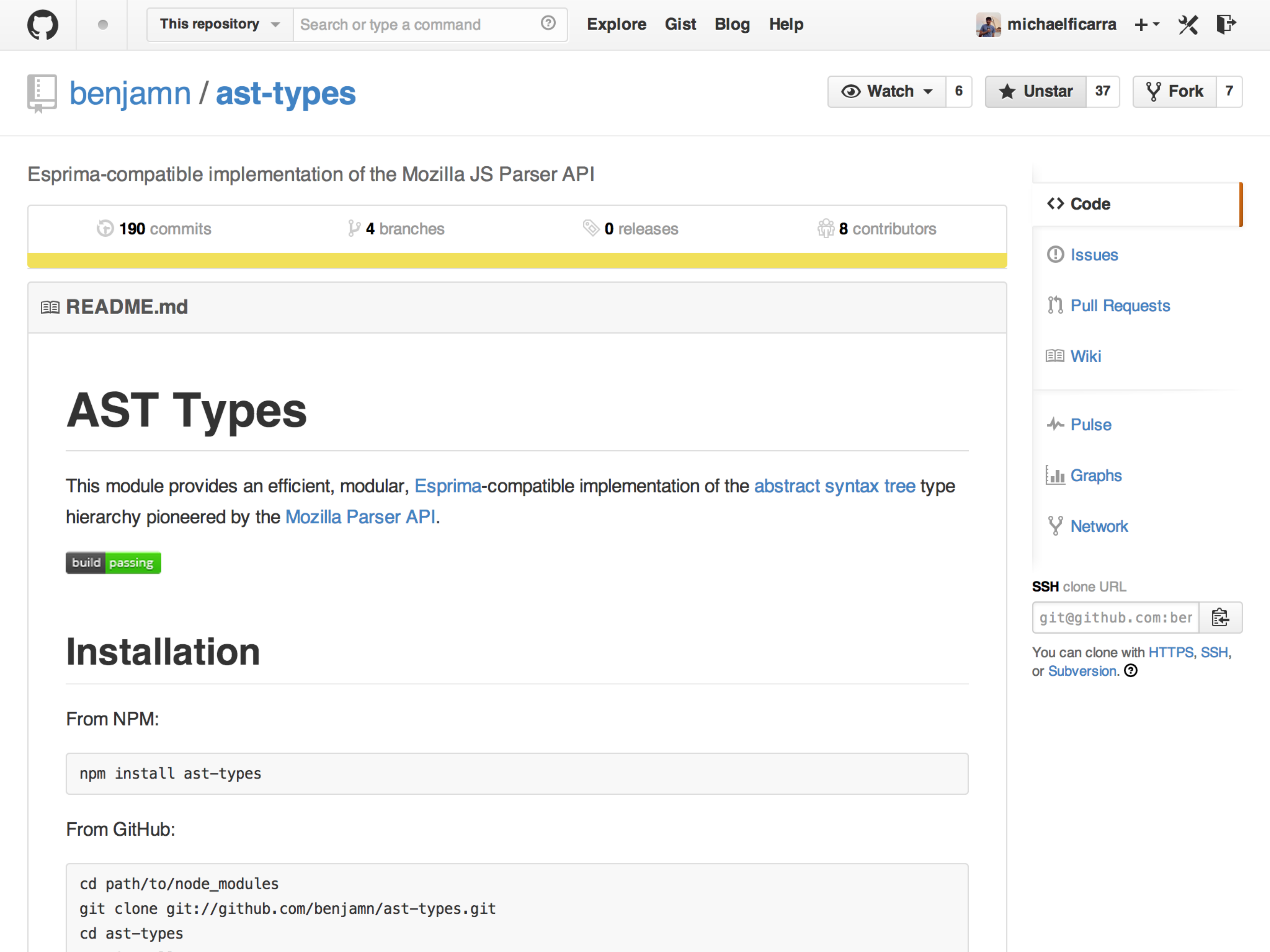
Estraverse (estraverse) is ECMAScript traversal functions from esmangle project.

Example Usage

The following code will output all variables declared at the root of a file.

```
estraverse.traverse(ast, {
    enter: function (node, parent) {
        if (node.type == 'FunctionExpression' || node.type == 'FunctionDeclaration')
            return estraverse.VisitorOption.Skip;
    },
    leave: function (node, parent) {
        if (node.type == 'VariableDeclarator')
          console.log(node.id.name);
    }
});
```

We can use `this.skip` and `this.break` functions instead of using Skip and Break.



benjamn / ast-types

Esprima-compatible implementation of the Mozilla JS Parser API

190 commits 4 branches 0 releases 8 contributors

README.md

AST Types

This module provides an efficient, modular, Esprima-compatible implementation of the abstract syntax tree type hierarchy pioneered by the Mozilla Parser API.

Installation

From NPM:

```
npm install ast-types
```

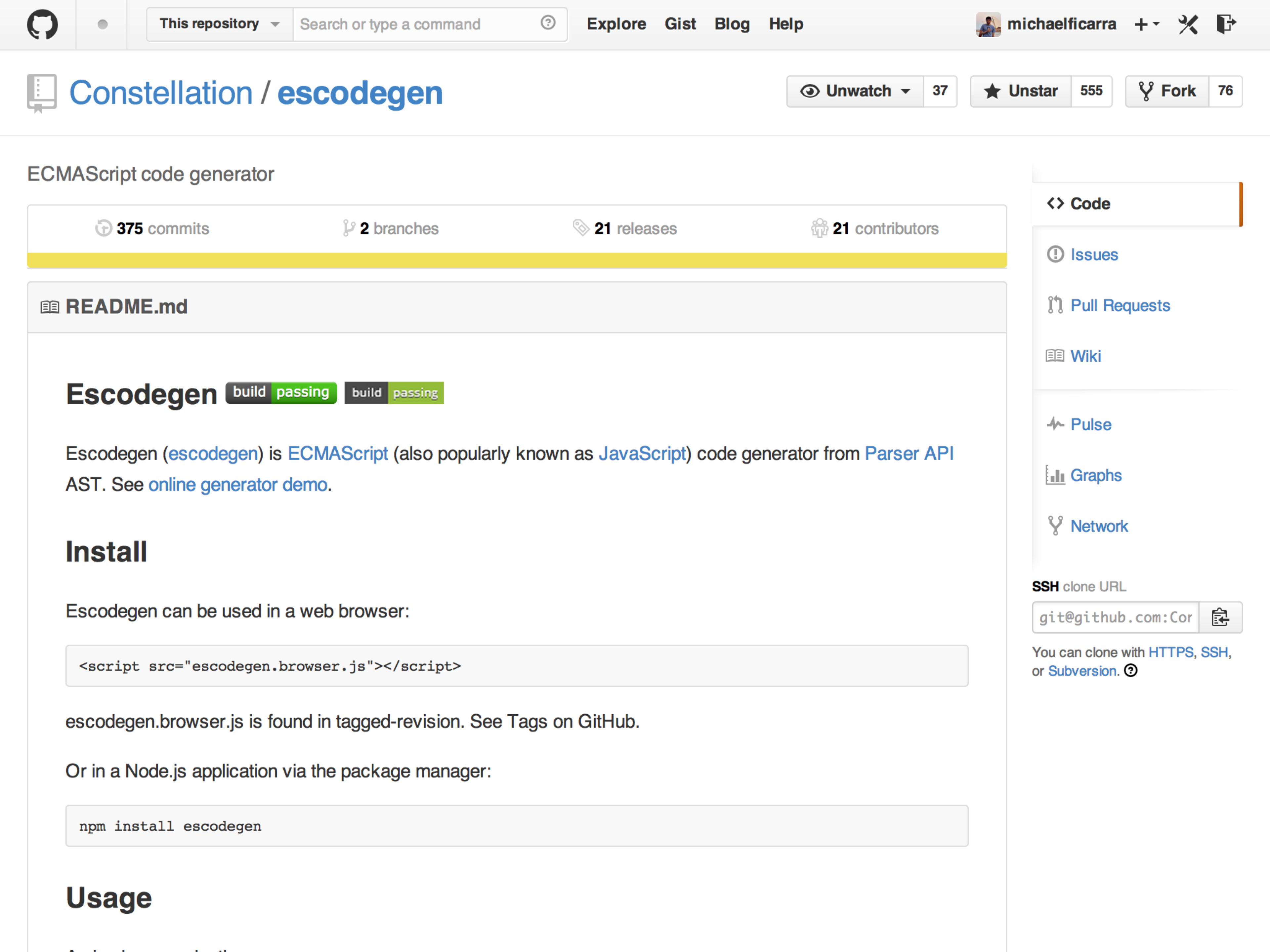
From GitHub:

```
cd path/to/node_modules
git clone git://github.com/benjamn/ast-types.git
cd ast-types
```

SSH clone URL

git@github.com:ber

You can clone with HTTPS, SSH, or Subversion.



Constellation / escodegen

ECMAScript code generator

375 commits 2 branches 21 releases 21 contributors

SSH clone URL

git@github.com:Cor

You can clone with HTTPS, SSH, or Subversion.

README.md

Escodegen

Escodegen (escodegen) is ECMAScript (also popularly known as JavaScript) code generator from Parser API AST. See online generator demo.

Install

Escodegen can be used in a web browser:

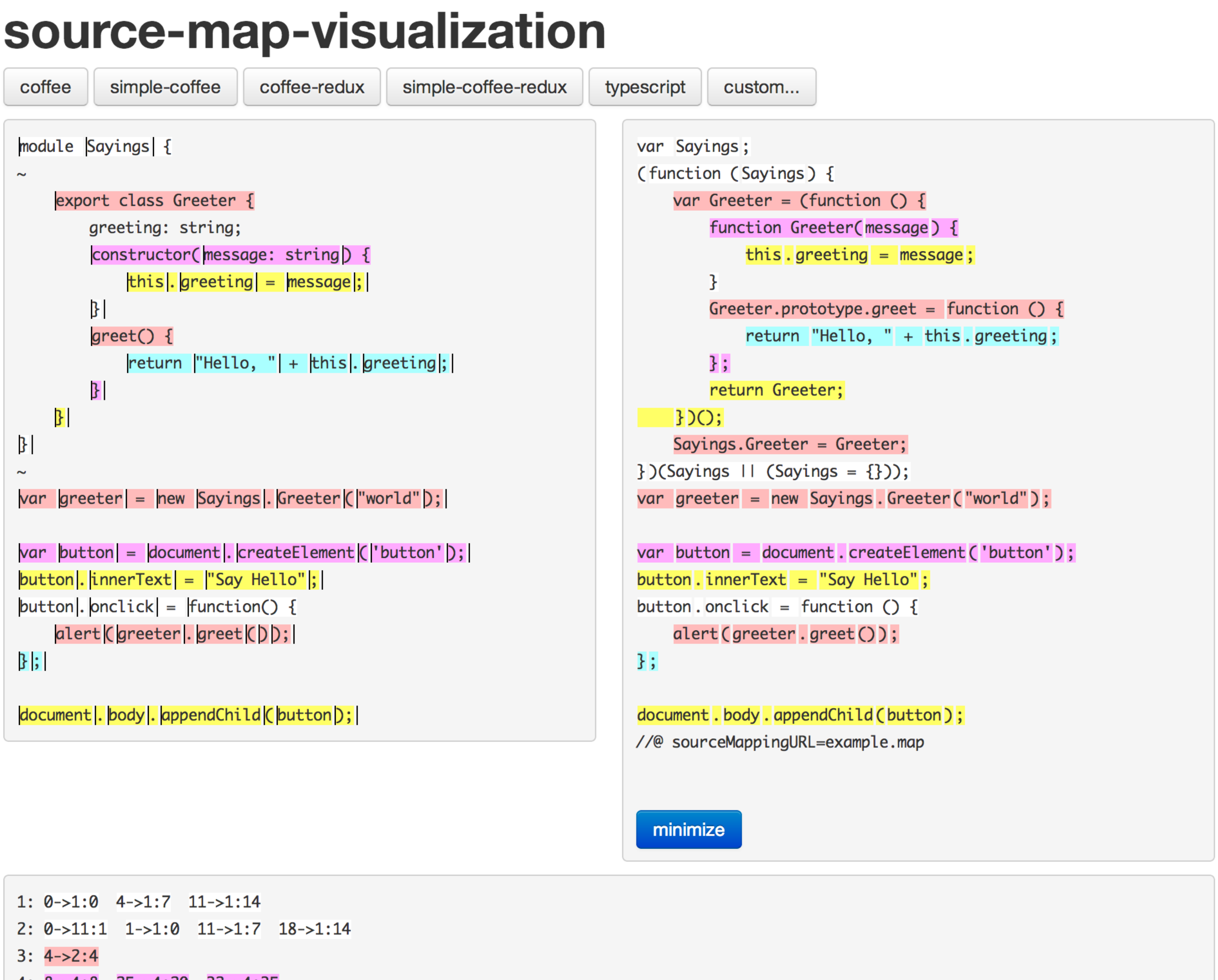
```
<script src="escodegen.browser.js"></script>
```

escodegen.browser.js is found in tagged-revision. See Tags on GitHub.

Or in a Node.js application via the package manager:

```
npm install escodegen
```

Usage

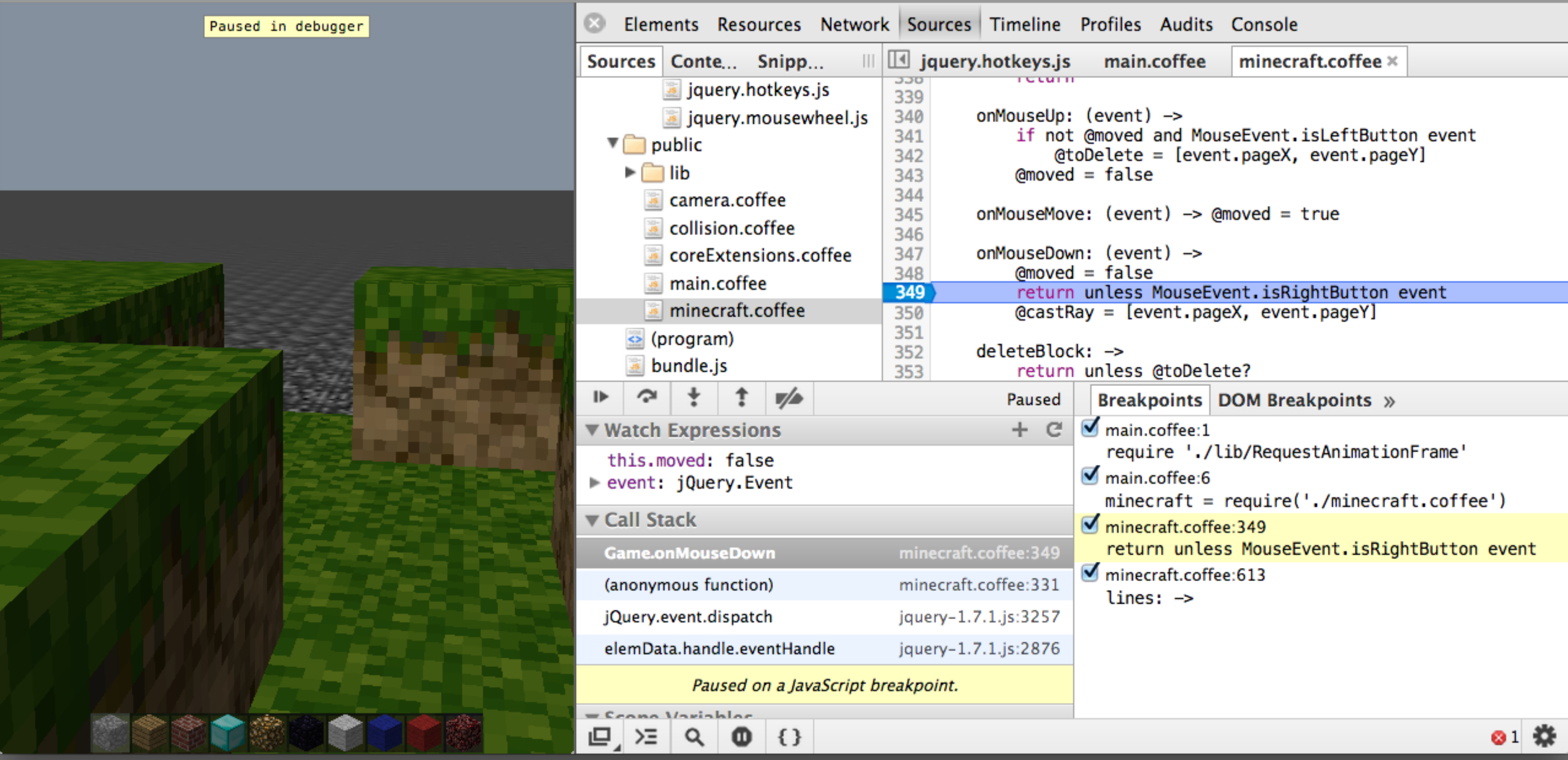


source-map-visualization

```
module Sayings {
~
    export class Greeter {
        greeting: string;
        constructor(message: string) {
            this.greeting = message;
        }
        greet() {
            return "Hello, " + this.greeting;
        }
    }
}
~
var greeter = new Sayings.Greeter("world");

var button = document.createElement('button');
button.innerText = "Say Hello";
button.onclick = function() {
    alert(greeter.greet());
};

document.body.appendChild(button);
```

```
var Sayings;
(function (Sayings) {
    var Greeter = (function () {
        function Greeter(message) {
            this.greeting = message;
        }
        Greeter.prototype.greet = function () {
            return "Hello, " + this.greeting;
        };
        return Greeter;
    })();
    Sayings.Greeter = Greeter;
})(Sayings || (Sayings = {}));
var greeter = new Sayings.Greeter("world");

var button = document.createElement('button');
button.innerText = "Say Hello";
button.onclick = function () {
    alert(greeter.greet());
};

document.body.appendChild(button);
//@ sourceMappingURL=example.map
```

```
1: 0->1:0  4->1:7  11->1:14
2: 0->11:1  1->1:0  11->1:7  18->1:14
3: 4->2:4
```



```
    onMouseUp: (event) ->
        if not @moved and MouseEvent.isLeftButton event
            @toDelete = [event.pageX, event.pageY]
        @moved = false

    onMouseMove: (event) -> @moved = true

    onMouseDown: (event) ->
        @moved = false
        return unless MouseEvent.isRightButton event
        @castRay = [event.pageX, event.pageY]

    deleteBlock: ->
        return unless @toDelete?
```

Watch Expressions

```
this.moved: false
▶ event: jQuery.Event
```

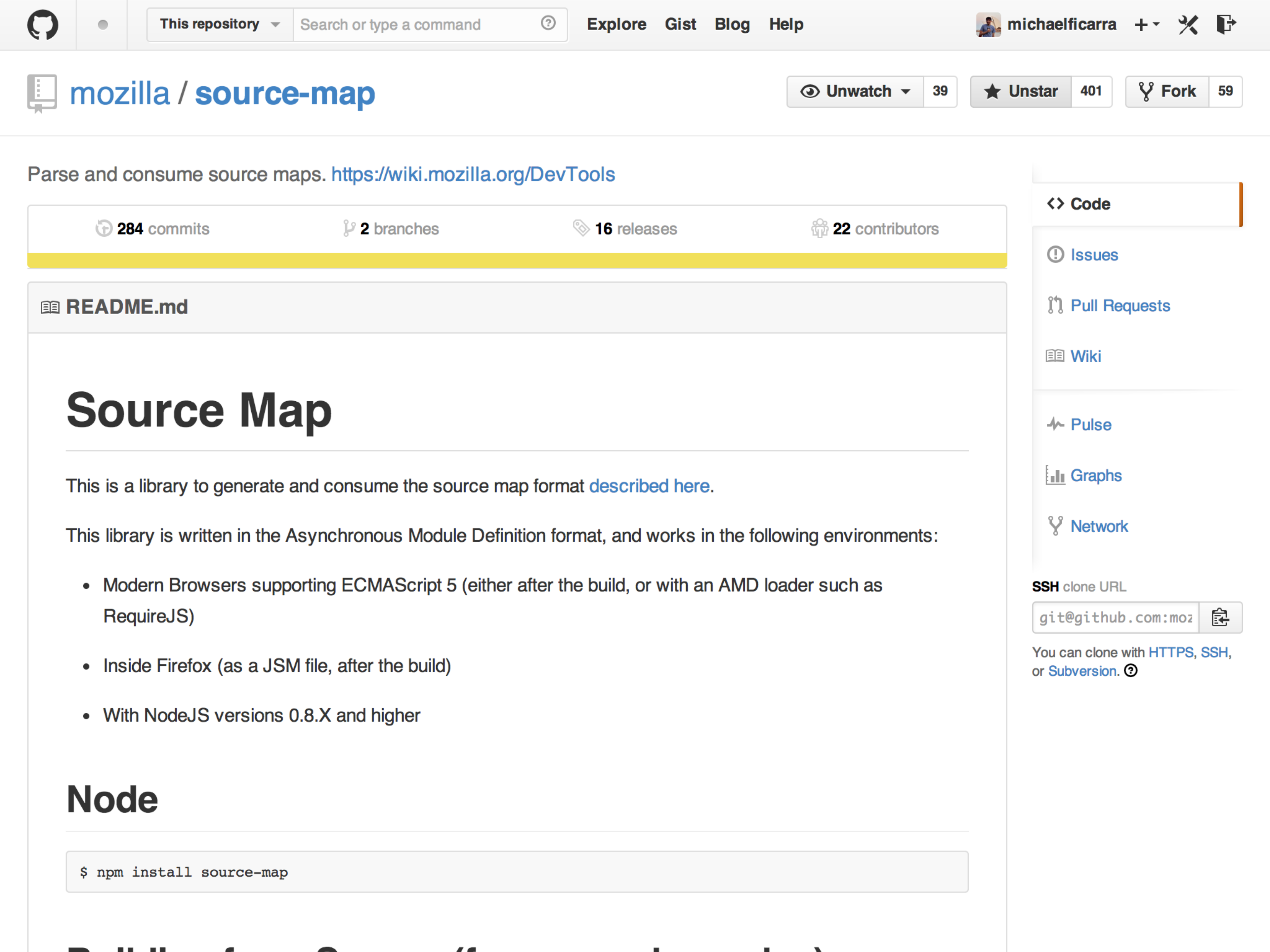
Call Stack

```
Game.onMouseDown                 minecraft.coffee:349
(anonymous function)             minecraft.coffee:331
jQuery.event.dispatch            jquery-1.7.1.js:3257
elemData.handle.eventHandle      jquery-1.7.1.js:2876
```

Paused on a JavaScript breakpoint.

Breakpoints

```
✓ main.coffee:1
  require './lib/RequestAnimationFrame'
✓ main.coffee:6
  minecraft = require('./minecraft.coffee')
✓ minecraft.coffee:349
  return unless MouseEvent.isRightButton event
✓ minecraft.coffee:613
  lines: ->
```



mozilla / source-map

Parse and consume source maps. https://wiki.mozilla.org/DevTools

284 commits 2 branches 16 releases 22 contributors

README.md

Source Map

This is a library to generate and consume the source map format described here.

This library is written in the Asynchronous Module Definition format, and works in the following environments:

- Modern Browsers supporting ECMAScript 5 (either after the build, or with an AMD loader such as RequireJS)
- Inside Firefox (as a JSM file, after the build)
- With NodeJS versions 0.8.X and higher

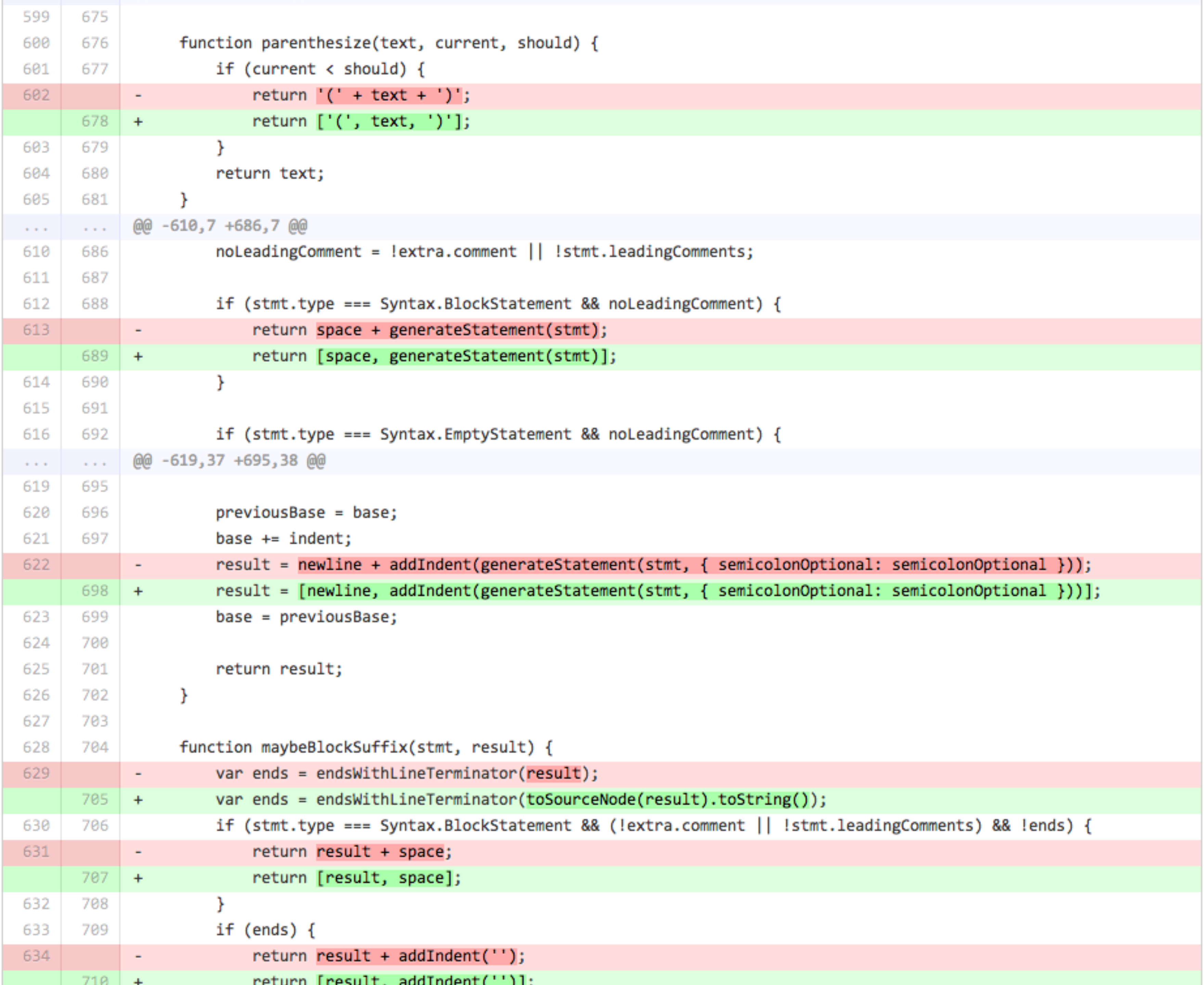
Node

```
$ npm install source-map
```

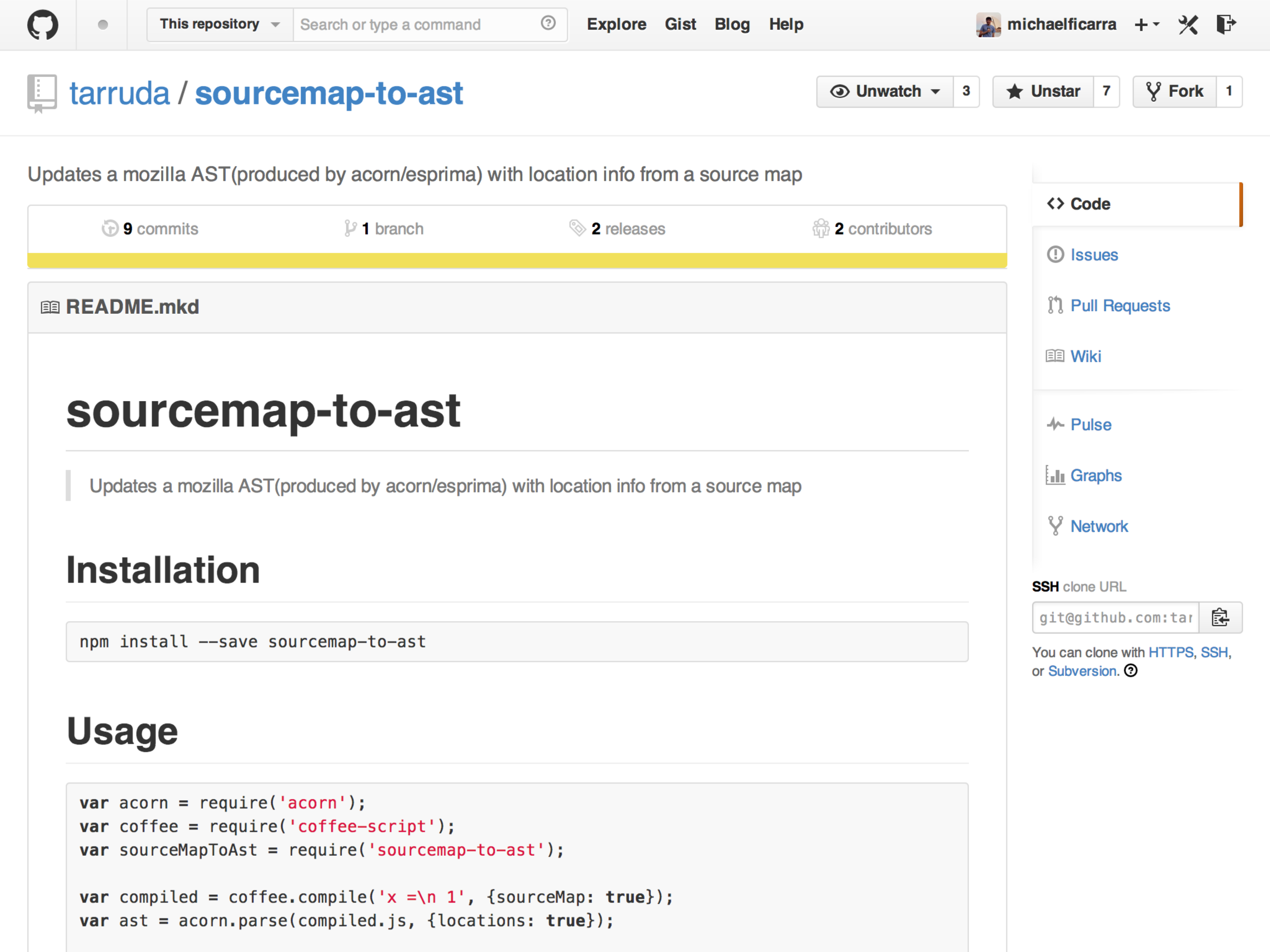
SSH clone URL

git@github.com:moz

You can clone with HTTPS, SSH, or Subversion.



```diff
 
     function parenthesize(text, current, should) {
         if (current < should) {
-            return '(' + text + ')';
+            return ['(', text, ')'];
         }
         return text;
     }
@@ -610,7 +686,7 @@
         noLeadingComment = !extra.comment || !stmt.leadingComments;
 
         if (stmt.type === Syntax.BlockStatement && noLeadingComment) {
-            return space + generateStatement(stmt);
+            return [space, generateStatement(stmt)];
         }
 
         if (stmt.type === Syntax.EmptyStatement && noLeadingComment) {
@@ -619,37 +695,38 @@
 
         previousBase = base;
         base += indent;
-        result = newline + addIndent(generateStatement(stmt, { semicolonOptional: semicolonOptional }));
+        result = [newline, addIndent(generateStatement(stmt, { semicolonOptional: semicolonOptional }))];
         base = previousBase;
 
         return result;
     }
 
     function maybeBlockSuffix(stmt, result) {
-        var ends = endsWithLineTerminator(result);
+        var ends = endsWithLineTerminator(toSourceNode(result).toString());
         if (stmt.type === Syntax.BlockStatement && (!extra.comment || !stmt.leadingComments) && !ends) {
-            return result + space;
+            return [result, space];
         }
         if (ends) {
-            return result + addIndent('');
+            return [result, addIndent('')];
```



tarruda / sourcemap-to-ast

Updates a mozilla AST(produced by acorn/esprima) with location info from a source map

README.mkd

sourcemap-to-ast

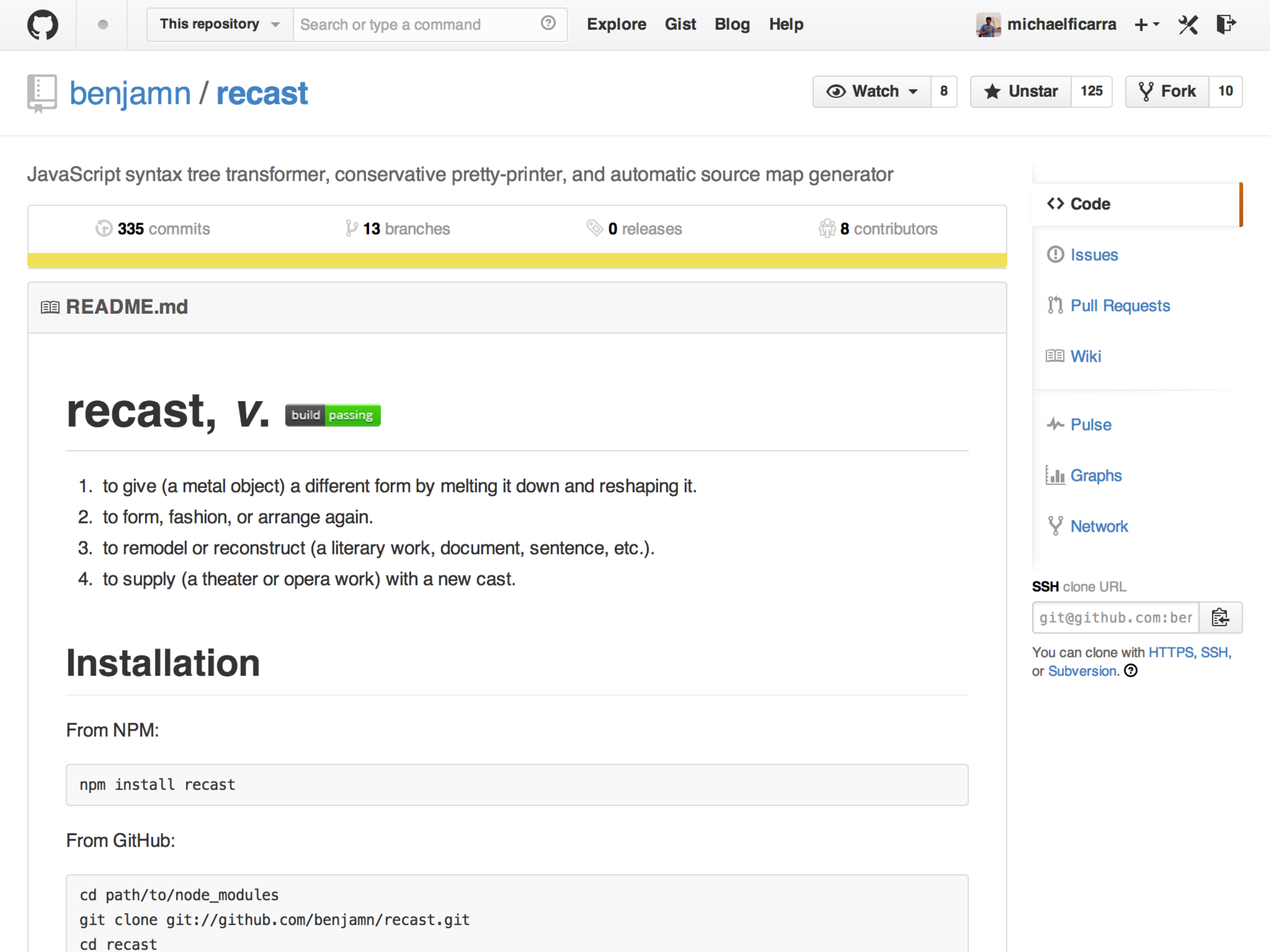
> Updates a mozilla AST(produced by acorn/esprima) with location info from a source map

Installation

```
npm install --save sourcemap-to-ast
```

Usage

```
var acorn = require('acorn');
var coffee = require('coffee-script');
var sourceMapToAst = require('sourcemap-to-ast');

var compiled = coffee.compile('x =\n 1', {sourceMap: true});
var ast = acorn.parse(compiled.js, {locations: true});
```



benjamn / recast

JavaScript syntax tree transformer, conservative pretty-printer, and automatic source map generator

recast, *v.*

1. to give (a metal object) a different form by melting it down and reshaping it.
2. to form, fashion, or arrange again.
3. to remodel or reconstruct (a literary work, document, sentence, etc.).
4. to supply (a theater or opera work) with a new cast.

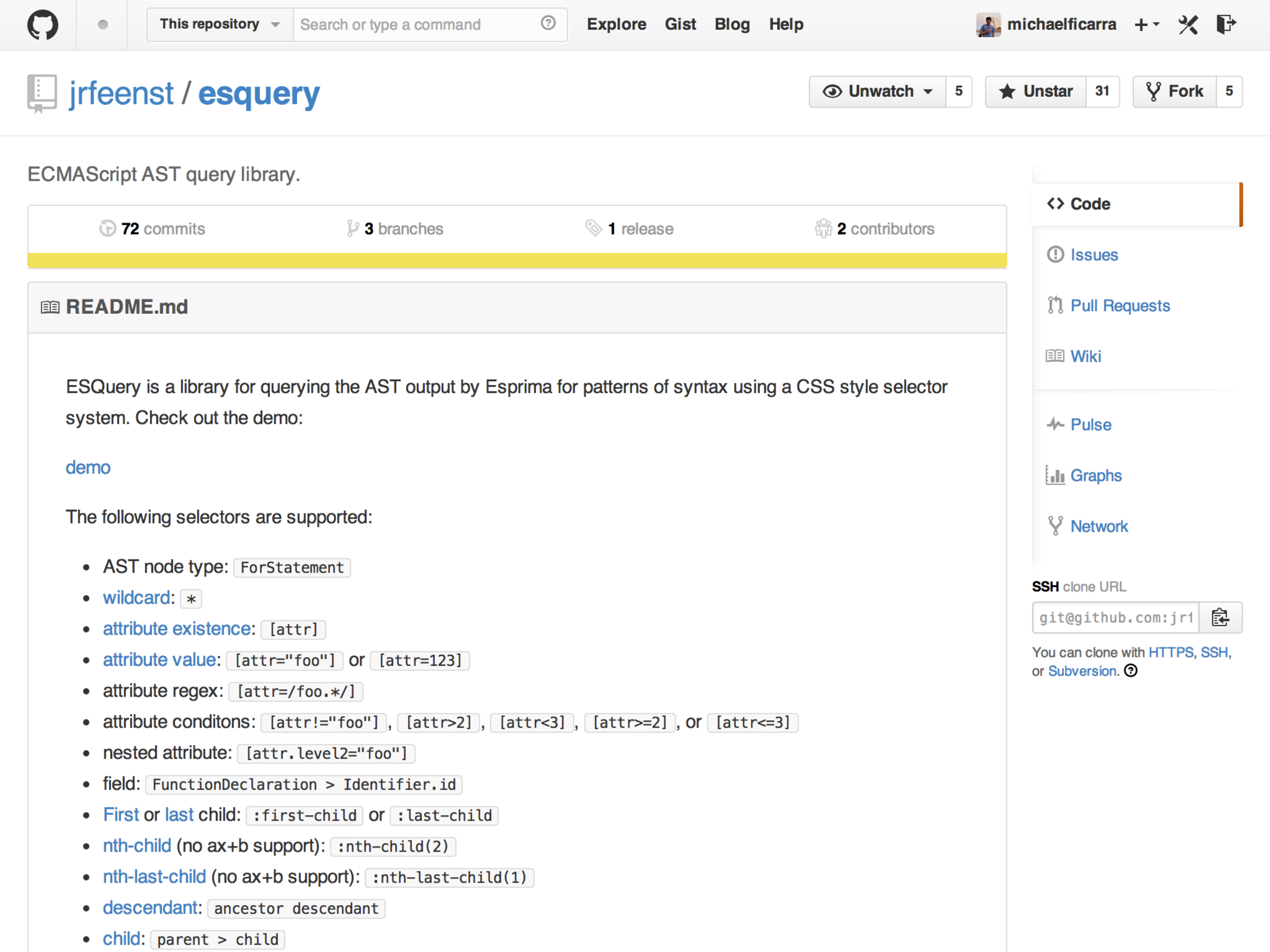
Installation

From NPM:

```
npm install recast
```

From GitHub:

```
cd path/to/node_modules
git clone git://github.com/benjamn/recast.git
cd recast
```



jrfeenst / esquery

ECMAScript AST query library.

72 commits · 3 branches · 1 release · 2 contributors

You can clone with HTTPS, SSH, or Subversion.

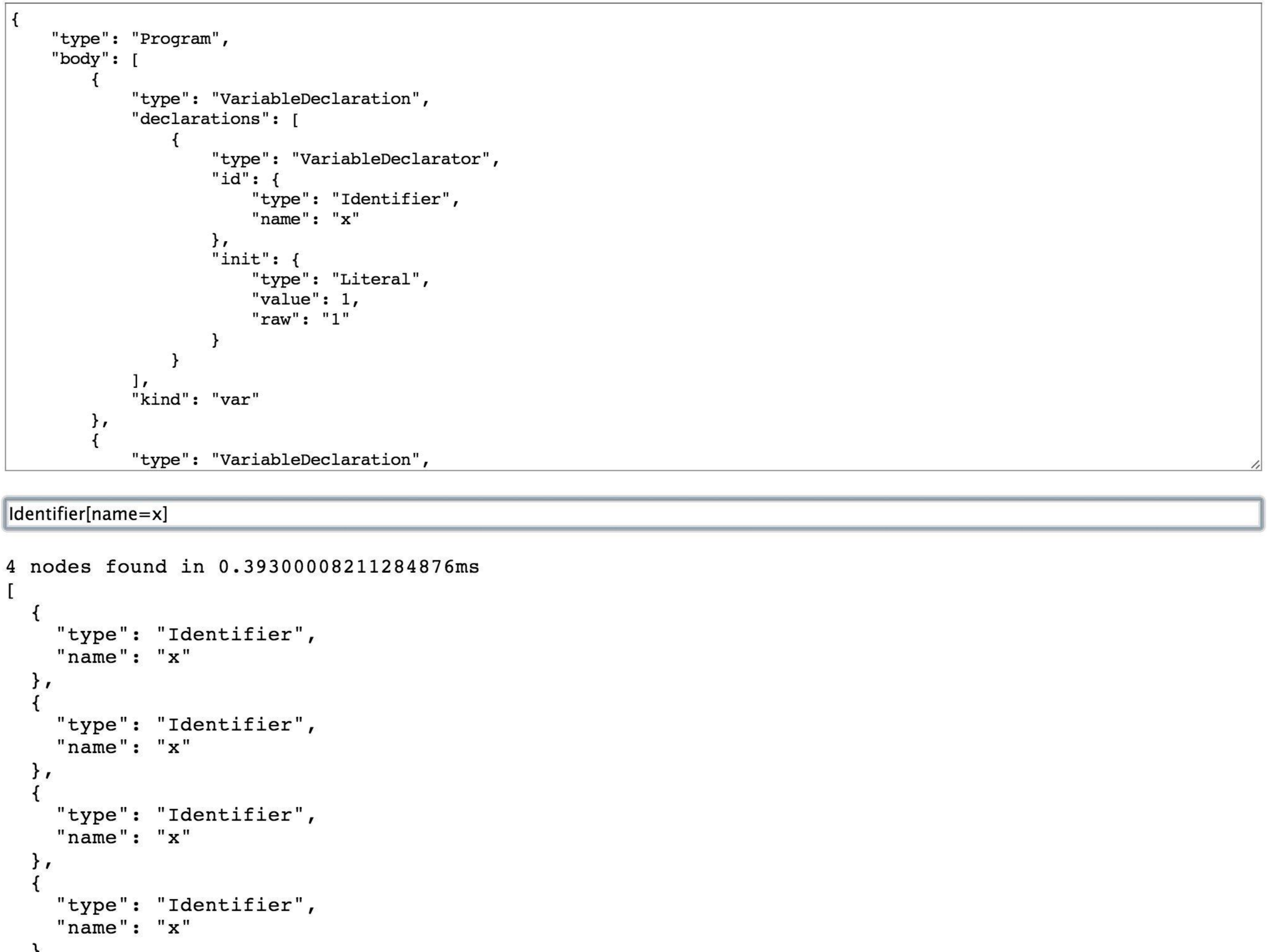
README.md

ESQuery is a library for querying the AST output by Esprima for patterns of syntax using a CSS style selector system. Check out the demo:

demo

The following selectors are supported:

- AST node type: `ForStatement`
- wildcard: `*`
- attribute existence: `[attr]`
- attribute value: `[attr="foo"]` or `[attr=123]`
- attribute regex: `[attr=/foo.*/]`
- attribute conditons: `[attr!="foo"]`, `[attr>2]`, `[attr<3]`, `[attr>=2]`, or `[attr<=3]`
- nested attribute: `[attr.level2="foo"]`
- field: `FunctionDeclaration > Identifier.id`
- First or last child: `:first-child` or `:last-child`
- nth-child (no ax+b support): `:nth-child(2)`
- nth-last-child (no ax+b support): `:nth-last-child(1)`
- descendant: `ancestor descendant`
- child: `parent > child`



```
{
    "type": "Program",
    "body": [
        {
            "type": "VariableDeclaration",
            "declarations": [
                {
                    "type": "VariableDeclarator",
                    "id": {
                        "type": "Identifier",
                        "name": "x"
                    },
                    "init": {
                        "type": "Literal",
                        "value": 1,
                        "raw": "1"
                    }
                }
            ],
            "kind": "var"
        },
        {
            "type": "VariableDeclaration",
```

```
Identifier[name=x]
```

```
4 nodes found in 0.39300008211284876ms
[
  {
    "type": "Identifier",
    "name": "x"
  },
  {
    "type": "Identifier",
    "name": "x"
  },
  {
    "type": "Identifier",
    "name": "x"
  },
  {
    "type": "Identifier",
    "name": "x"
  }
```

Static Analysis

This repository | Search or type a command | Explore Gist Blog Help | michaelficarra

Constellation / escope

Unwatch 14 | Unstar 121 | Fork 21

Escope: ECMAScript scope analyzer

87 commits | 2 branches | 4 releases | 10 contributors

Code | Issues | Pull Requests | Wiki | Pulse | Graphs | Network

SSH clone URL

git@github.com:Cor

You can clone with HTTPS, SSH, or Subversion.

README.md

Escope (escope) is ECMAScript scope analyzer extracted from esmangle project.

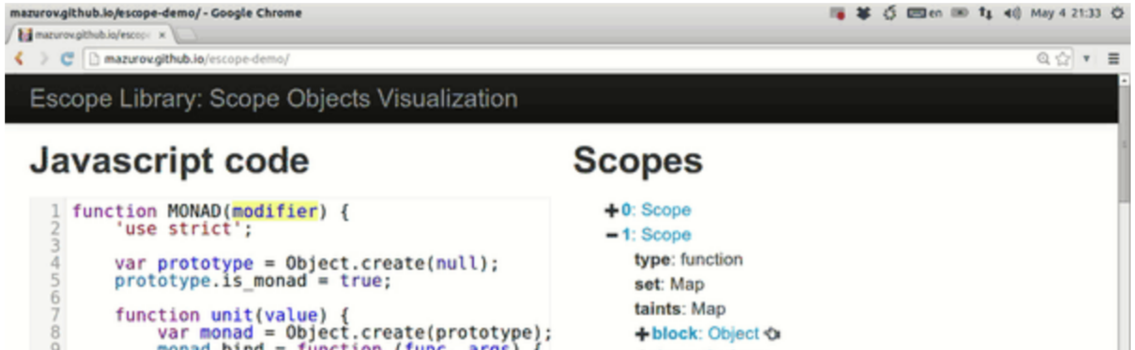
build passing

Document

Generated JSDoc is here.

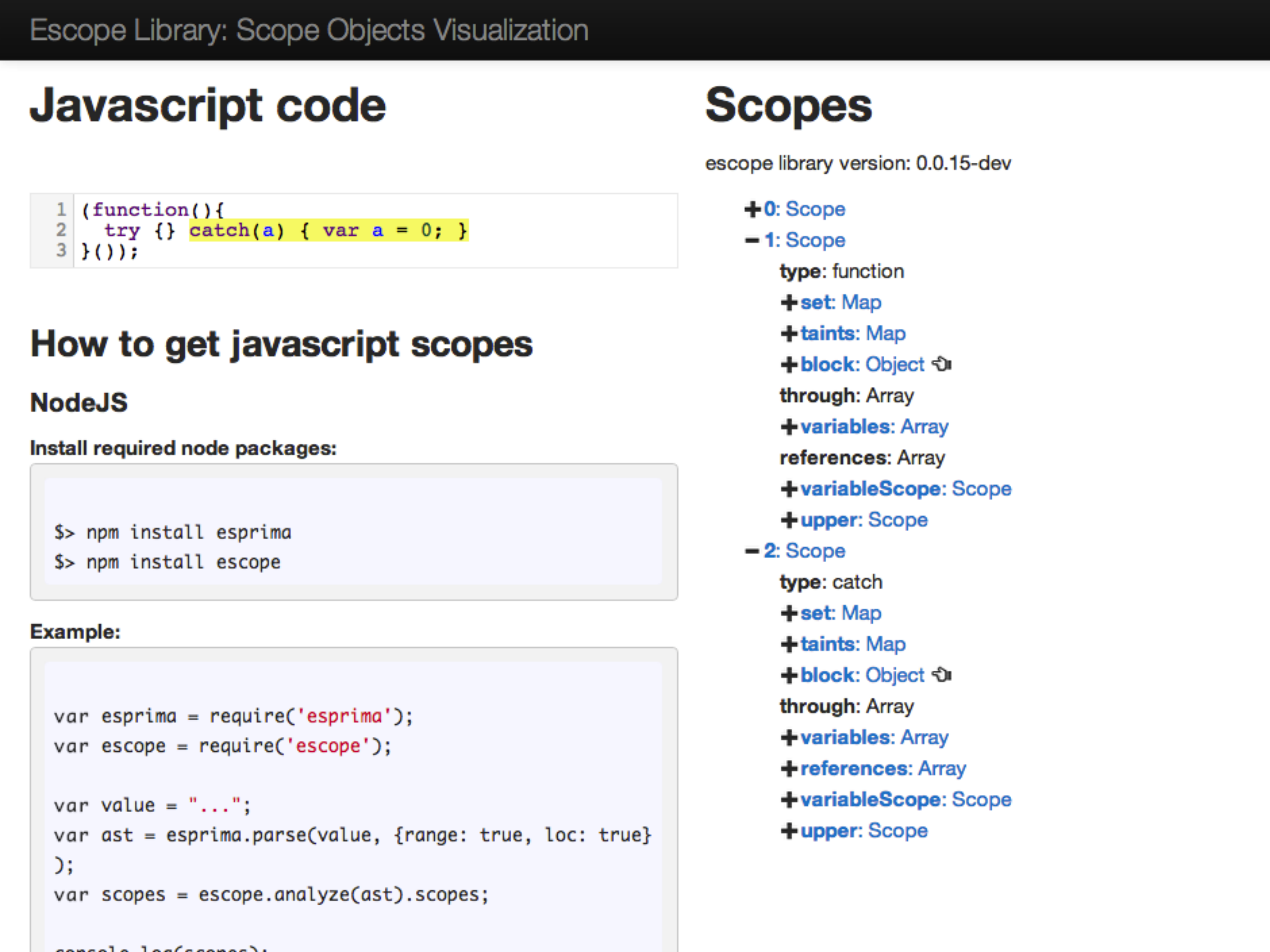
Demos and Tools

Demonstration is here by Sasha Mazurov (twitter: @mazurov). issue


mazurov.github.io/escope-demo/ - Google Chrome
mazurov.github.io/escope-demo/
Escope Library: Scope Objects Visualization
Javascript code
Scopes

```
function MONAD(modifier) {
    'use strict';

    var prototype = Object.create(null);
    prototype.is_monad = true;

    function unit(value) {
        var monad = Object.create(prototype);
```

```
+0: Scope
-1: Scope
  type: function
  set: Map
  taints: Map
  +block: Object
```



Escope Library: Scope Objects Visualization

Javascript code

```
(function(){
  try {} catch(a) { var a = 0; }
}());
```

How to get javascript scopes

NodeJS

Install required node packages:

```
$> npm install esprima
$> npm install escope
```

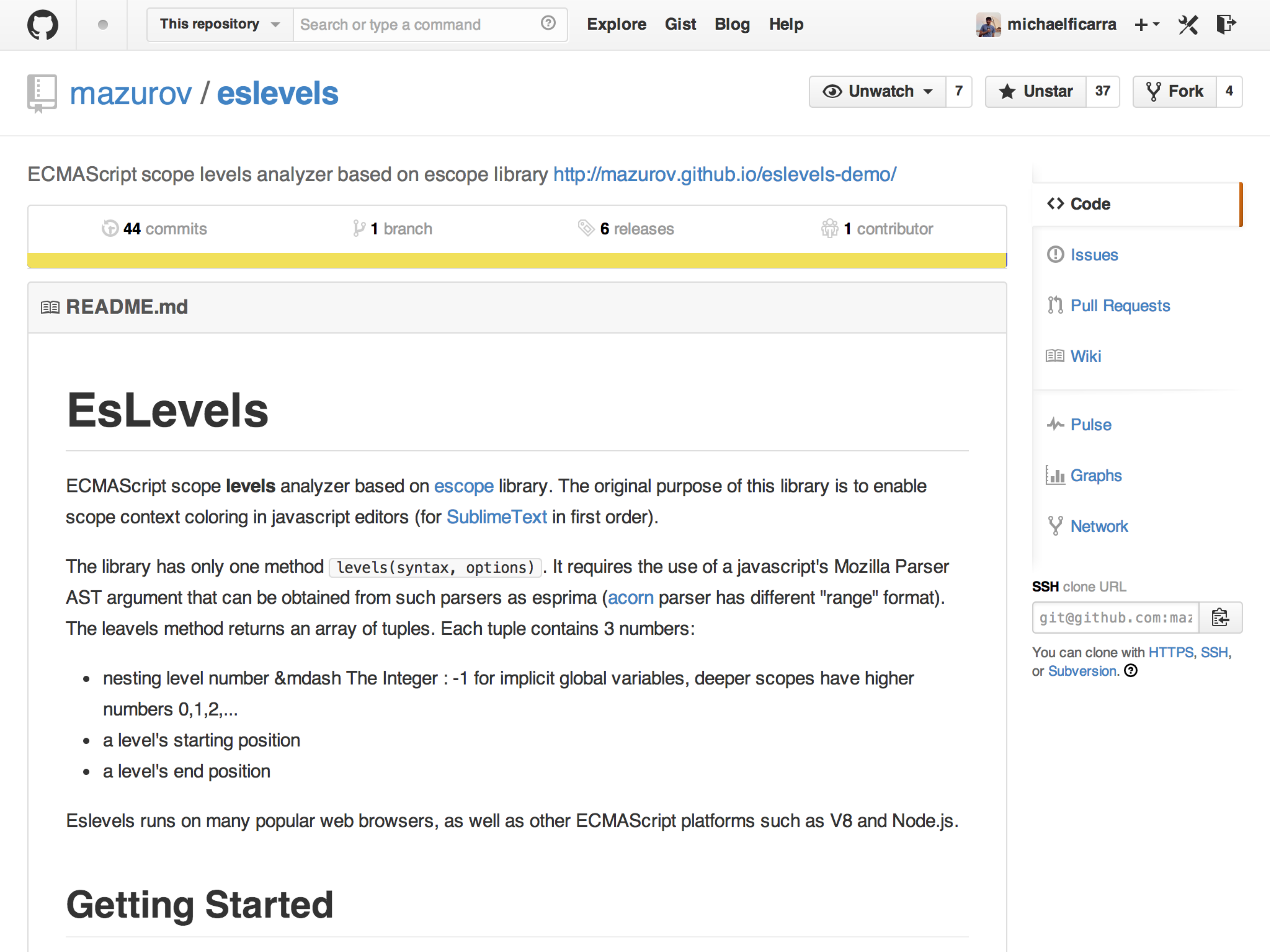
Example:

```
var esprima = require('esprima');
var escope = require('escope');

var value = "...";
var ast = esprima.parse(value, {range: true, loc: true}
);
var scopes = escope.analyze(ast).scopes;

console.log(scopes);
```

Scopes

escope library version: 0.0.15-dev

```
+0: Scope
−1: Scope
    type: function
    +set: Map
    +taints: Map
    +block: Object
    through: Array
    +variables: Array
    references: Array
    +variableScope: Scope
    +upper: Scope
−2: Scope
    type: catch
    +set: Map
    +taints: Map
    +block: Object
    through: Array
    +variables: Array
    +references: Array
    +variableScope: Scope
    +upper: Scope
```



mazurov / eslevels

ECMAScript scope levels analyzer based on escope library http://mazurov.github.io/eslevels-demo/

44 commits 1 branch 6 releases 1 contributor

README.md

EsLevels

ECMAScript scope **levels** analyzer based on escope library. The original purpose of this library is to enable scope context coloring in javascript editors (for SublimeText in first order).

The library has only one method `levels(syntax, options)`. It requires the use of a javascript's Mozilla Parser AST argument that can be obtained from such parsers as esprima (acorn parser has different "range" format). The leavels method returns an array of tuples. Each tuple contains 3 numbers:

- nesting level number &mdash The Integer : -1 for implicit global variables, deeper scopes have higher numbers 0,1,2,...
- a level's starting position
- a level's end position

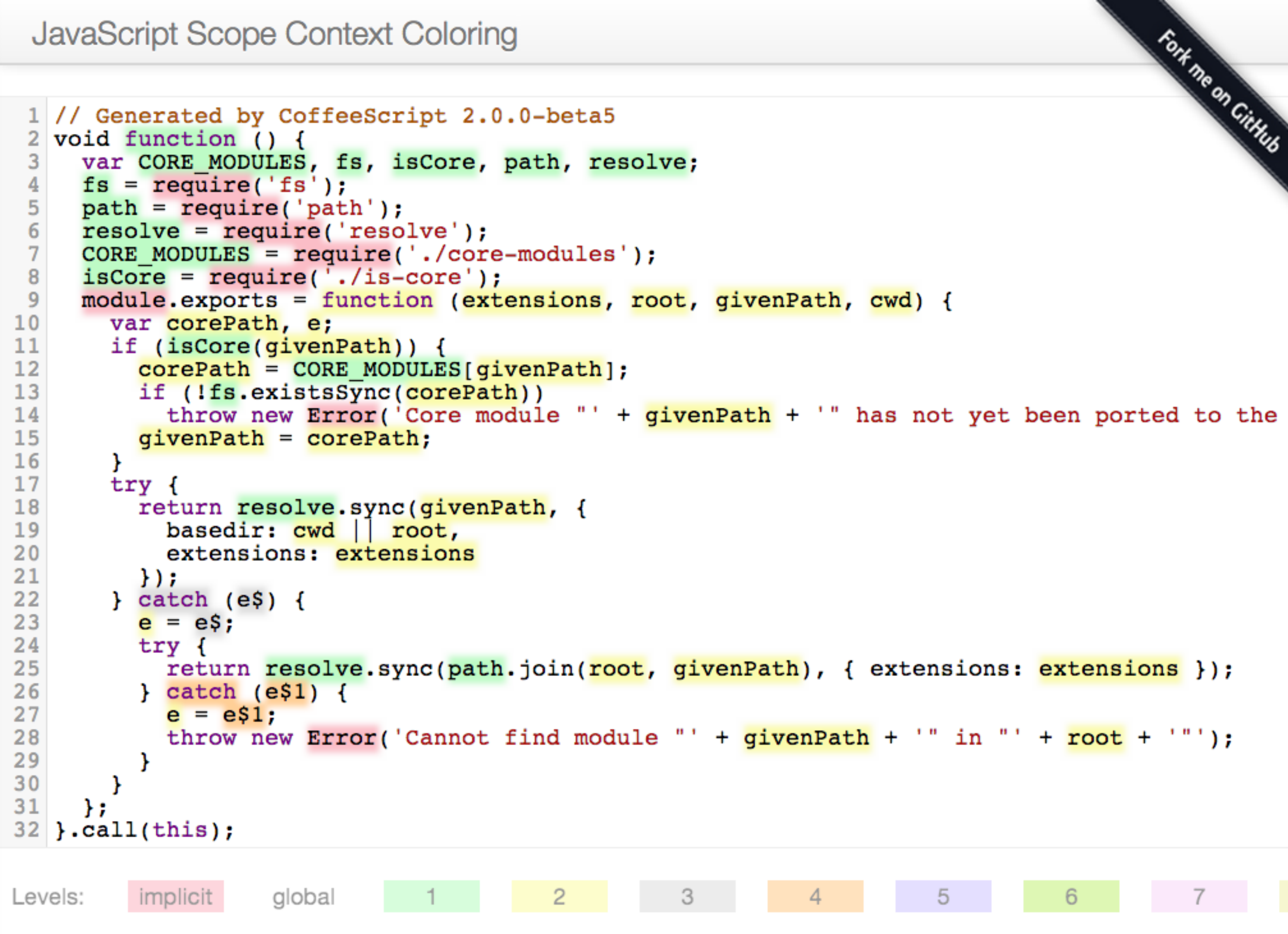
Eslevels runs on many popular web browsers, as well as other ECMAScript platforms such as V8 and Node.js.

Getting Started

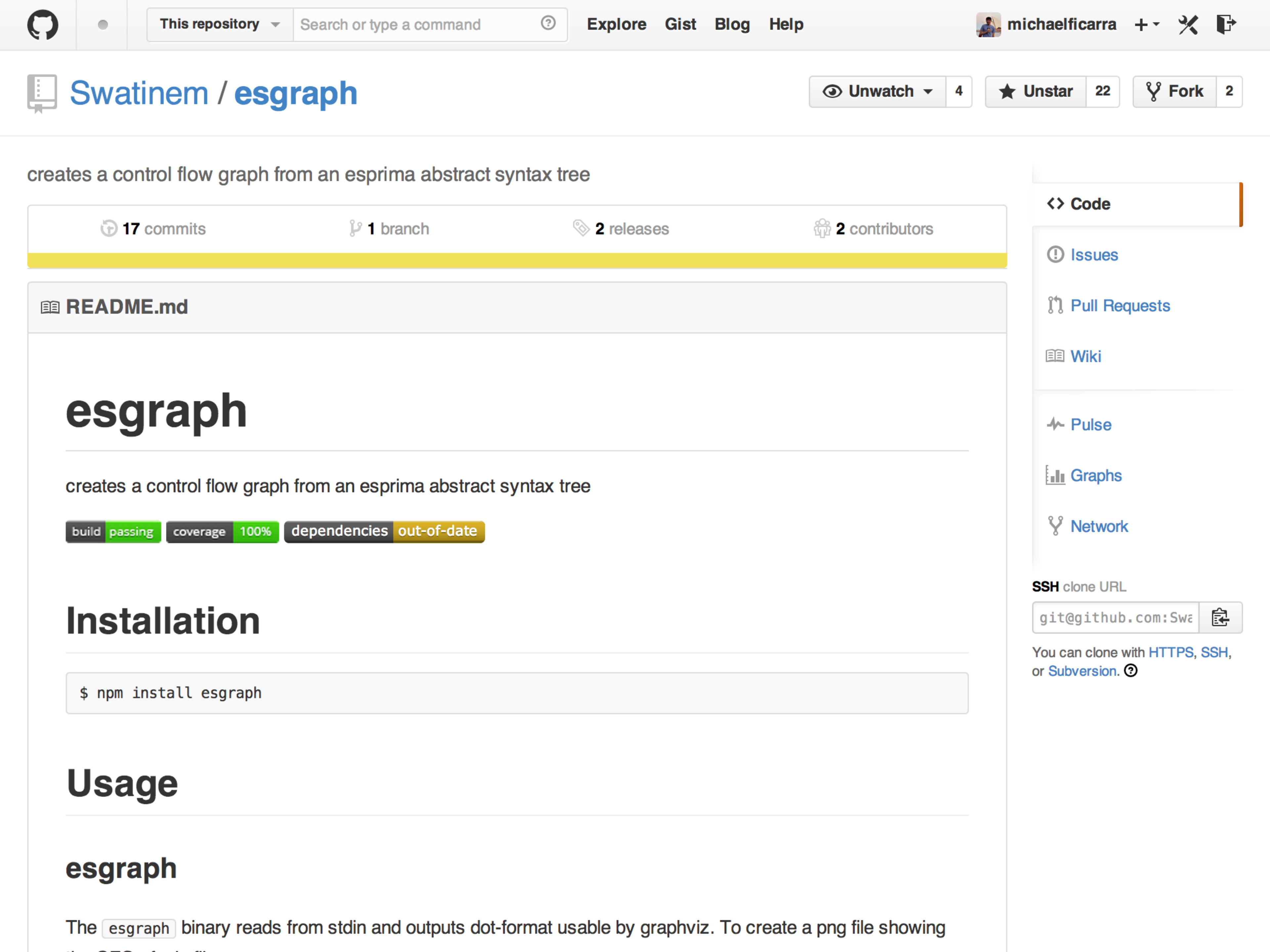
SSH clone URL

`git@github.com:maz`

You can clone with HTTPS, SSH, or Subversion.



```
// Generated by CoffeeScript 2.0.0-beta5
void function () {
  var CORE_MODULES, fs, isCore, path, resolve;
  fs = require('fs');
  path = require('path');
  resolve = require('resolve');
  CORE_MODULES = require('./core-modules');
  isCore = require('./is-core');
  module.exports = function (extensions, root, givenPath, cwd) {
    var corePath, e;
    if (isCore(givenPath)) {
      corePath = CORE_MODULES[givenPath];
      if (!fs.existsSync(corePath))
        throw new Error('Core module "' + givenPath + '" has not yet been ported to the
      givenPath = corePath;
    }
    try {
      return resolve.sync(givenPath, {
        basedir: cwd || root,
        extensions: extensions
      });
    } catch (e$) {
      e = e$;
      try {
        return resolve.sync(path.join(root, givenPath), { extensions: extensions });
      } catch (e$1) {
        e = e$1;
        throw new Error('Cannot find module "' + givenPath + '" in "' + root + '"');
      }
    }
  };
}.call(this);
```



Swatinem / esgraph

creates a control flow graph from an esprima abstract syntax tree

You can clone with HTTPS, SSH, or Subversion.

README.md

esgraph

creates a control flow graph from an esprima abstract syntax tree

Installation

```
$ npm install esgraph
```

Usage

esgraph

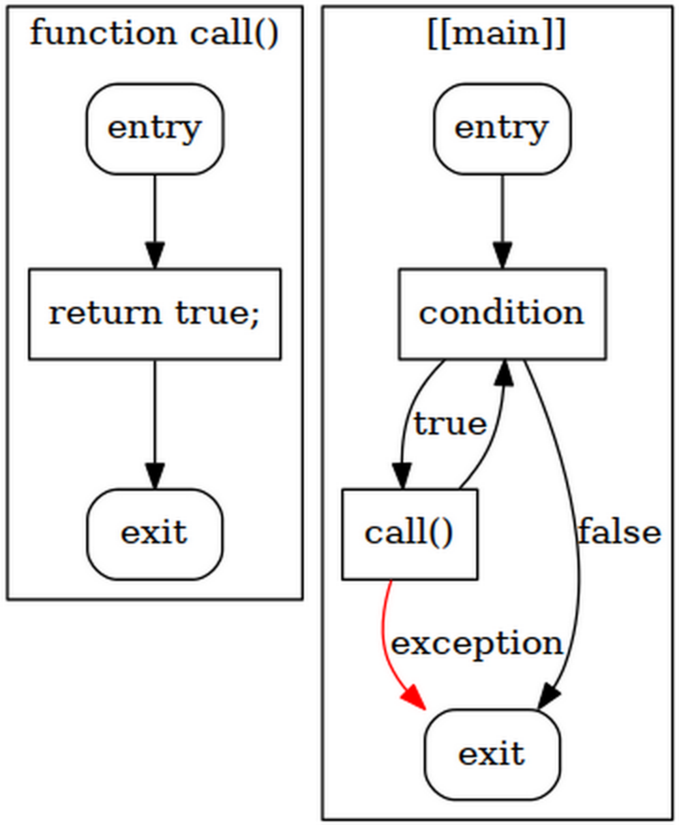
The `esgraph` binary reads from stdin and outputs dot-format usable by graphviz. To create a png file showing

Usage

esgraph

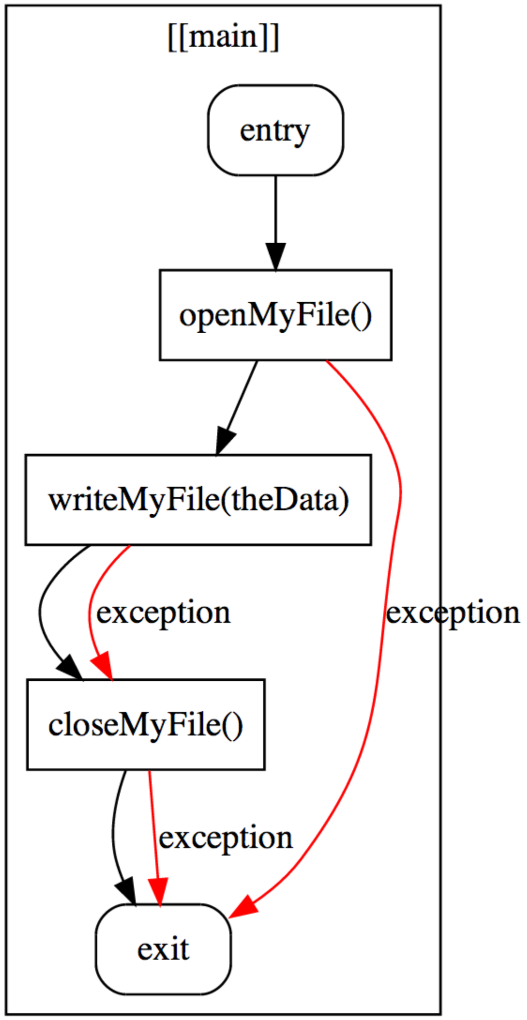
The `esgraph` binary reads from stdin and outputs dot-format usable by graphviz. To create a png file showing the CFG of a js file:

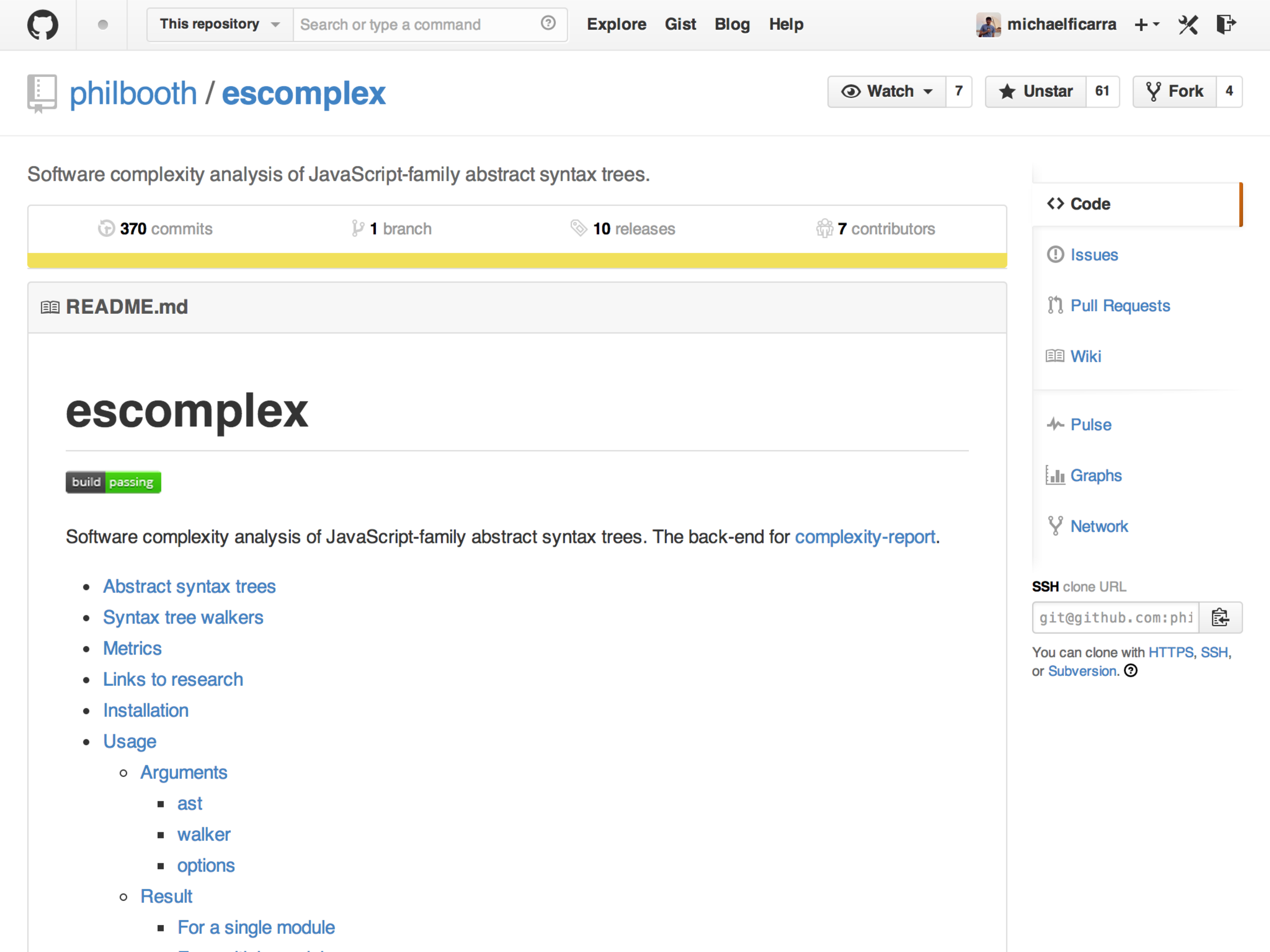
```
$ cat $file | esgraph | dot -Tpng > output.png
```



```
openMyFile()
try {
    writeMyFile(theData);
} finally {
    closeMyFile();
}
```

Convert!





philbooth / escomplex

Software complexity analysis of JavaScript-family abstract syntax trees.

README.md

escomplex

Software complexity analysis of JavaScript-family abstract syntax trees. The back-end for complexity-report.

- Abstract syntax trees
- Syntax tree walkers
- Metrics
- Links to research
- Installation
- Usage
  - Arguments
    - ast
    - walker
    - options
  - Result
    - For a single module

SSH clone URL

```
git@github.com:phi
```

You can clone with HTTPS, SSH, or Subversion.

For a single module

If a single abstract syntax tree object is passed in the `ast` argument, the result will be a report object that looks like the following:

```
{
    maintainability: 171,
    dependencies: [],
    aggregate: {
        sloc: {
            logical: 0,
            physical: 0
        },
        params: 0,
        cyclomatic: 1,
        cyclomaticDensity: 1,
        halstead: {
            vocabulary: 0,
            difficulty: 0,
            volume: 0,
            effort: 0,
            bugs: 0,
            time: 0
        }
    },
    functions: [
        {
            name: '',
            line: 0,
            sloc: {
                logical: 0,
                physical: 0
            },
            params: 0,
            cyclomatic: 1,
            cyclomaticDensity: 1,
            halstead: {
                vocabulary: 0,
                difficulty: 0
```

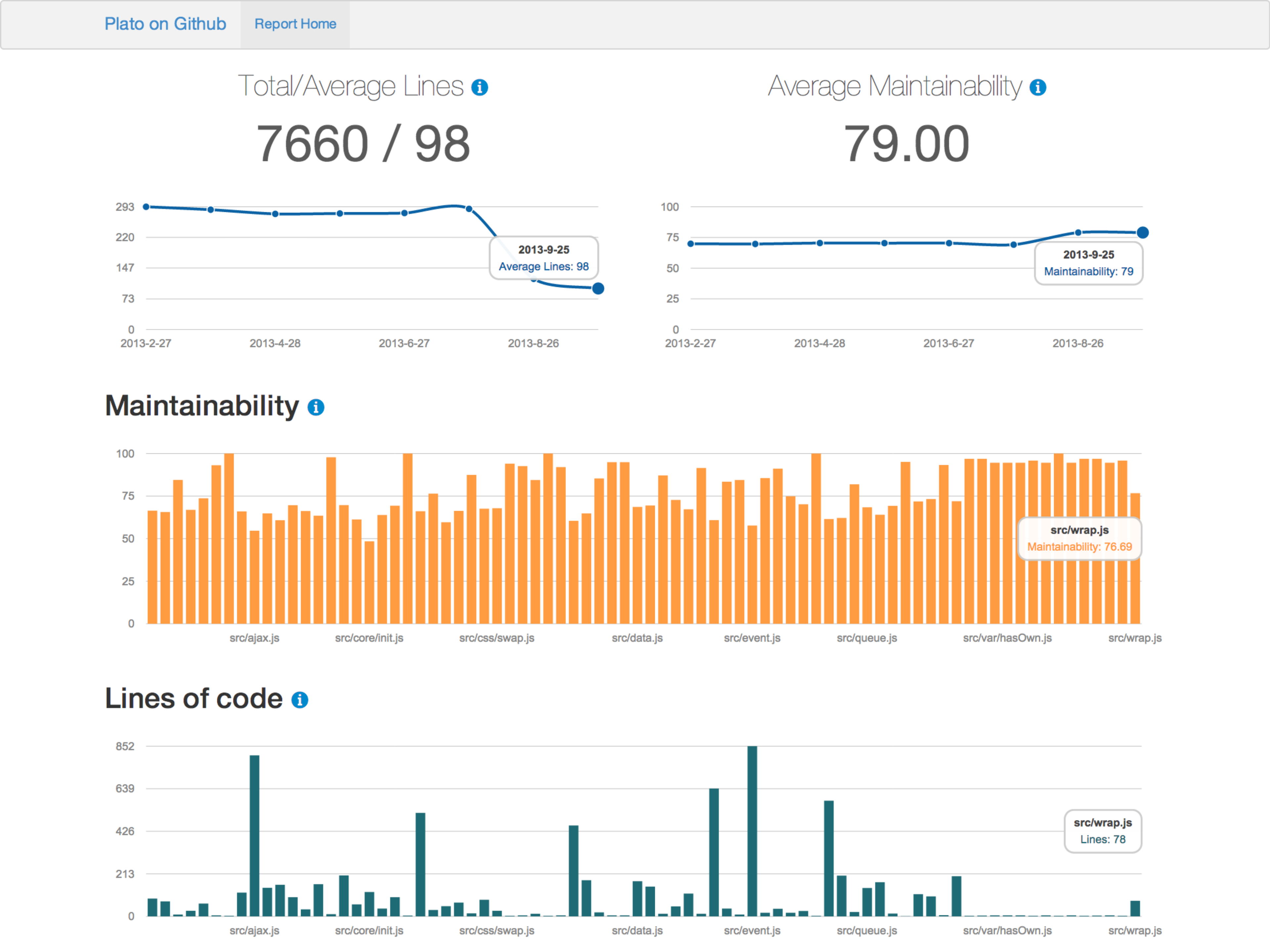
For multiple modules

If an array of syntax trees is passed in the `ast` argument, the result will be an object that looks like the following:

```
{
    reports: [
        ...
    ],
    adjacencyMatrix: [
        [ 0 ]
    ],
    firstOrderDensity: 0,
    visibilityMatrix: [
        [ 0 ]
    ],
    changeCost: 100,
    coreSize: 100
}
```

Those properties are defined as follows:

- `result.reports`: An array of report objects, each one in the same format described above but with an extra property `path` that matches the `path` property from its corresponding syntax tree. This `path` property is required because the reports array gets sorted during dependency analysis.
- `result.adjacencyMatrix`: The adjacency design structure matrix (DSM) for the project. This is a two-dimensional array, each dimension with the same order and length as the `reports` array. Each row and column represents its equivalent indexed module from the `reports` array, with values along the horizontal being `1` when that module directly depends on another and values along the vertical being `1` when that module is directly depended on by another. All other values are `0`.
- `result.firstOrderDensity`: The first-order density for the project.
- `result.visibilityMatrix`: The visibility DSM for the project. Like the adjacency matrix, but expanded to incorporate indirect dependencies.
- `result.changeCost`: The change cost for the project.
- `result.coreSize`: The core size for the project.



Total/Average Lines

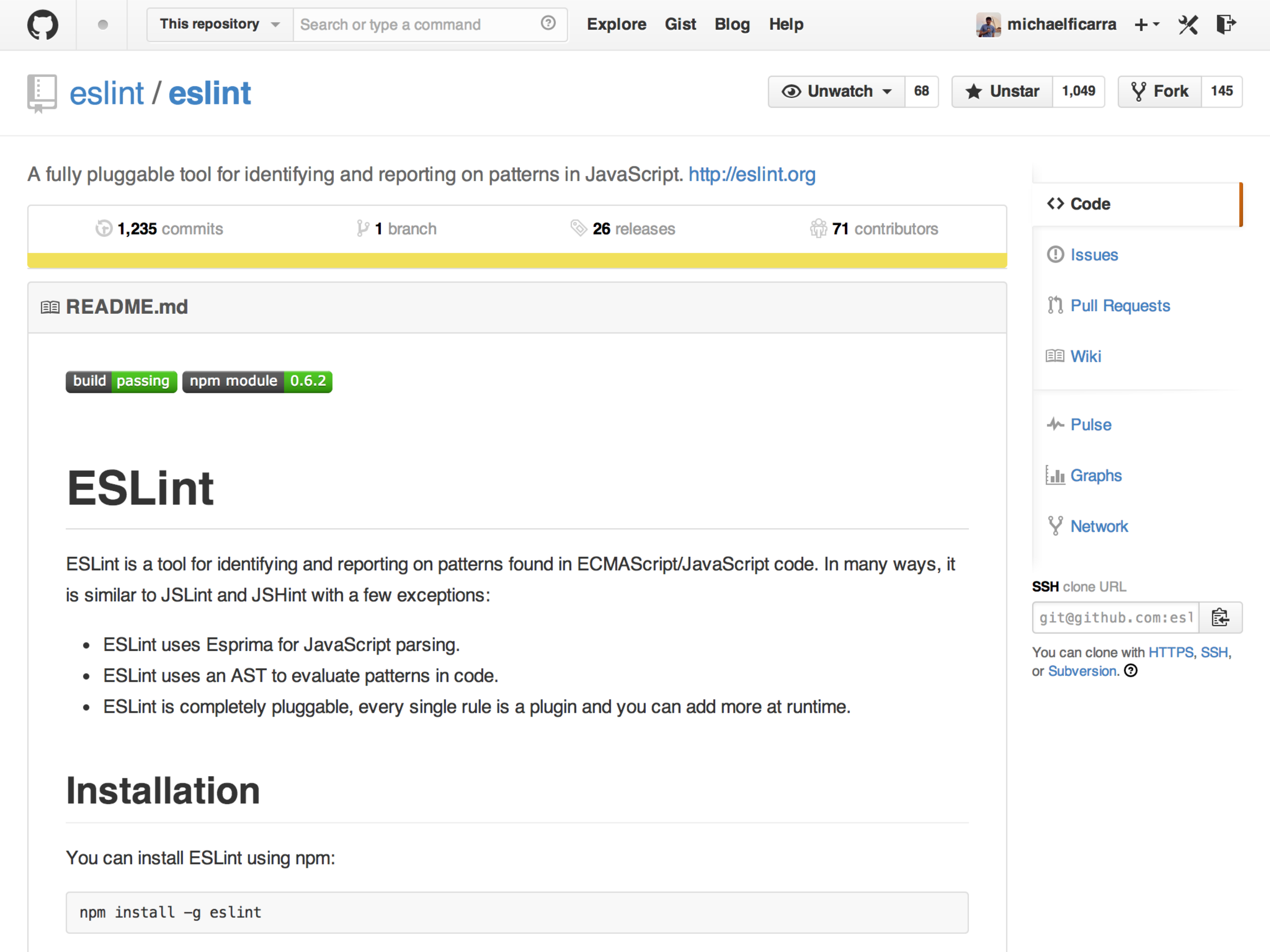
7660 / 98

Average Maintainability

79.00

Maintainability

Lines of code



A fully pluggable tool for identifying and reporting on patterns in JavaScript. http://eslint.org

1,235 commits | 1 branch | 26 releases | 71 contributors

README.md

build passing | npm module 0.6.2

ESLint

ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code. In many ways, it is similar to JSLint and JSHint with a few exceptions:

- ESLint uses Esprima for JavaScript parsing.
- ESLint uses an AST to evaluate patterns in code.
- ESLint is completely pluggable, every single rule is a plugin and you can add more at runtime.

Installation

You can install ESLint using npm:

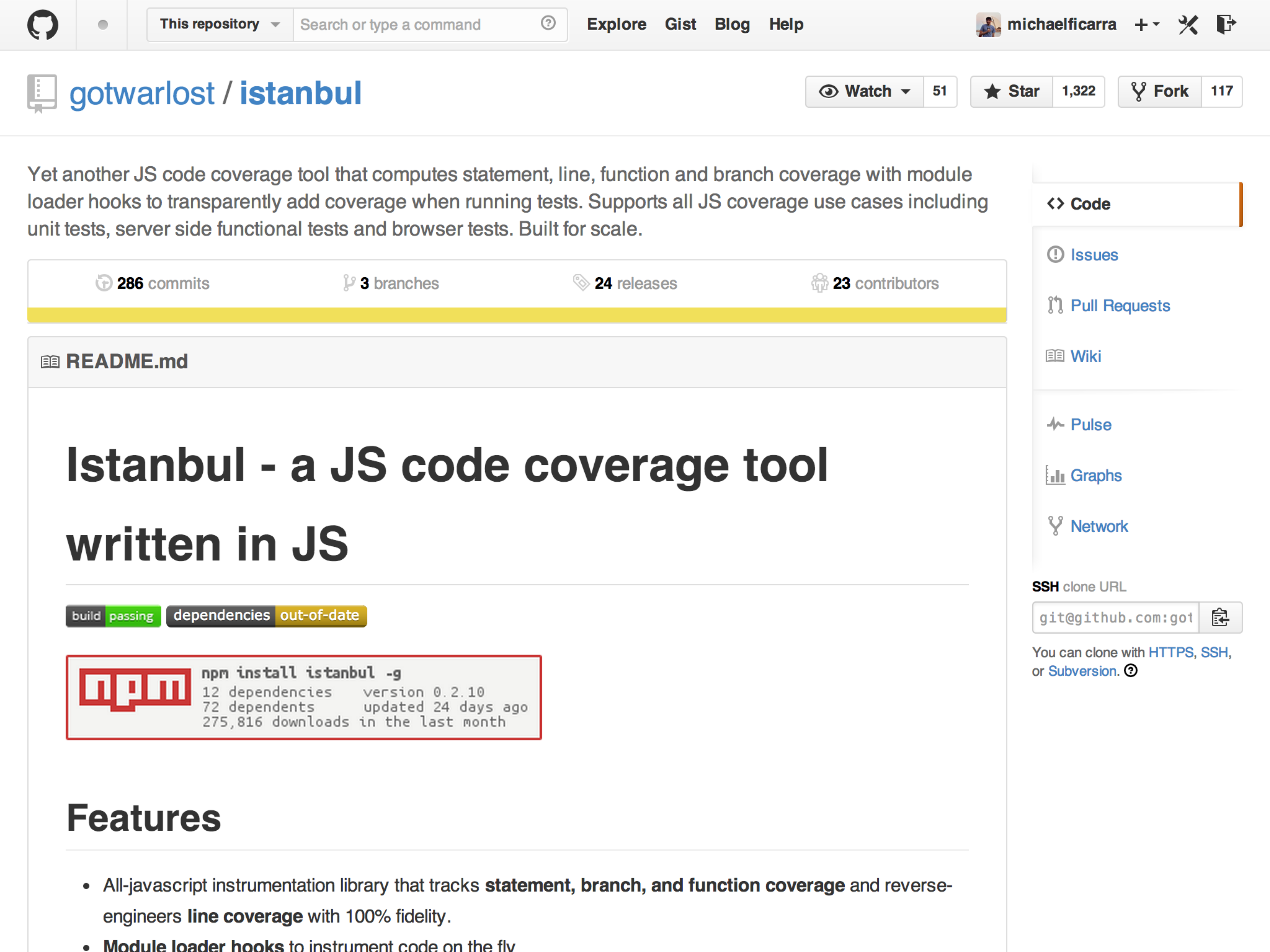
```
npm install -g eslint
```

SSH clone URL

git@github.com:esl

You can clone with HTTPS, SSH, or Subversion.

```javascript
"use strict";

// Disallow sparse arrays

module.exports = function(context) {

  return {

    "ArrayExpression": function(node) {
      if (node.elements.indexOf(null) > -1) {
        context.report(node, "Unexpected comma in middle of array.");
      }
    }

  };

};
```



gotwarlost / istanbul

Yet another JS code coverage tool that computes statement, line, function and branch coverage with module loader hooks to transparently add coverage when running tests. Supports all JS coverage use cases including unit tests, server side functional tests and browser tests. Built for scale.

286 commits   3 branches   24 releases   23 contributors

README.md

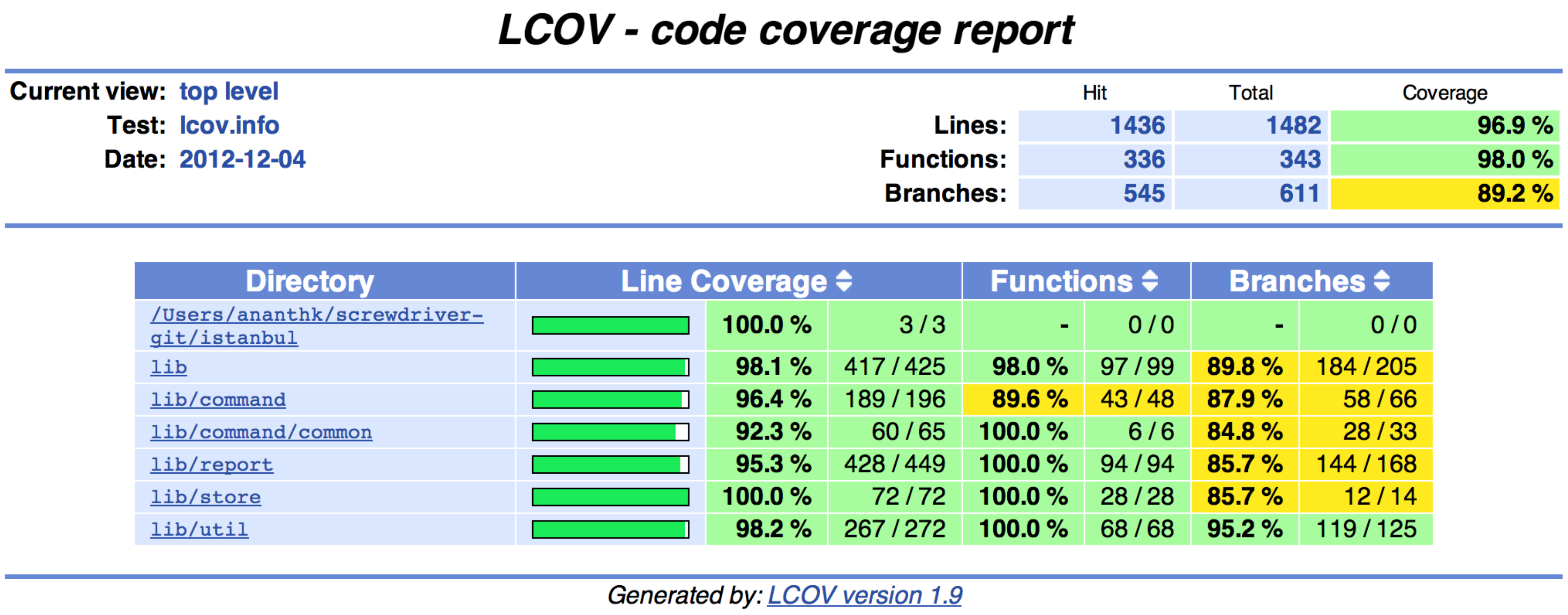
Istanbul - a JS code coverage tool written in JS

```
npm install istanbul -g
12 dependencies      version 0.2.10
72 dependents        updated 24 days ago
275,816 downloads in the last month
```

Features

- All-javascript instrumentation library that tracks **statement, branch, and function coverage** and reverse-engineers **line coverage** with 100% fidelity.
- **Module loader hooks** to instrument code on the fly

You can clone with HTTPS, SSH, or Subversion.



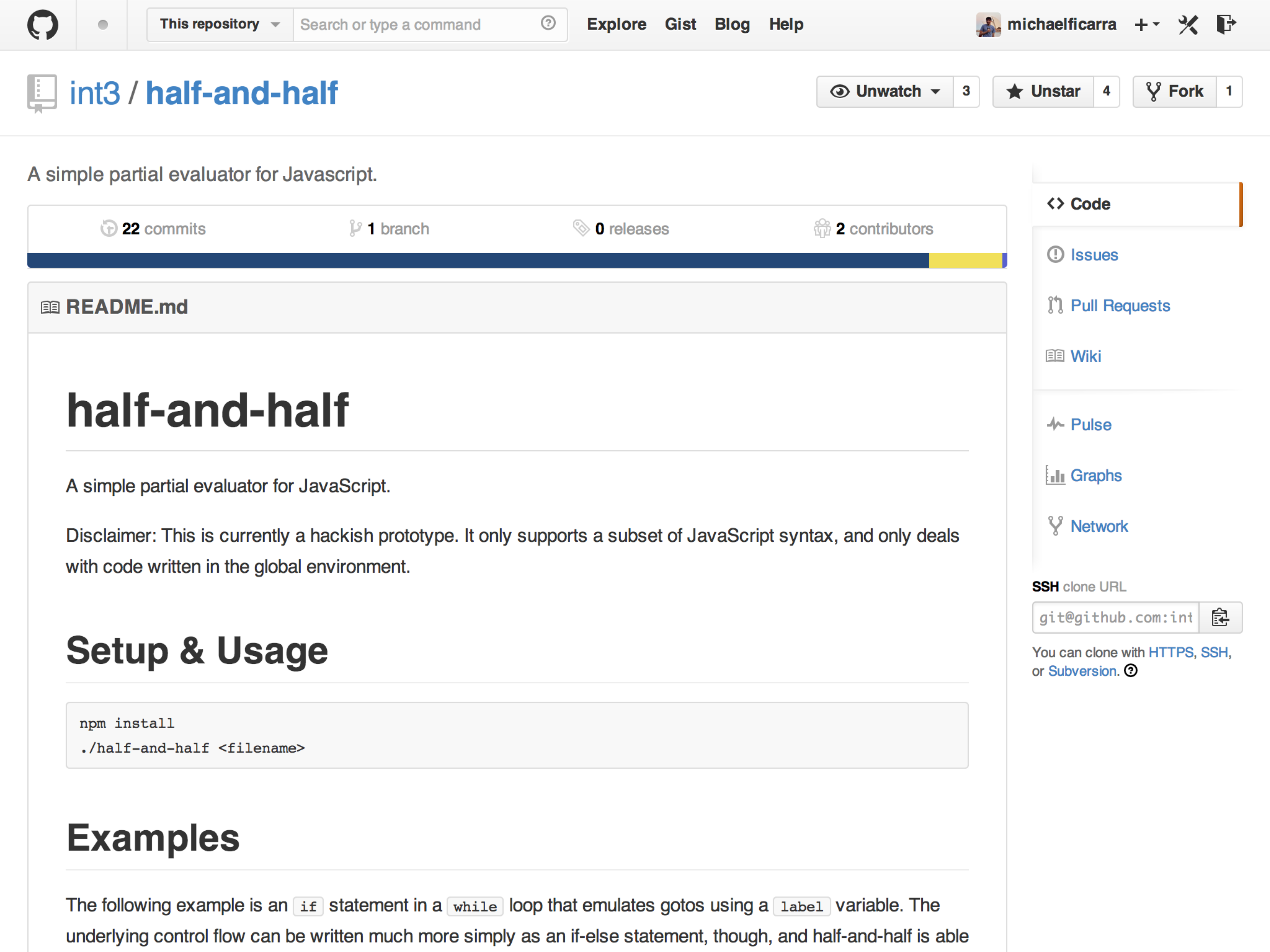
LCOV - code coverage report

Current view:	top level		Hit	Total	Coverage
Test:	lcov.info	**Lines:**	1436	1482	96.9 %
Date:	2012-12-04	**Functions:**	336	343	98.0 %
		Branches:	545	611	89.2 %

Directory	Line Coverage ⬍			Functions ⬍		Branches ⬍	
/Users/ananthk/screwdriver-git/istanbul	🟩	100.0 %	3 / 3	-	0 / 0	-	0 / 0
lib	🟩	98.1 %	417 / 425	98.0 %	97 / 99	89.8 %	184 / 205
lib/command	🟩	96.4 %	189 / 196	89.6 %	43 / 48	87.9 %	58 / 66
lib/command/common	🟩	92.3 %	60 / 65	100.0 %	6 / 6	84.8 %	28 / 33
lib/report	🟩	95.3 %	428 / 449	100.0 %	94 / 94	85.7 %	144 / 168
lib/store	🟩	100.0 %	72 / 72	100.0 %	28 / 28	85.7 %	12 / 14
lib/util	🟩	98.2 %	267 / 272	100.0 %	68 / 68	95.2 %	119 / 125

Generated by: LCOV version 1.9

Evaluation



A simple partial evaluator for Javascript.

22 commits 1 branch 0 releases 2 contributors

README.md

half-and-half

A simple partial evaluator for JavaScript.

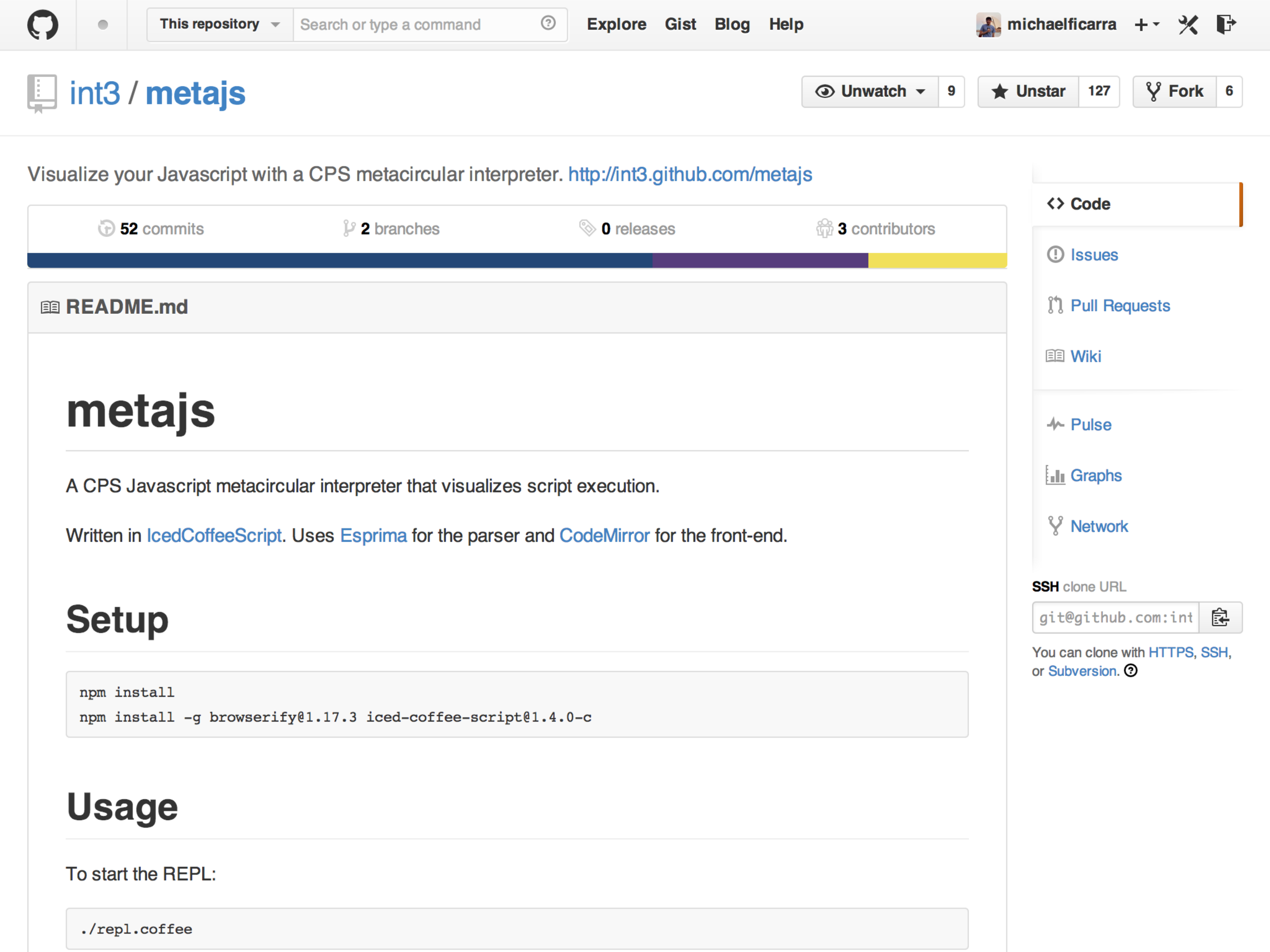
Disclaimer: This is currently a hackish prototype. It only supports a subset of JavaScript syntax, and only deals with code written in the global environment.

Setup & Usage

```
npm install
./half-and-half <filename>
```

Examples

The following example is an `if` statement in a `while` loop that emulates gotos using a `label` variable. The underlying control flow can be written much more simply as an if-else statement, though, and half-and-half is able



int3 / metajs

Visualize your Javascript with a CPS metacircular interpreter. http://int3.github.com/metajs

52 commits 2 branches 0 releases 3 contributors

README.md

metajs

A CPS Javascript metacircular interpreter that visualizes script execution.

Written in IcedCoffeeScript. Uses Esprima for the parser and CodeMirror for the front-end.

Setup

```
npm install
npm install -g browserify@1.17.3 iced-coffee-script@1.4.0-c
```

Usage

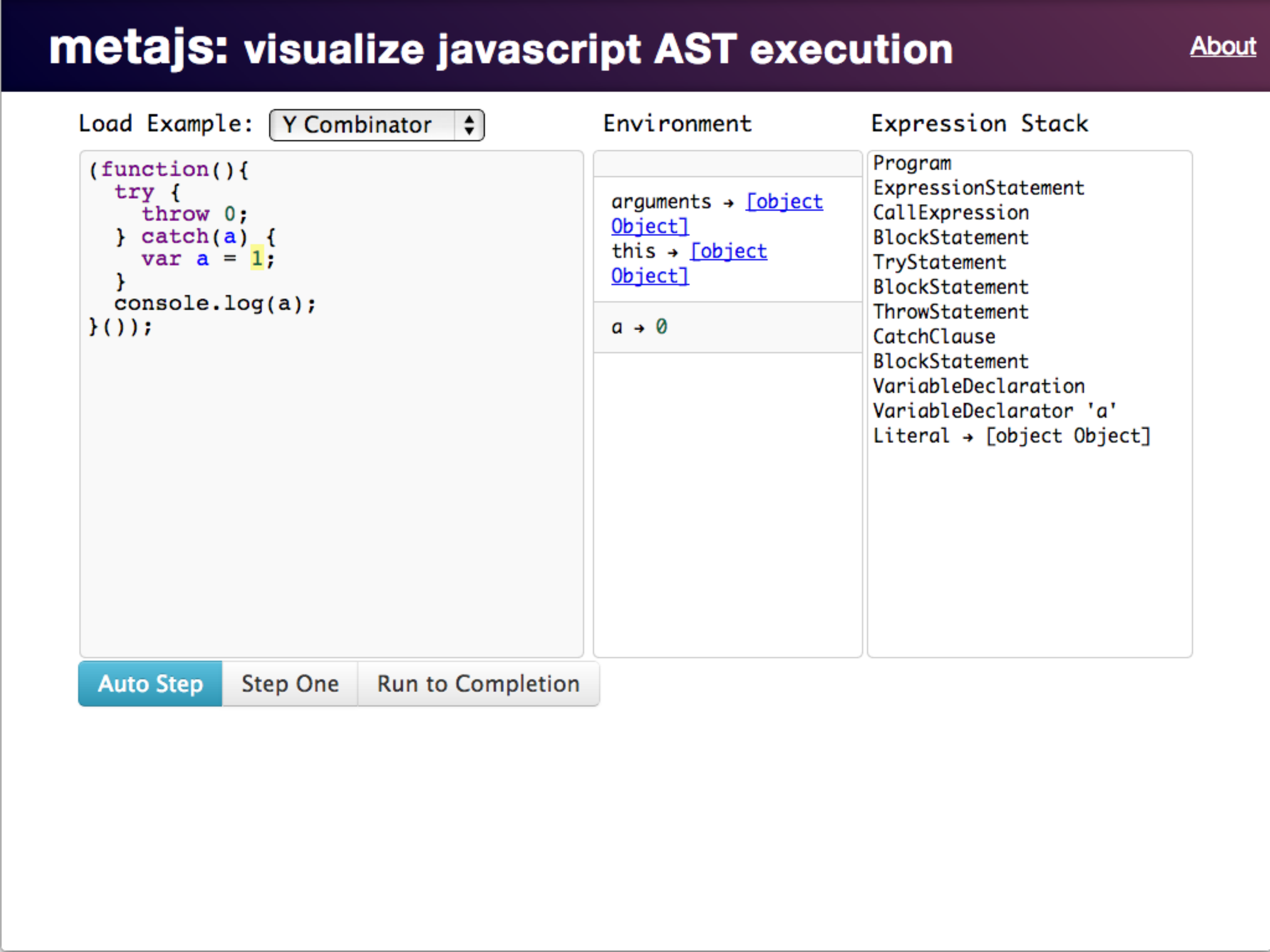
To start the REPL:

```
./repl.coffee
```

SSH clone URL

git@github.com:int

You can clone with HTTPS, SSH, or Subversion.

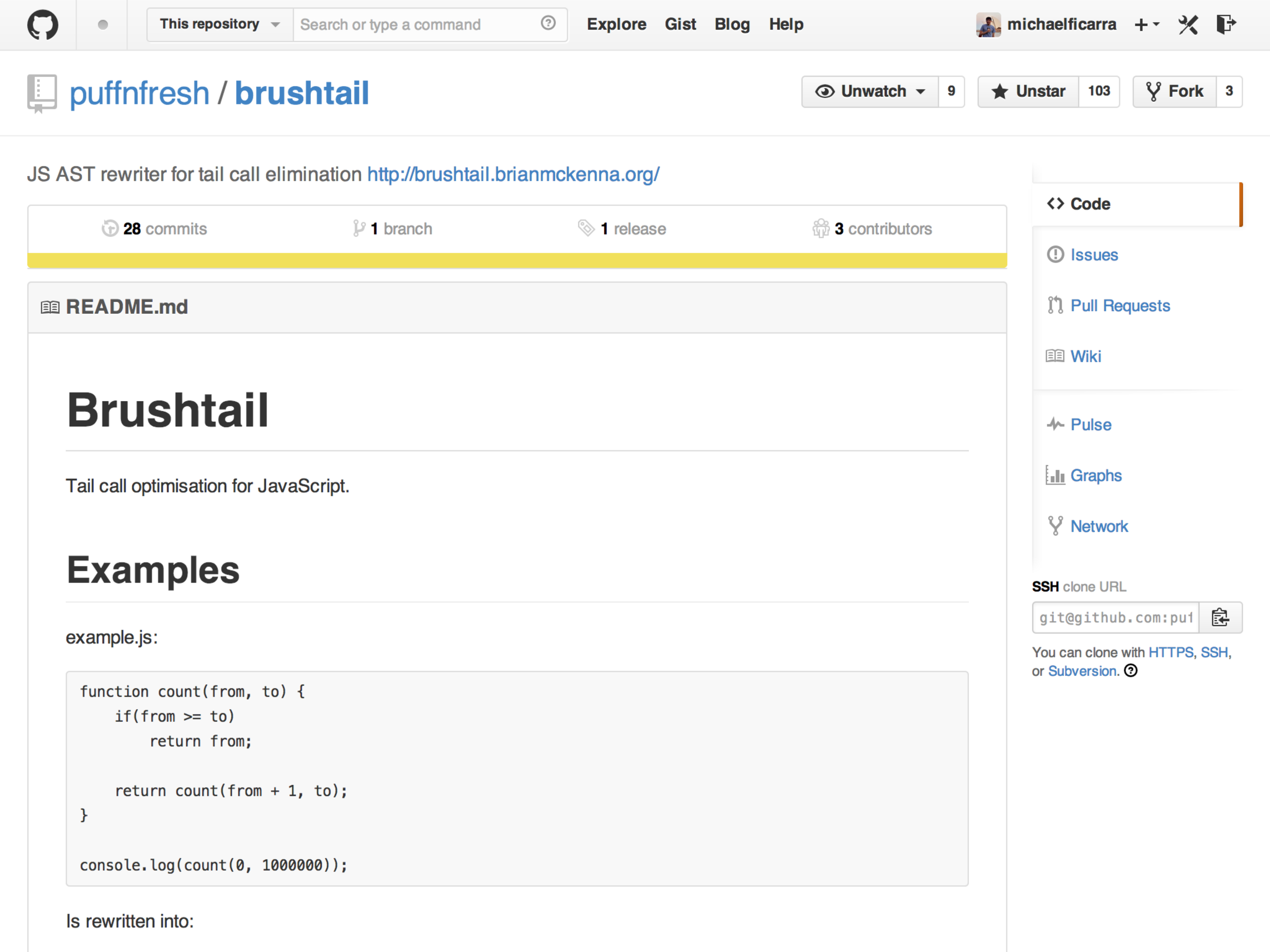

metajs: visualize javascript AST execution
About
Load Example: Y Combinator
Environment
Expression Stack
Auto Step
Step One
Run to Completion

```
(function(){
  try {
    throw 0;
  } catch(a) {
    var a = 1;
  }
  console.log(a);
}());
```

```
arguments → [object Object]
this → [object Object]

a → 0
```

```
Program
ExpressionStatement
CallExpression
BlockStatement
TryStatement
BlockStatement
ThrowStatement
CatchClause
BlockStatement
VariableDeclaration
VariableDeclarator 'a'
Literal → [object Object]
```

Program Transformation



puffnfresh / brushtail

JS AST rewriter for tail call elimination http://brushtail.brianmckenna.org/

28 commits 1 branch 1 release 3 contributors

README.md

Brushtail

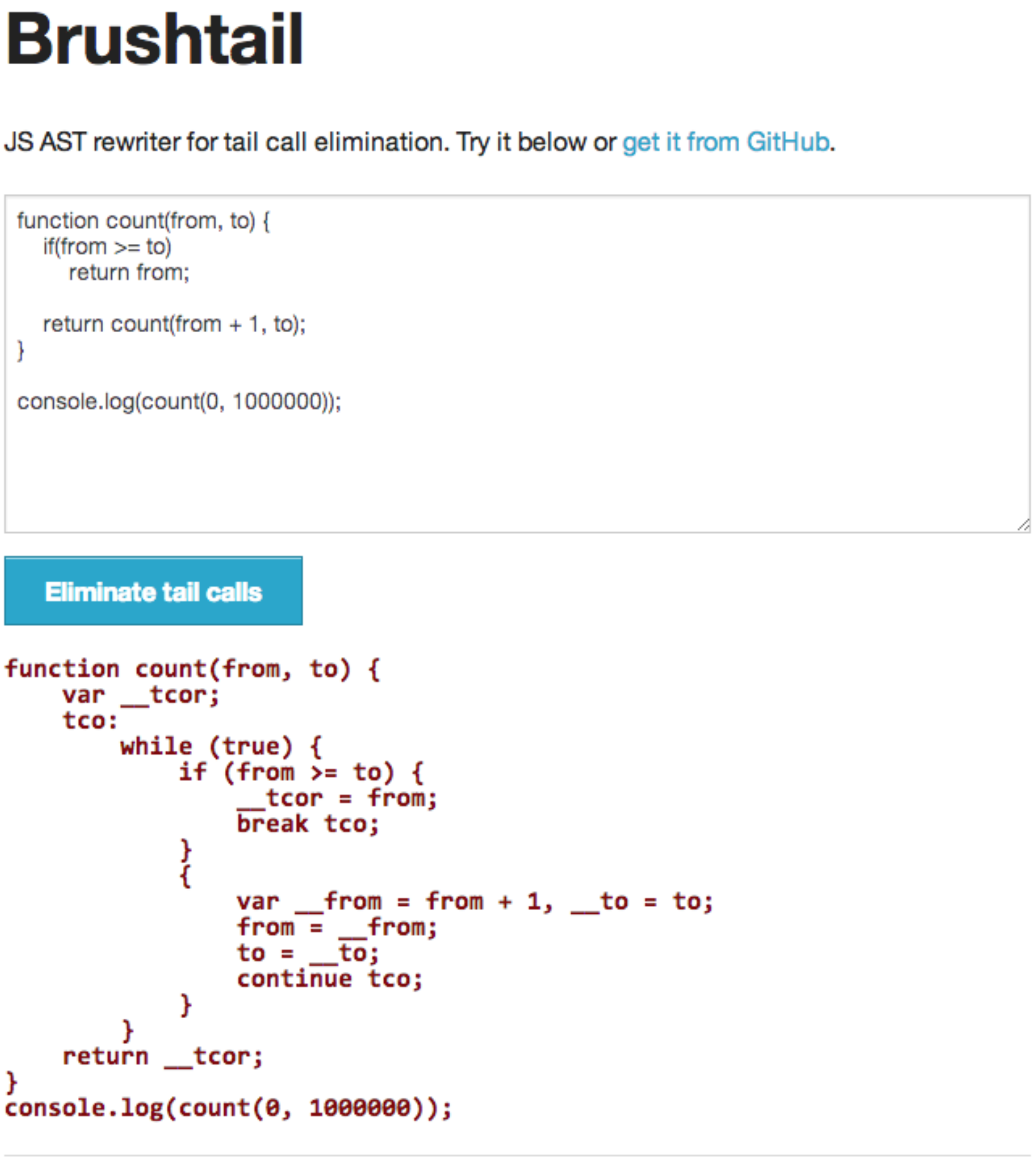
Tail call optimisation for JavaScript.

Examples

example.js:

```
function count(from, to) {
    if(from >= to)
        return from;

    return count(from + 1, to);
}

console.log(count(0, 1000000));
```

Is rewritten into:

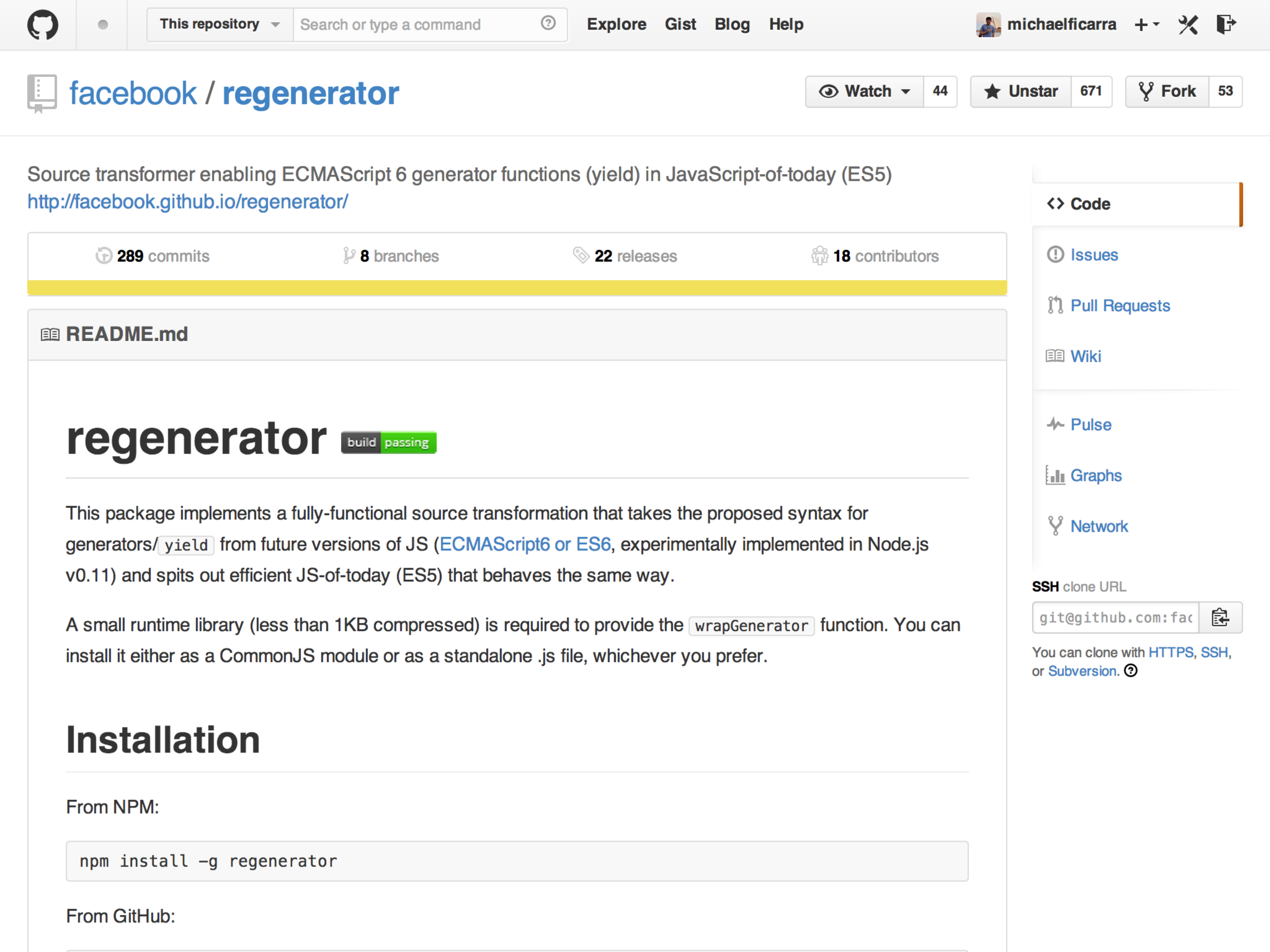


Brushtail

JS AST rewriter for tail call elimination. Try it below or get it from GitHub.

```
function count(from, to) {
    if(from >= to)
        return from;

    return count(from + 1, to);
}

console.log(count(0, 1000000));
```

```
function count(from, to) {
    var __tcor;
    tco:
        while (true) {
            if (from >= to) {
                __tcor = from;
                break tco;
            }
            {
                var __from = from + 1, __to = to;
                from = __from;
                to = __to;
                continue tco;
            }
        }
    return __tcor;
}
console.log(count(0, 1000000));
```



facebook / regenerator

Source transformer enabling ECMAScript 6 generator functions (yield) in JavaScript-of-today (ES5)
http://facebook.github.io/regenerator/

289 commits | 8 branches | 22 releases | 18 contributors

README.md

regenerator

This package implements a fully-functional source transformation that takes the proposed syntax for generators/ `yield` from future versions of JS (ECMAScript6 or ES6, experimentally implemented in Node.js v0.11) and spits out efficient JS-of-today (ES5) that behaves the same way.

A small runtime library (less than 1KB compressed) is required to provide the `wrapGenerator` function. You can install it either as a CommonJS module or as a standalone .js file, whichever you prefer.

Installation

From NPM:

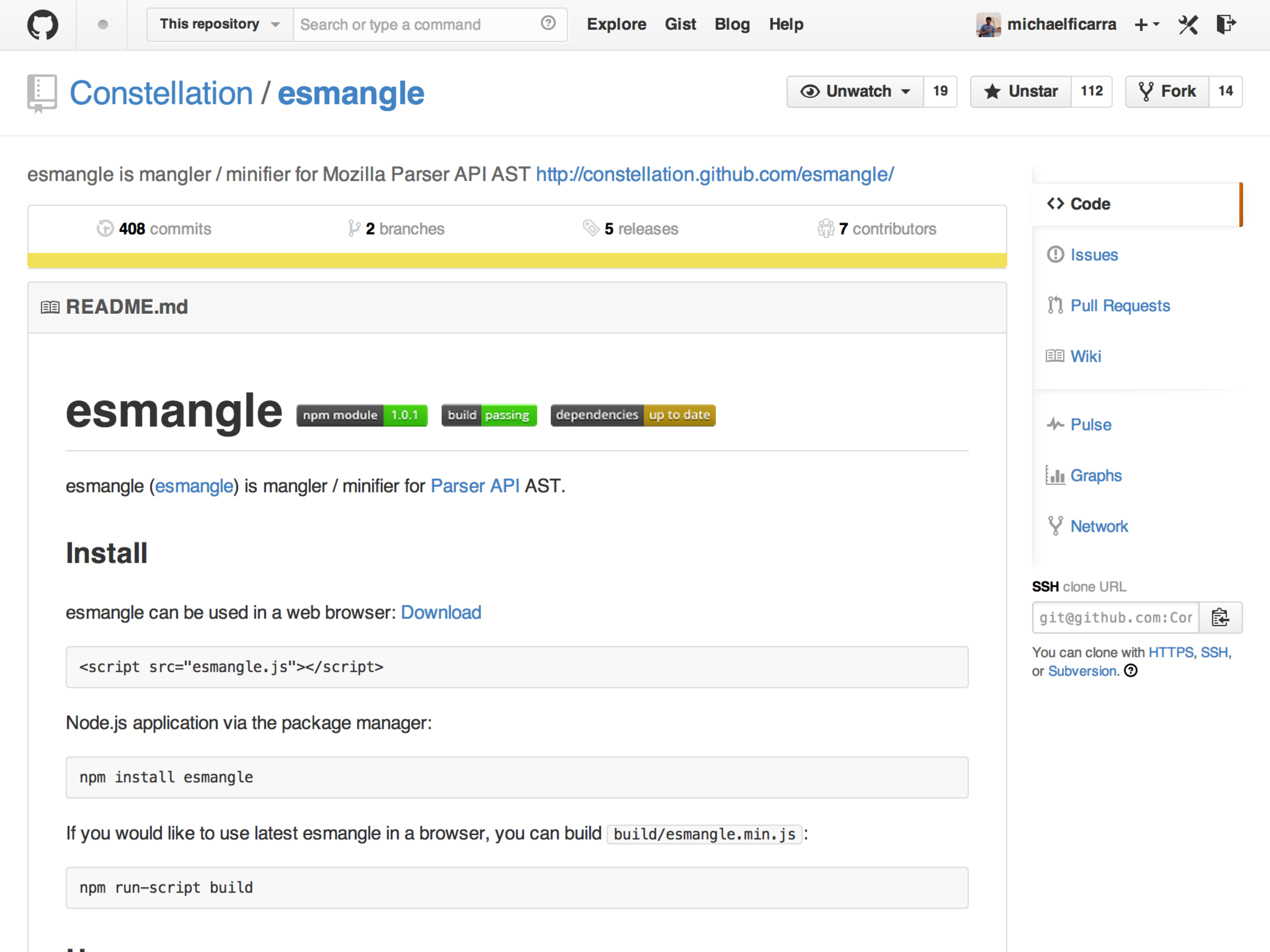
```
npm install -g regenerator
```

From GitHub:

SSH clone URL

`git@github.com:fac`

You can clone with HTTPS, SSH, or Subversion.



This repository Search or type a command Explore Gist Blog Help michaelficarra

Constellation / esmangle

Unwatch 19 Unstar 112 Fork 14

esmangle is mangler / minifier for Mozilla Parser API AST http://constellation.github.com/esmangle/

408 commits 2 branches 5 releases 7 contributors

Code
Issues
Pull Requests
Wiki
Pulse
Graphs
Network

SSH clone URL
git@github.com:Cor
You can clone with HTTPS, SSH, or Subversion.

README.md

esmangle npm module 1.0.1 build passing dependencies up to date

esmangle (esmangle) is mangler / minifier for Parser API AST.

Install

esmangle can be used in a web browser: Download

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

Node.js application via the package manager:

```
npm install esmangle
```

If you would like to use latest esmangle in a browser, you can build `build/esmangle.min.js`:

```
npm run-script build
```

Simplification Phase

```
{ type: "CallExpression"
, callee: {type: "Identifier", name: "f"}
, arguments:
  [
    { type: "UnaryExpression"
    , prefix: true, operator: "!"
    , argument:
      { type: "UnaryExpression"
      , prefix: true, operator: "!"
      , argument:
        { type: "UnaryExpression"
        , prefix: true, operator: "!"
        , argument: {type: "Identifier", name: "a"}
        } } }
  ]
}
```

```
f(!!!a)
```

$$\Downarrow$$

```
f(!a)
```

Simplification Phase

```
{ type: "CallExpression"
, callee: {type: "Identifier", name: "f"}
, arguments:
  [
    { type: "UnaryExpression"
    , prefix: true, operator: "!"
    , argument: {type: "Identifier", name: "a"}
    }
  ]
}
```

```
f(!!!a)
```

$$\Big\downarrow$$

```
f(!a)
```

Expansion Phase

```
{ type: "MemberExpression"
, computed: false
, object: {type: "Identifier", name: "a"}
, property: {type: "Identifier", name: "Infinity"}
}
```

$$\begin{tikzcd}\texttt{a.Infinity} \arrow[d] \\ \texttt{a[1/0]}\end{tikzcd}$$

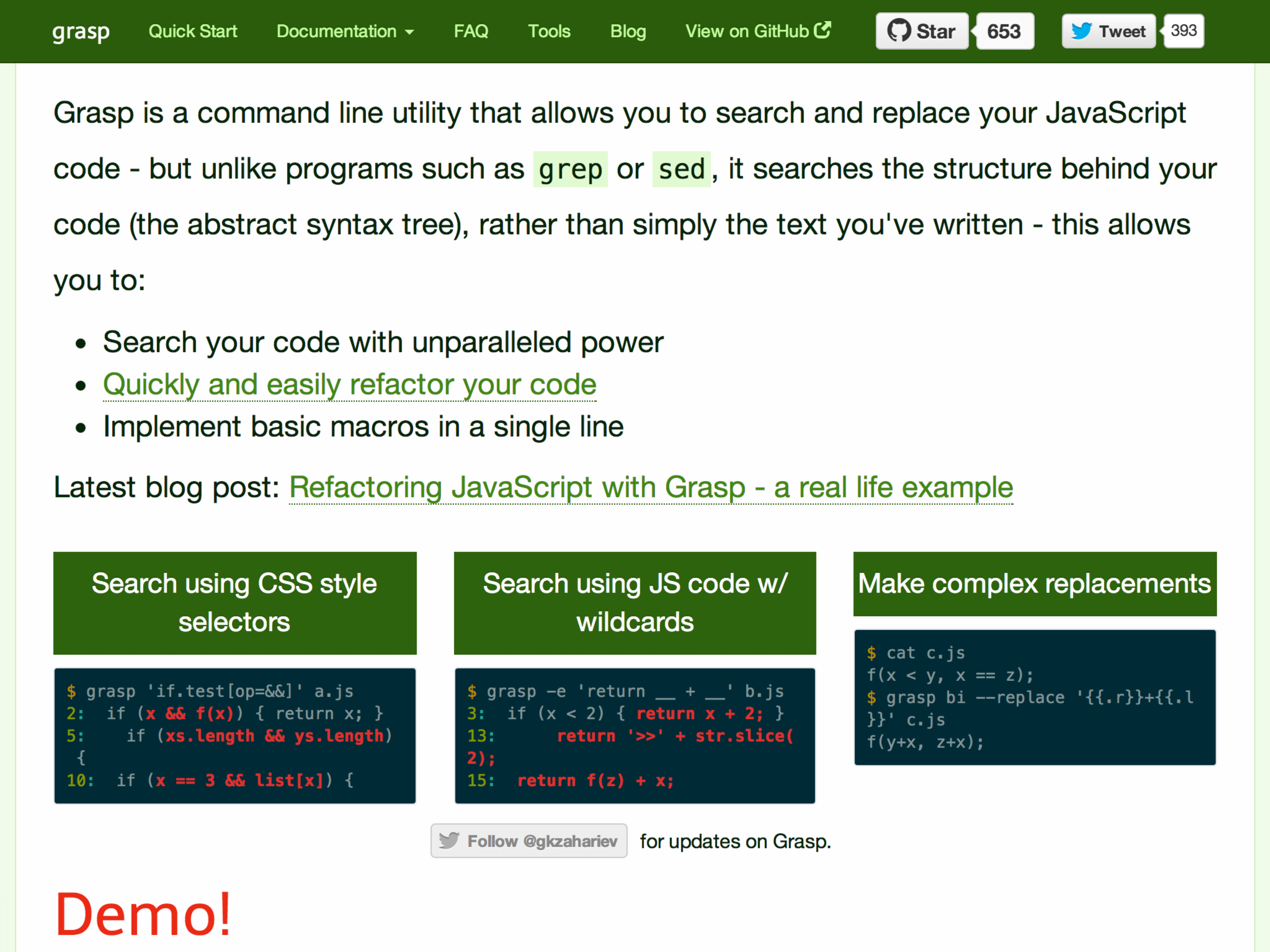
Expansion Phase

```
{ type: "MemberExpression"
, computed: true
, object: {type: "Identifier", name: "a"}
, property:
  { type: "BinaryExpression"
  , operator: "/"
  , left: {type: "Literal", value: 1, raw: "1"}
  , right: {type: "Literal", value: 0, raw: "0" }
  }
}
```

```
a.Infinity
```

$\Downarrow$

```
a[1/0]
```



grasp Quick Start Documentation ▾ FAQ Tools Blog View on GitHub Star 653 Tweet 393

Grasp is a command line utility that allows you to search and replace your JavaScript code - but unlike programs such as `grep` or `sed`, it searches the structure behind your code (the abstract syntax tree), rather than simply the text you've written - this allows you to:

- Search your code with unparalleled power
- Quickly and easily refactor your code
- Implement basic macros in a single line

Latest blog post: Refactoring JavaScript with Grasp - a real life example

Search using CSS style selectors

```
$ grasp 'if.test[op=&&]' a.js
2:  if (x && f(x)) { return x; }
5:    if (xs.length && ys.length)
 {
10:  if (x == 3 && list[x]) {
```

Search using JS code w/ wildcards

```
$ grasp -e 'return __ + __' b.js
3:  if (x < 2) { return x + 2; }
13:       return '>>' + str.slice(
2);
15:  return f(z) + x;
```

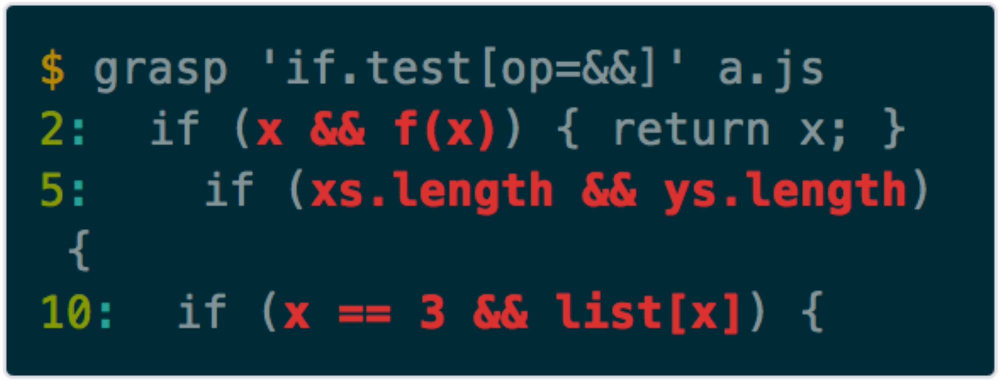
Make complex replacements

```
$ cat c.js
f(x < y, x == z);
$ grasp bi --replace '{{.r}}+{{.l
}}' c.js
f(y+x, z+x);
```

Follow @gkzahariev for updates on Grasp.

Demo!

Search using CSS style selectors



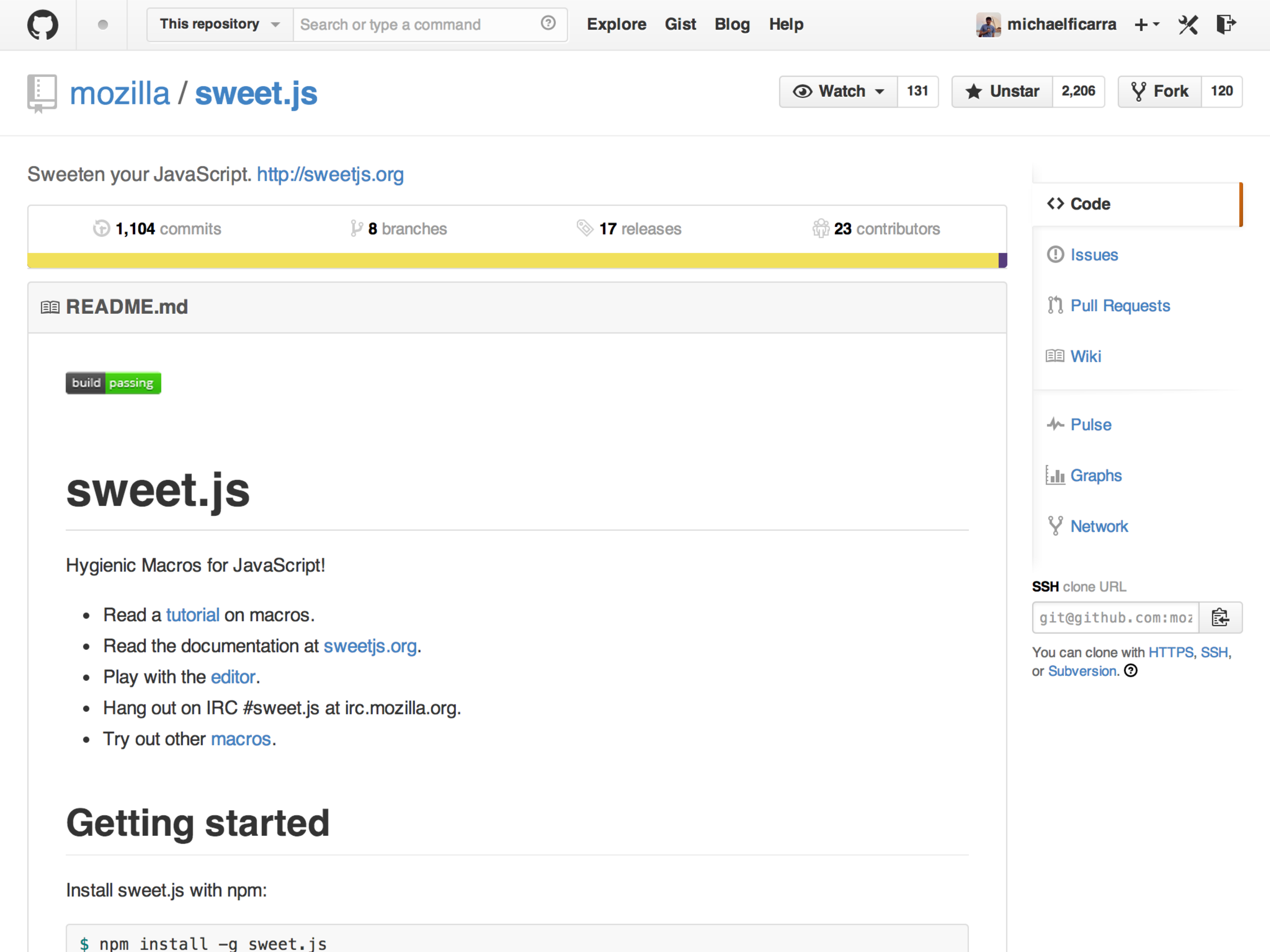
```
$ grasp 'if.test[op=&&]' a.js
2:  if (x && f(x)) { return x; }
5:    if (xs.length && ys.length)
 {
10:  if (x == 3 && list[x]) {
```

Search using JS code w/ wildcards

```
$ grasp -e 'return __ + __' b.js
3:  if (x < 2) { return x + 2; }
13:      return '>>' + str.slice(
2);
15:  return f(z) + x;
```

Make complex replacements

```
$ cat c.js
f(x < y, x == z);
$ grasp bi --replace '{{.r}}+{{.l
}}' c.js
f(y+x, z+x);
```



mozilla / sweet.js

Sweeten your JavaScript. http://sweetjs.org

SSH clone URL

`git@github.com:moz`

You can clone with HTTPS, SSH, or Subversion.

README.md

sweet.js

Hygienic Macros for JavaScript!

- Read a tutorial on macros.
- Read the documentation at sweetjs.org.
- Play with the editor.
- Hang out on IRC #sweet.js at irc.mozilla.org.
- Try out other macros.

Getting started

Install sweet.js with npm:

```
$ npm install -g sweet.js
```

SWEETEN YOUR JAVASCRIPT

Sweet.js brings hygienic macros from languages like Scheme and Rust to JavaScript. Macros allow you to sweeten the syntax of JavaScript and craft the language you've always wanted.

Wish the `function` keyword in JavaScript wasn't so long? What if you could define functions with `def` instead?

Macros let you do this!

```
macro def {
  case $name:ident $params $body => {
    function $name $params $body
  }
}

def sweet(a) {
  console.log("Macros are sweet!");
}

```

Try it!

```
function sweet(a$2) {
    console.log('Macros are sweet!');
}
```

Want a better way to make "classy" objects?

Macros can do that too!

```
macro class {
```

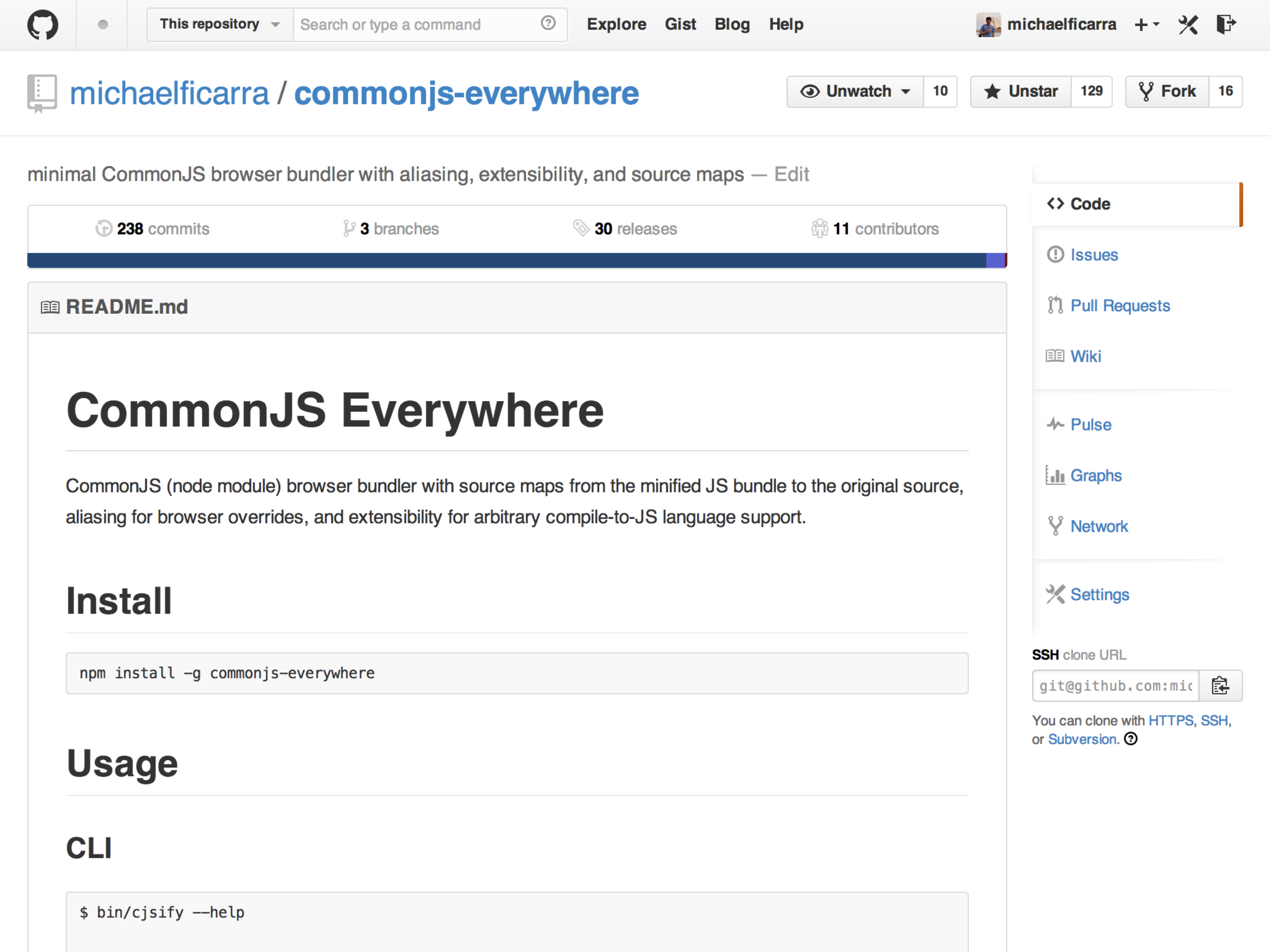
Want a better way to make "classy" objects?

Macros can do that too!

```
macro class {
  case $className:ident {
    constructor $constParam $constBody
    $($methodName:ident $methodParam $methodBody) ... } => {

    function $className $constParam $constBody

    $($className.prototype.$methodName
      = function $methodName $methodParam $methodBody; ) ...

  }
}

class Person {
  constructor(name) {
    this.name = name;
  }

  say(msg) {
    console.log(this.name + " says: " + msg);
  }
}
var bob = new Person("Bob");
bob.say("Macros are sweet!");

```

Try it!

```
function Person(name$6) {
    this.name = name$6;
}
Person.prototype.say = function say(msg$8) {
    console.log(this.name + ' says: ' + msg$8);
};
var bob$10 = new Person('Bob');
bob$10.say('Macros are sweet!');
```



michaelficarra / commonjs-everywhere

minimal CommonJS browser bundler with aliasing, extensibility, and source maps — Edit

SSH clone URL

git@github.com:mic

You can clone with HTTPS, SSH, or Subversion.

README.md

CommonJS Everywhere

CommonJS (node module) browser bundler with source maps from the minified JS bundle to the original source, aliasing for browser overrides, and extensibility for arbitrary compile-to-JS language support.

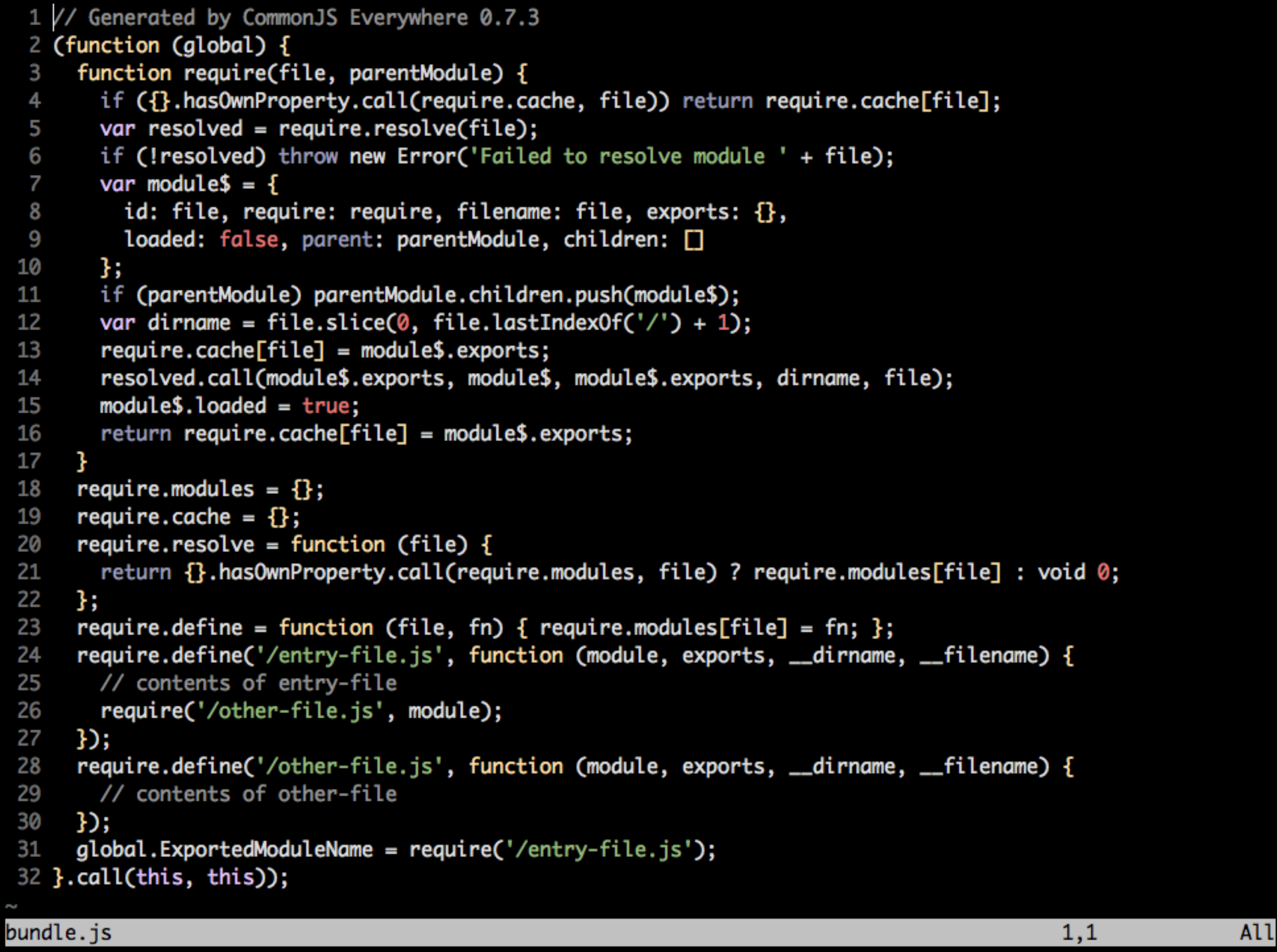
Install

```
npm install -g commonjs-everywhere
```

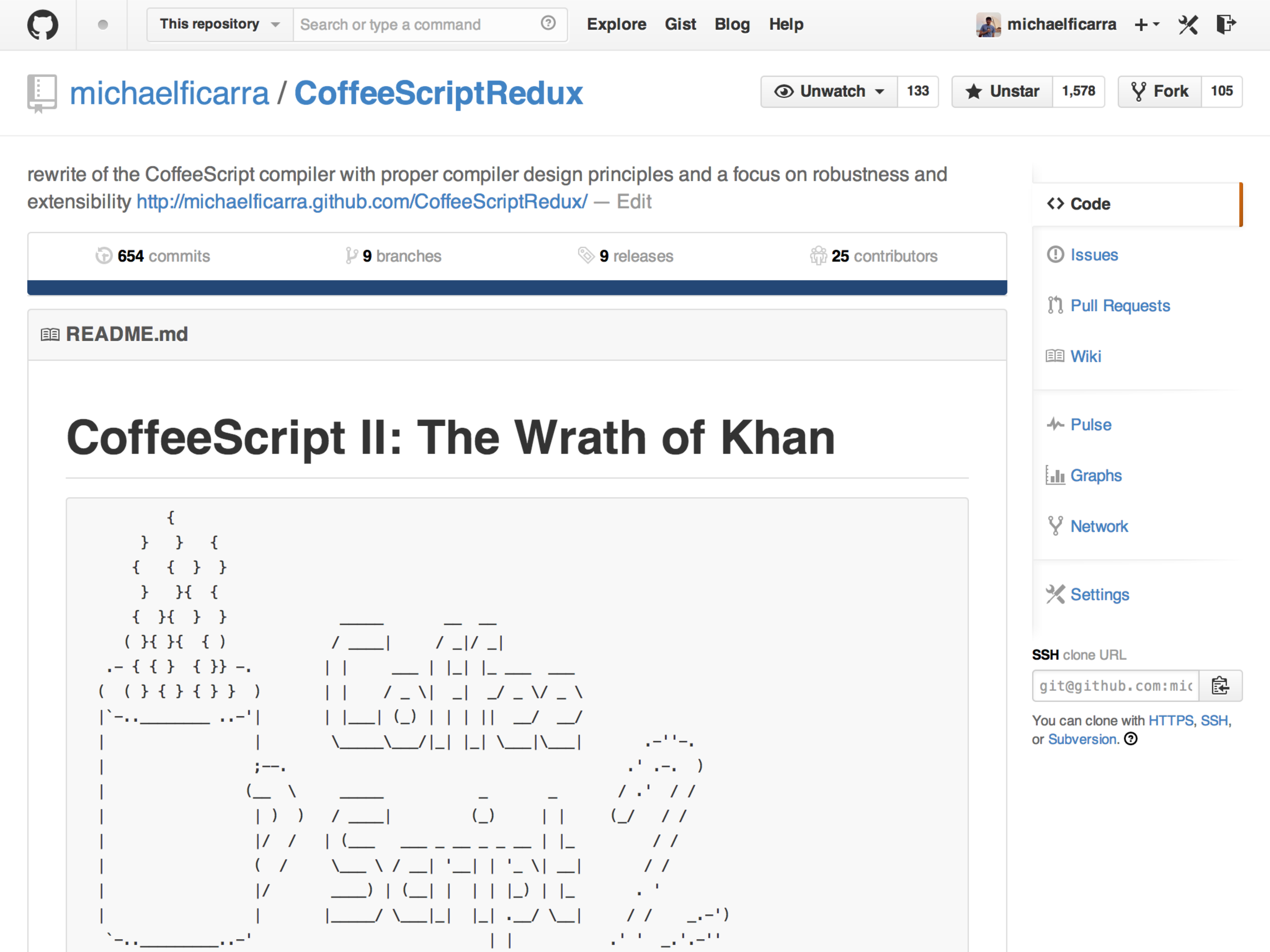
Usage

CLI

```
$ bin/cjsify --help
```



```
// Generated by CommonJS Everywhere 0.7.3
(function (global) {
  function require(file, parentModule) {
    if ({}.hasOwnProperty.call(require.cache, file)) return require.cache[file];
    var resolved = require.resolve(file);
    if (!resolved) throw new Error('Failed to resolve module ' + file);
    var module$ = {
      id: file, require: require, filename: file, exports: {},
      loaded: false, parent: parentModule, children: []
    };
    if (parentModule) parentModule.children.push(module$);
    var dirname = file.slice(0, file.lastIndexOf('/') + 1);
    require.cache[file] = module$.exports;
    resolved.call(module$.exports, module$, module$.exports, dirname, file);
    module$.loaded = true;
    return require.cache[file] = module$.exports;
  }
  require.modules = {};
  require.cache = {};
  require.resolve = function (file) {
    return {}.hasOwnProperty.call(require.modules, file) ? require.modules[file] : void 0;
  };
  require.define = function (file, fn) { require.modules[file] = fn; };
  require.define('/entry-file.js', function (module, exports, __dirname, __filename) {
    // contents of entry-file
    require('/other-file.js', module);
  });
  require.define('/other-file.js', function (module, exports, __dirname, __filename) {
    // contents of other-file
  });
  global.ExportedModuleName = require('/entry-file.js');
}.call(this, this));
~
bundle.js                                                              1,1           All
```



michaelficarra / CoffeeScriptRedux

rewrite of the CoffeeScript compiler with proper compiler design principles and a focus on robustness and extensibility http://michaelficarra.github.com/CoffeeScriptRedux/ — Edit

654 commits | 9 branches | 9 releases | 25 contributors

SSH clone URL

`git@github.com:mic`

You can clone with HTTPS, SSH, or Subversion.

README.md

CoffeeScript II: The Wrath of Khan

```
          {
       }   }   {
      {   {  }  }
       }   }{  {
      {  }{  }  }              _____       __  __
     ( }{ }{  { )             / ____|     / _|/ _|
   .- { { }  { }} -.         | |     ___ | |_| |_ ___  ___
  (  ( } { } { } }  )        | |    / _ \|  _|  _/ _ \/ _ \
  |`-..________ ..-'|        | |___| (_) | | | ||  __/  __/
  |                 |         \_____\___/|_| |_| \___|\___|          .-''-.
  |                 ;--.                                          .' .-.  )
  |                (__  \       _____           _       _        / .'  / /
  |                 | )  )    / ____|         (_)     | |       (_/   / /
  |                 |/  /    | (___   ___ _ __ _ _ __ | |_           / /
  |                 (  /      \___ \ / __| '__| | '_ \| __|         / /
  |                 |/        ____) | (__| |  | | |_) | |_         . '
  |                 |        |_____/ \___|_|  |_| .__/ \__|       / /    _.-')
   `-.._________..-'                              | |            .' '  _.'.-''
```

Parser API

by 22 contributors:

Recent builds of the standalone SpiderMonkey shell include a reflection of the SpiderMonkey parser, made available as a JavaScript API. This makes it easier to write tools in JavaScript that manipulate JavaScript source programs, such as syntax highlighters, static analyses, translators, compilers, obfuscators, etc.

NOTE: Several projects are using this specification. Please do not make changes to it without consulting with the authors of Esprima, Escodegen, and Acorn.

Example:

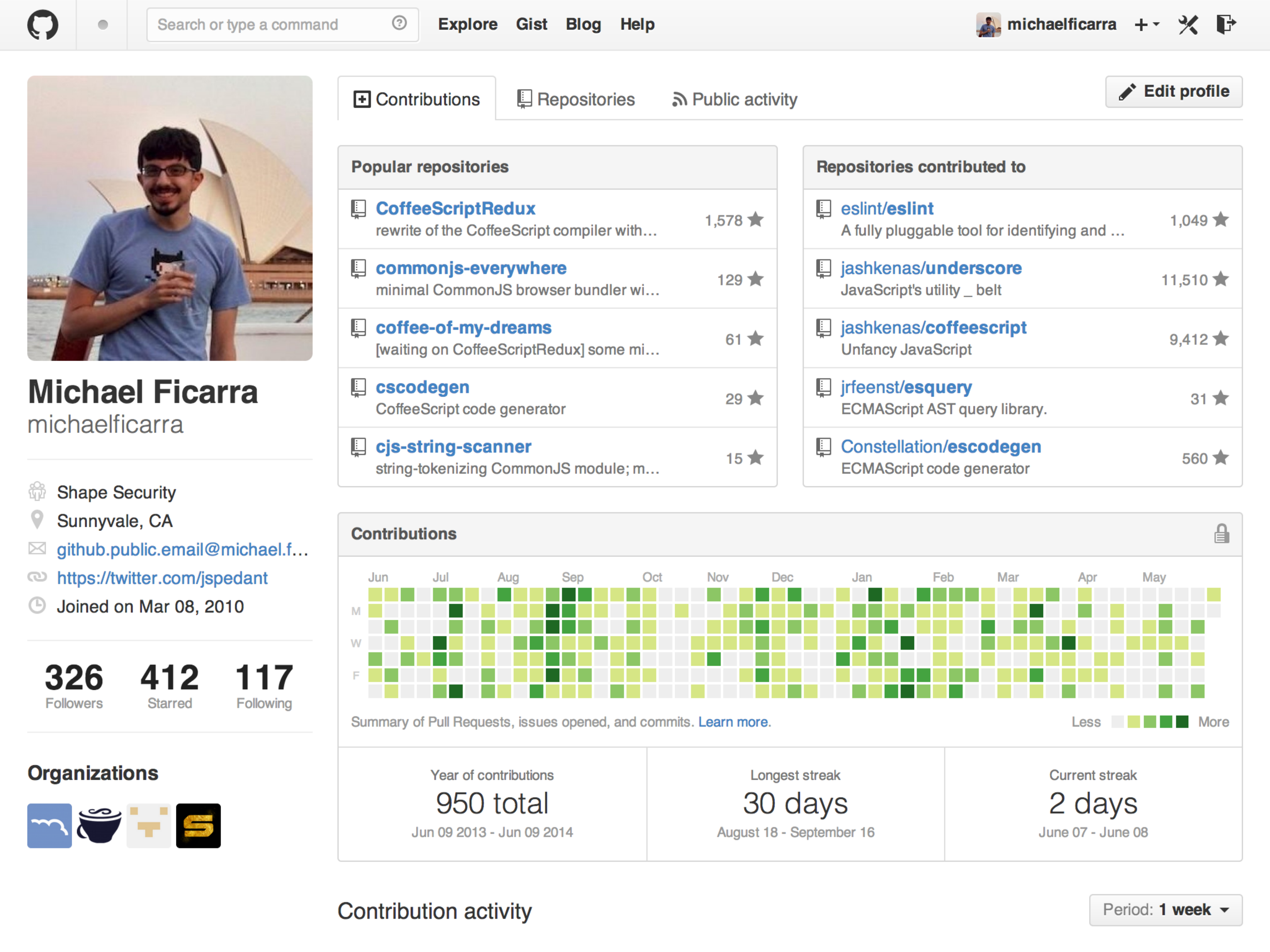
```
> var expr = Reflect.parse("obj.foo + 42").body[0].expression
> expr.left.property
({loc:null, type:"Identifier", name:"foo"})
> expr.right
({loc:{source:null, start:{line:1, column:10}, end:{line:1, column:12}}, type:"Literal", value:42})
```

It is also available since Firefox 7; it can be imported into the global object via:

```
Components.utils.import("resource://gre/modules/reflect.jsm")
```

Future Work

- Standard CST
- Standard ASG



Search or type a command | Explore | Gist | Blog | Help | michaelficarra

Contributions | Repositories | Public activity | Edit profile

Michael Ficarra
michaelficarra

- Shape Security
- Sunnyvale, CA
- github.public.email@michael.f…
- https://twitter.com/jspedant
- Joined on Mar 08, 2010

326	412	117
Followers	Starred	Following

Organizations

Popular repositories

Repository	Description	Stars
CoffeeScriptRedux	rewrite of the CoffeeScript compiler with…	1,578
commonjs-everywhere	minimal CommonJS browser bundler wi…	129
coffee-of-my-dreams	[waiting on CoffeeScriptRedux] some mi…	61
cscodegen	CoffeeScript code generator	29
cjs-string-scanner	string-tokenizing CommonJS module; m…	15

Repositories contributed to

Repository	Description	Stars
eslint/eslint	A fully pluggable tool for identifying and …	1,049
jashkenas/underscore	JavaScript's utility _ belt	11,510
jashkenas/coffeescript	Unfancy JavaScript	9,412
jrfeenst/esquery	ECMAScript AST query library.	31
Constellation/escodegen	ECMAScript code generator	560

Contributions

Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr May

M W F

Summary of Pull Requests, issues opened, and commits. Learn more.

Less ⬜ 🟩 🟩 🟩 🟩 More

Year of contributions	Longest streak	Current streak
950 total	30 days	2 days
Jun 09 2013 - Jun 09 2014	August 18 - September 16	June 07 - June 08

Contribution activity

Period: **1 week**