# Jasmine

The first steps to *finally* test-driving your JavaScript

You can find this presentation & code at:

http://v.gd/jasmine_intro

# Who?



You!

# What?



Jasmine!

# Where?




http://pivotal.github.com/jasmine

# When?

As soon as you’re ready to jump in

# Why?



Unfortunately, not enough time to cover the “why TDD my JavaScript?” question today

# More on Why:

http://v.gd/javascript



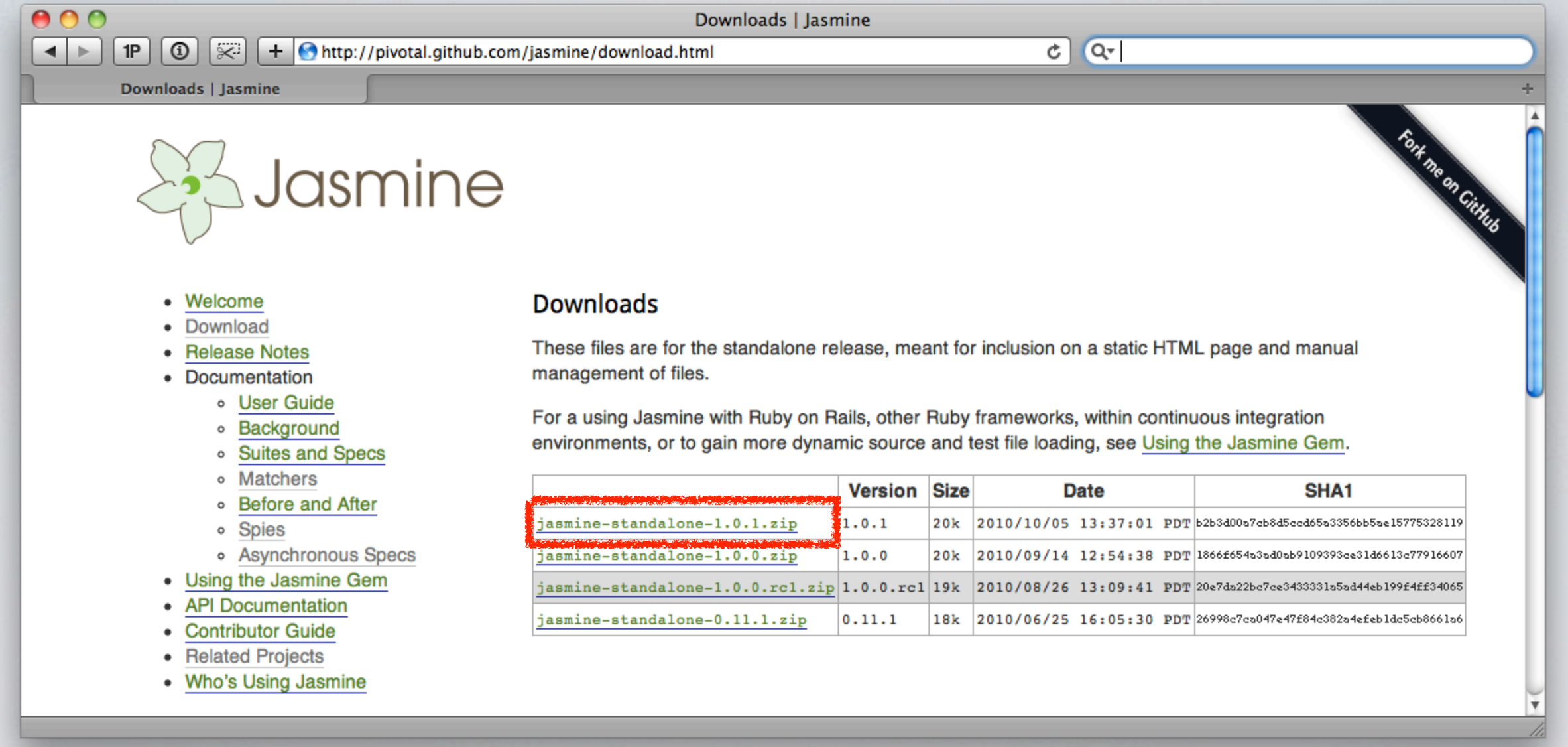
However, I did recently give a talk about why JavaScript is worthy of TDD.

iWork slides:
http://public.iwork.com/document/?a=p94397438&d=JavaScript_CraftsmanShip__38__TDD_-_Searls_9_47_13_47_2010.key

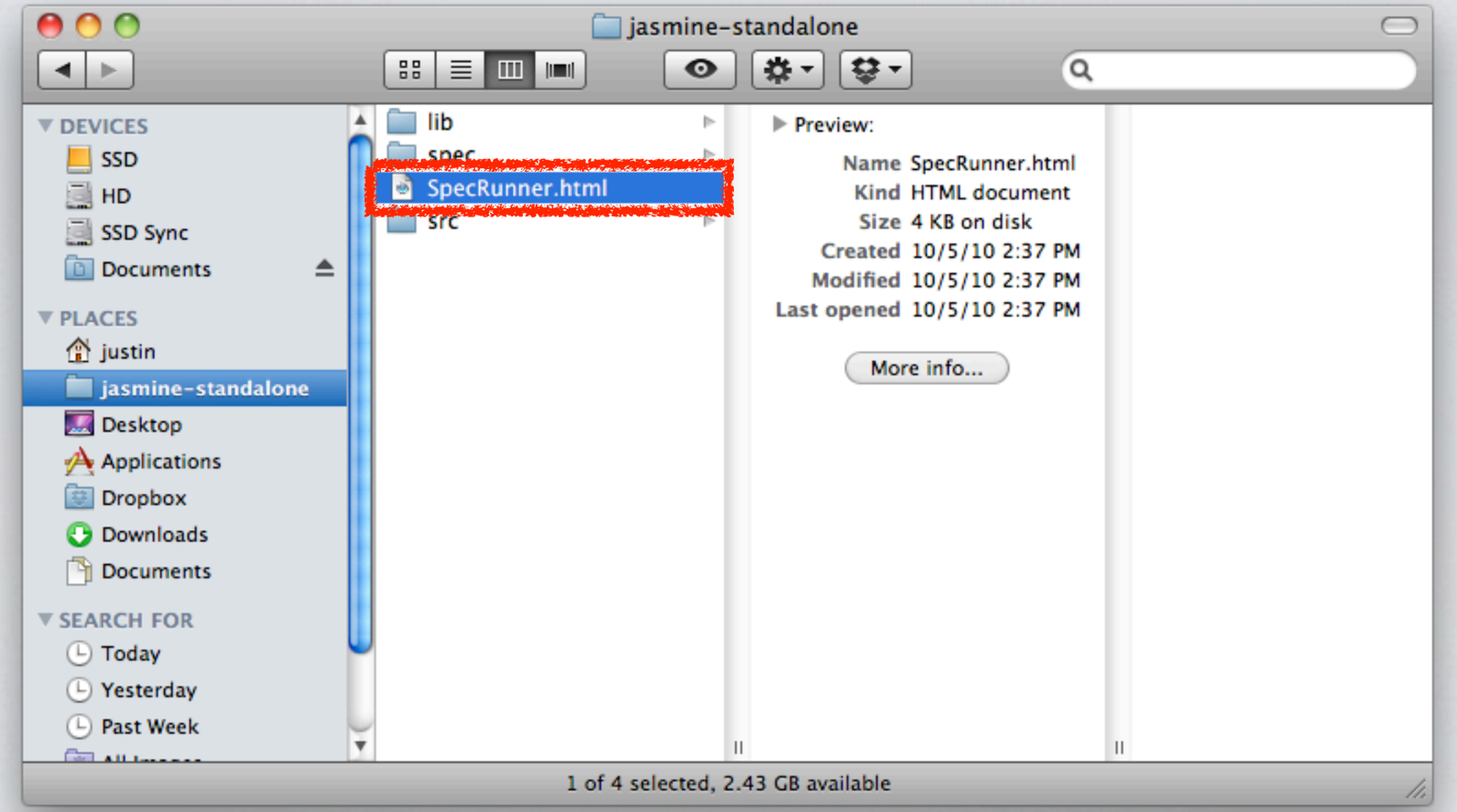
Slideshare:
http://www.slideshare.net/searls/javascript-craftsmanship-why-javascript-is-worthy-of-tdd

# How?

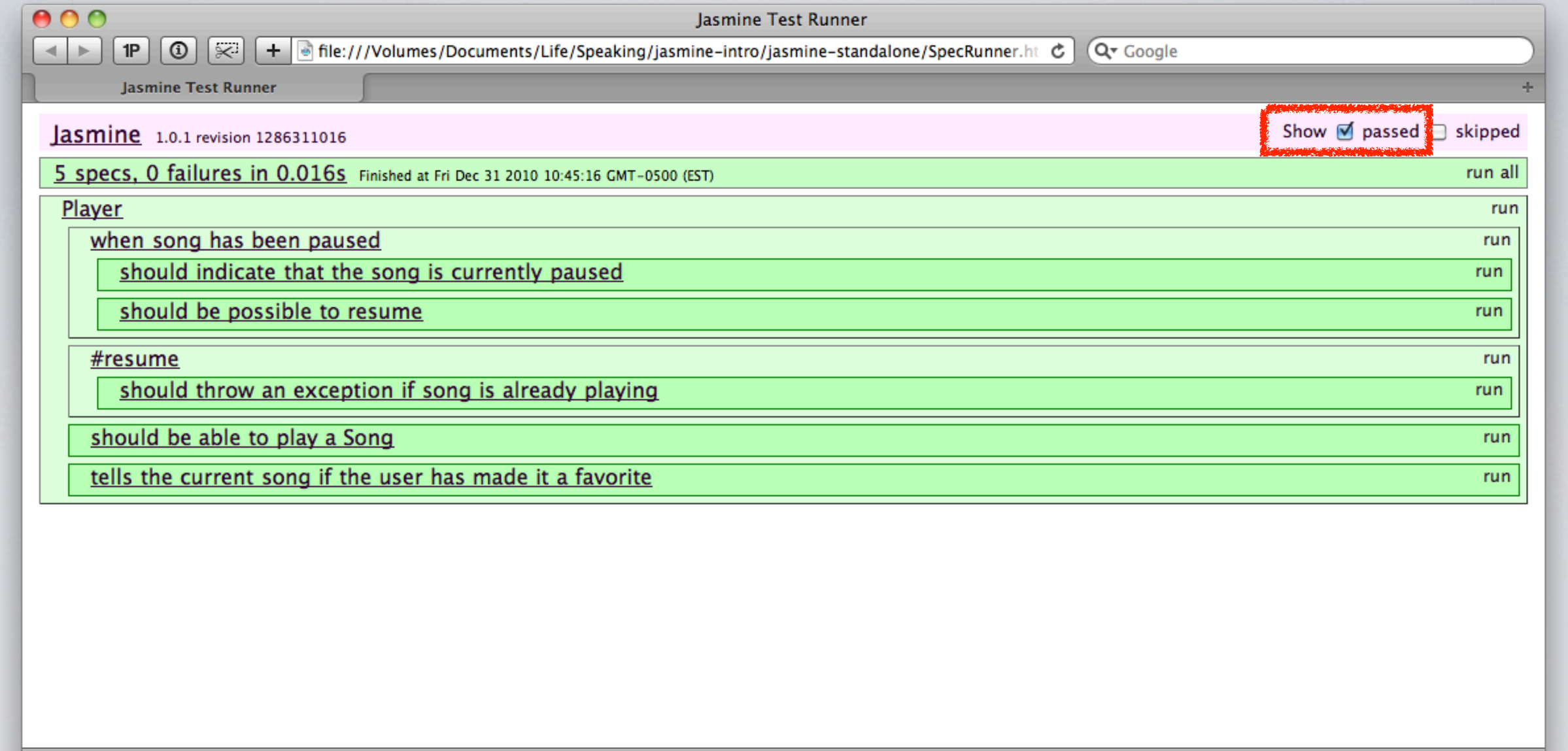
# 1. Get Jasmine




# For starters, download the latest standalone




# Open `SpecRunner.html` in a browser




Check "passed" to see the example specs

# 2. Start Spec'ing!

PlayerSpec.js — jasmine-standalone

PlayerSpec.js

```
describe("Player", function() {
  //...
});
```

Line: 2 Column: 5 JavaScript Soft Tabs: 2

# Let's start fresh!



This is a top-level example group; most spec files are arranged 1-to-1 with the code in the files they specify so they can be easily located and contain a single outer-most example group specified by invoking the `describe` function.

PlayerSpec.js — jasmine-standalone

PlayerSpec.js

```
describe("Player", function() {
  it("is defined");
});
```

Line: 2 Column: 20 JavaScript Soft Tabs: 2

# Pending examples



A pending example specifies something we know we want our code to do, but that we may not be implementing quite yet. If you have an existing expectation that you need to make pending, you can prepend it with an ‘x’, a la:

```
xit(“description”,function(){
  expect(code).toBePerfect();
});
```

PlayerSpec.js — jasmine-standalone

PlayerSpec.js

```
describe("Player", function() {
  it("is defined",function(){
    expect(Player).toBeDefined();
  });
});
```

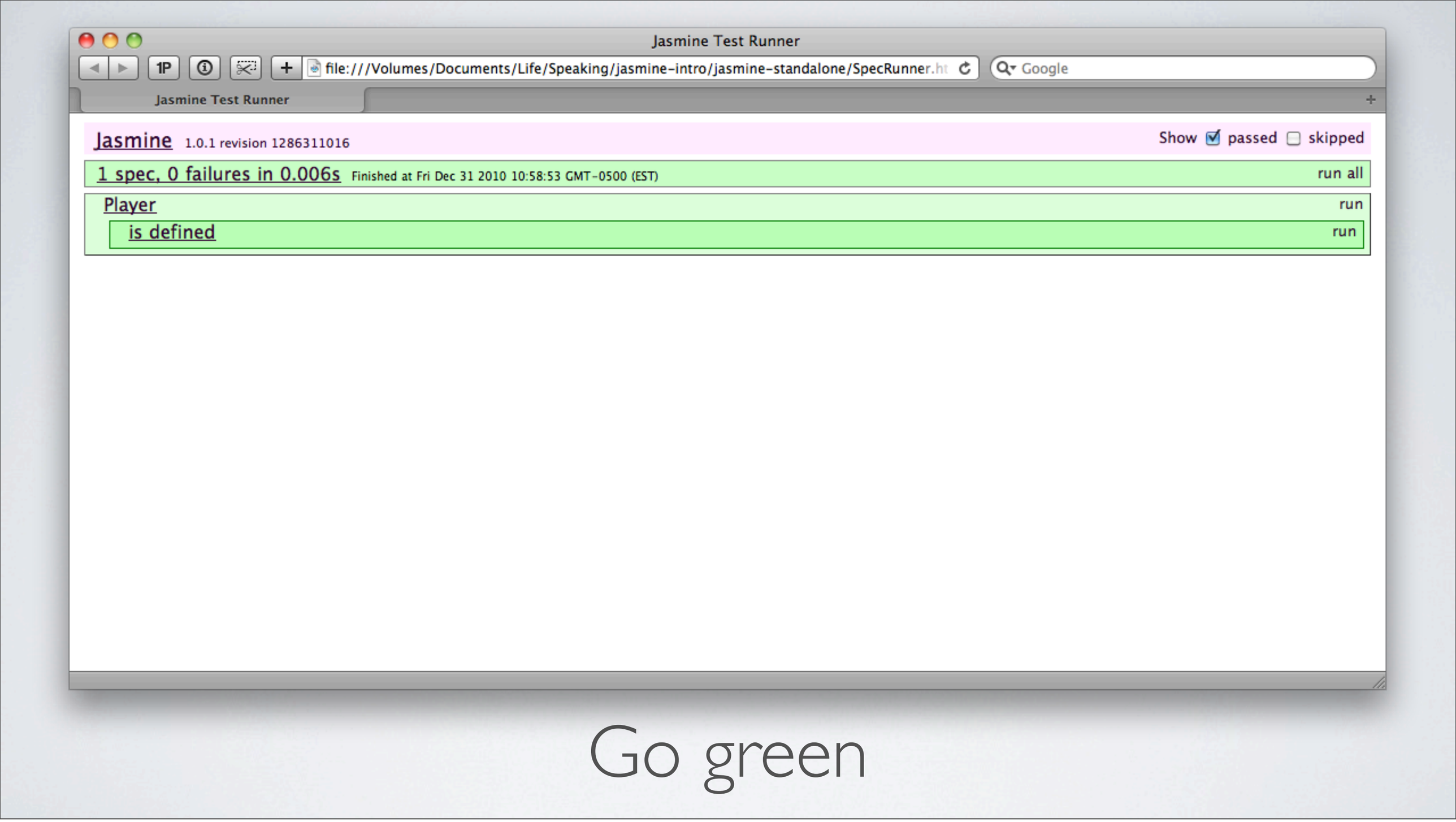
Line: 2 Column: 18 JavaScript Soft Tabs: 2

# Great expectations



You can find all of Jasmine's default matchers here: http://pivotal.github.com/jasmine/matchers.html

If you're using jQuery, you can there's a great collection of handy jQuery-specific matchers too: https://github.com/velesin/jasmine-jquery




# Go green



Simply refresh the page to re-run your specs.

A fun trick is to add a meta element to your spec runner's head to refresh the page every given interval to achieve a "poor man's CI": <meta http-equiv="refresh" content="3">

PlayerSpec.js — jasmine-standalone

SpecRunner.html | PlayerSpec.js

```
describe("Player", function() {
  var player;

});
```

Line: 3 Column: 1 | JavaScript | Soft Tabs: 2

# Nesting behavior for DRY, readable specs



You can nest example groups to specify cascading behavior. This can be really powerful at DRYing up your tests while retaining readability. I tend to introduce a new nested example group for each subsequent state change of my SUT that I need to specify. So that each nested example group can inherit the state of the parent, I often find myself performing arrange & act in a beforeEach() function and keep each it() focused on a single expectation regarding the behavior being specified.

PlayerSpec.js — jasmine-standalone

SpecRunner.html | PlayerSpec.js

```
describe("Player", function() {
  var player;
  beforeEach(function(){
    player = new Player();
    player.play();
  });

});
```

Line: 3 Column: 1 | JavaScript | Soft Tabs: 2

# Nesting behavior for DRY, readable specs

PlayerSpec.js — jasmine-standalone

SpecRunner.html | PlayerSpec.js

```
describe("Player", function() {
  var player;
  beforeEach(function(){
    player = new Player();
    player.play();
  });
  it('is playing',function(){
    expect(player.isPlaying).toBeTruthy();
  })
});
```

Line: 7 Column: 1 | JavaScript | Soft Tabs: 2 | Player

# Nesting behavior for DRY, readable specs

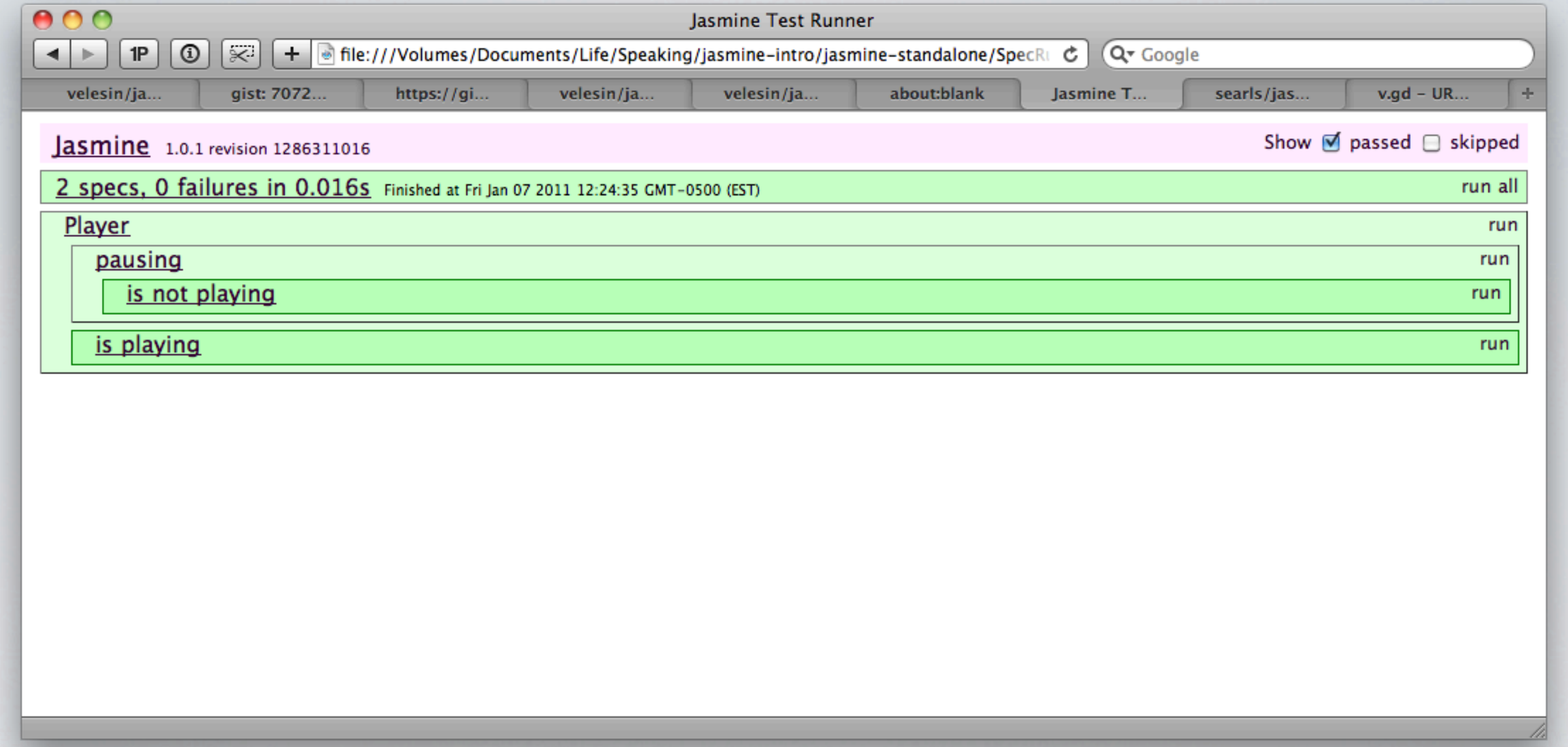
PlayerSpec.js — jasmine-standalone

SpecRunner.html | PlayerSpec.js

```
describe("Player", function() {
  var player;
  beforeEach(function(){
    player = new Player();
    player.play();
  });
  it('is playing',function(){
    expect(player.isPlaying).toBeTruthy();
  })
  describe('pausing',function(){
    beforeEach(function(){ player.pause(); });
    it('is not playing',function(){
      expect(player.isPlaying).not.toBe(true);
    })
  });
});
```

Line: 10 Column: 1 | JavaScript | Soft Tabs: 2 | Player

# Nesting behavior for DRY, readable specs


Jasmine Test Runner
file:///Volumes/Documents/Life/Speaking/jasmine-intro/jasmine-standalone/SpecR
Google
velesin/ja...
gist: 7072...
https://gi...
velesin/ja...
velesin/ja...
about:blank
Jasmine T...
searls/jas...
v.gd - UR...
Jasmine 1.0.1 revision 1286311016
Show passed skipped
2 specs, 0 failures in 0.016s Finished at Fri Jan 07 2011 12:24:35 GMT-0500 (EST)
run all
Player
run
pausing
run
is not playing
run
is playing
run

# Go green

# 3. Custom Matchers

```
expect(player.isPlaying).not.toBe(true);
```



Looking back on it, that matcher is a bit tough to read. Custom matchers can be great for refactoring expectations that are too complex, will be frequently duplicated, or could do a better job of revealing their intent.

SpecHelper.js — jasmine-standalone

SpecHelper.js

```
beforeEach(function() {
  this.addMatchers({
    toBePlaying: function(expectedSong) {
      var player = this.actual;
      return player.currentlyPlayingSong === expectedSong
          && player.isPlaying;
    }
  })
});
```

Line: 1 Column: 1 JavaScript Soft Tabs: 2

# Adding a custom matcher



It's as simple as adding an object literal of named matchers as above. You can even customize the message that's printed on expectation failures by providing a this.message function inside the matcher that returns a string.

PlayerSpec.js — jasmine-standalone

PlayerSpec.js

```
describe('pausing',function(){
  beforeEach(function(){ player.pause(); });
  it('is not playing',function(){
    expect(player).not.toBePlaying();
  })
});
```

Line: 13 Column: 27 JavaScript Soft Tabs: 2 Player

# Using a custom matcher



And any expectation can be inverted with a `.not` prefix, without any custom work on the matcher's part

```
expect(player.isPlaying).not.toBe(true);
```

```
expect(player).not.toBePlaying();
```

# 4. Spies

# Specify a behavior in isolation



Let’s say that when the jukebox starts playing, a loading indicator needs to be shown.

But because the loading indicator feature is covered by specs elsewhere and animating the real thing would slow down our test execution dramatically, we want to specify only that we invoke it (isolating ourselves from what the internal logic of showing and hiding the loading indicator).

PlayerSpec.js — jasmine-standalone

PlayerSpec.js | SpecRunner.html | SpecHelper.js | SongSpec.js | Song.js

```
describe("Player", function() {
  var player;
  beforeEach(function(){
    player = new Player();
    spyOn(loading,'show');
    player.play();
  });

});
```

Line: 8 Column: 1 | JavaScript | Soft Tabs: 2 | Player

# Tell Jasmine to spy on loading.show()

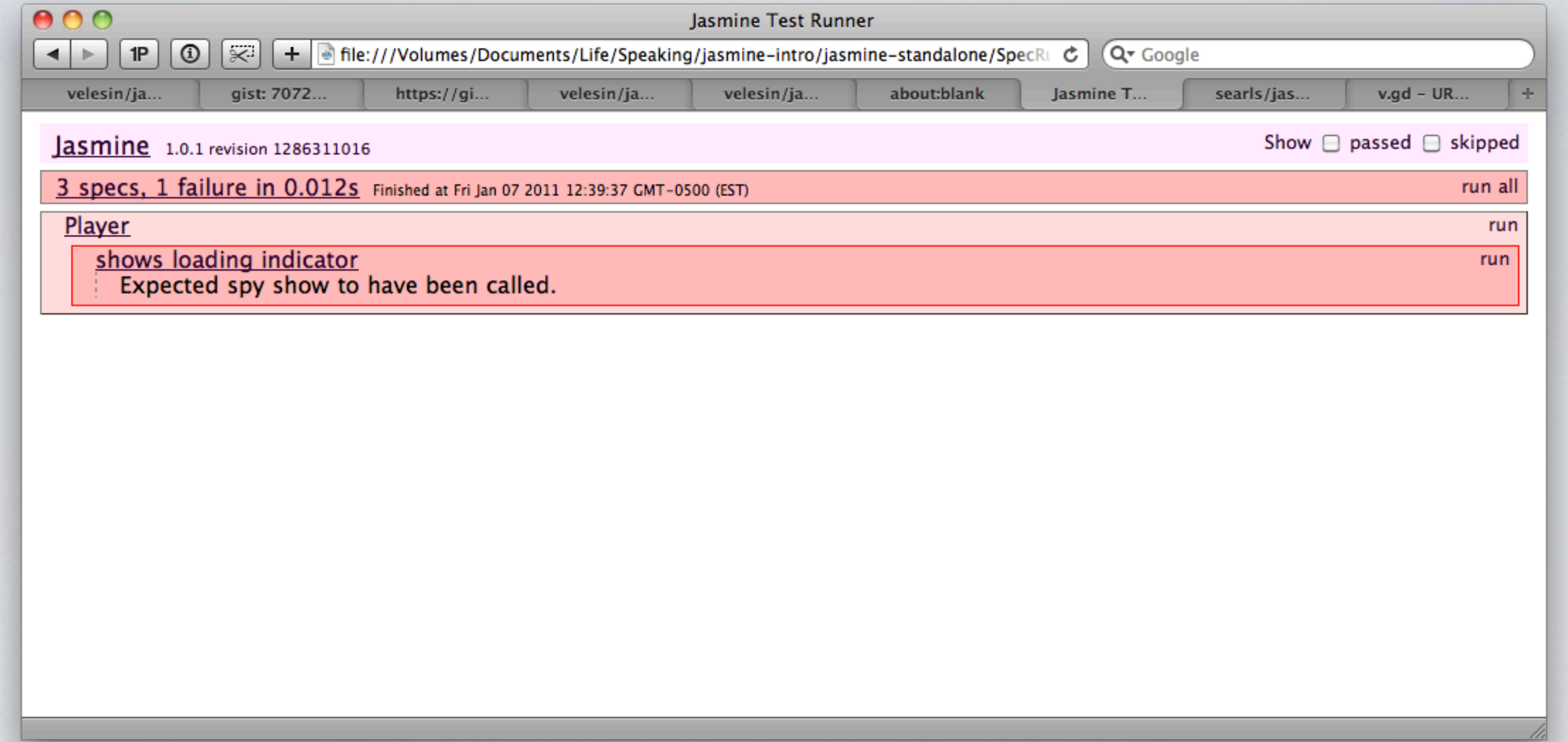
PlayerSpec.js — jasmine-standalone

PlayerSpec.js | Player.js | SpecRunner.html | SpecHelper.js | SongSpec.js | Song.js

```
describe("Player", function() {
  var player;
  beforeEach(function(){
    player = new Player();
    spyOn(loading,'show');
    player.play();
  });
  it('shows loading indicator',function(){
    expect(loading.show).toHaveBeenCalled();
});
```

Line: 10 Column: 6 | JavaScript | Soft Tabs: 2 | Player

# Expect the invocation


Jasmine Test Runner
file:///Volumes/Documents/Life/Speaking/jasmine-intro/jasmine-standalone/SpecR
Google
velesin/ja...
gist: 7072...
https://gi...
velesin/ja...
velesin/ja...
about:blank
Jasmine T...
searls/jas...
v.gd – UR...
Jasmine 1.0.1 revision 1286311016
Show passed skipped
3 specs, 1 failure in 0.012s Finished at Fri Jan 07 2011 12:39:37 GMT-0500 (EST)
run all
Player
run
shows loading indicator
run
Expected spy show to have been called.

# Go red

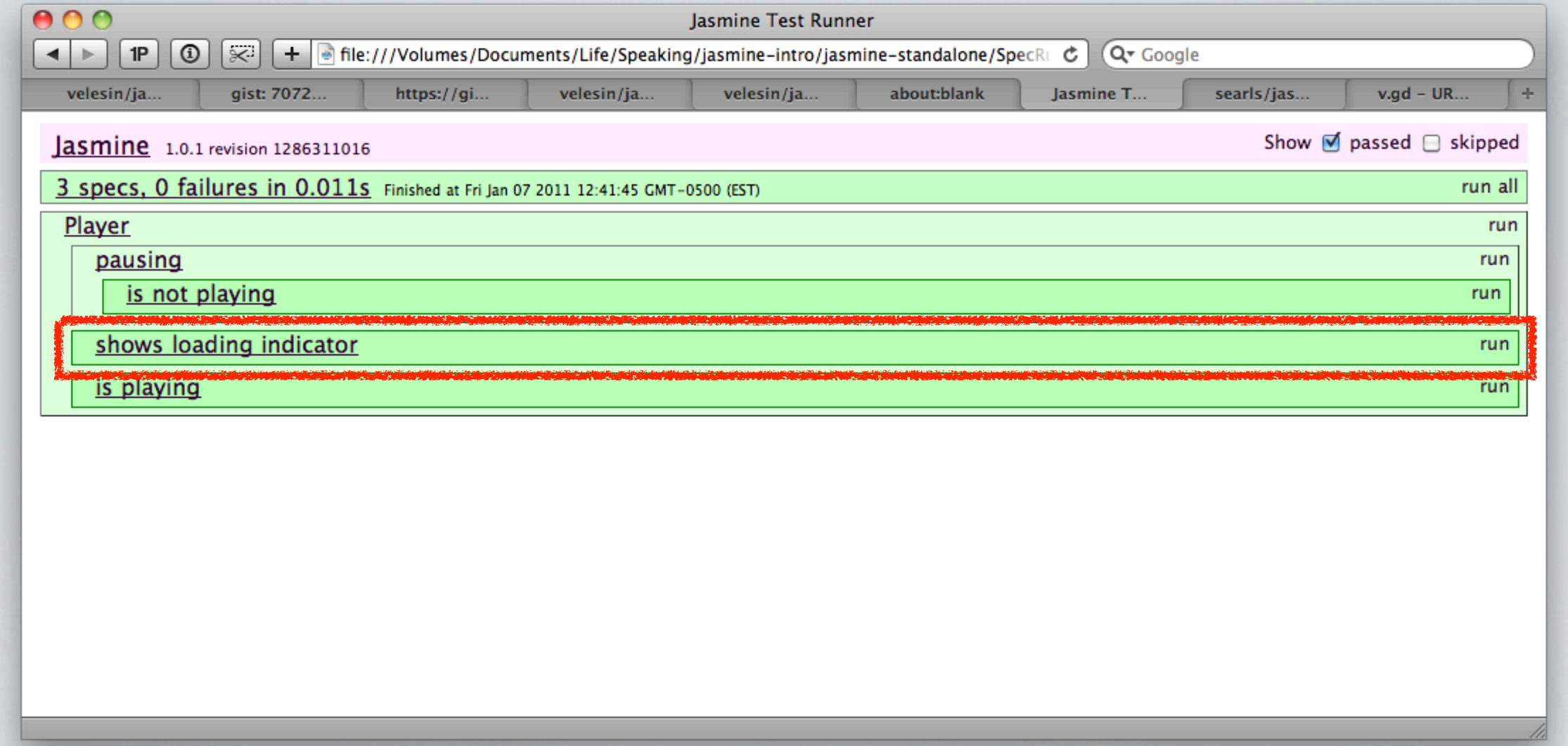
Player.js — jasmine-standalone

PlayerSpec.js | Player.js | SpecRunner.html | SpecHelper.js | SongSpec.js | Song.js

```
function Player() {
}
Player.prototype.play = function(song) {
  loading.show();
  this.currentlyPlayingSong = song;
  this.isPlaying = true;
};

Player.prototype.pause = function() {
  this.isPlaying = false;
};
```

Line: 4 Column: 18 | JavaScript | Soft Tabs: 2 | play

# Implement


Jasmine Test Runner
file:///Volumes/Documents/Life/Speaking/jasmine-intro/jasmine-standalone/SpecR
Google
velesin/ja...
gist: 7072...
https://gi...
velesin/ja...
velesin/ja...
about:blank
Jasmine T...
searls/jas...
v.gd – UR...
Jasmine 1.0.1 revision 1286311016
Show passed skipped
3 specs, 0 failures in 0.011s Finished at Fri Jan 07 2011 12:41:45 GMT-0500 (EST)
run all
Player
run
pausing
run
is not playing
run
shows loading indicator
run
is playing
run

Go green!

# 5. HTML Fixtures

"How do my specs see my HTML?"



This is the first question I usually hear when I people are first introduced to JavaScript testing. Most JavaScript written today is one-time-use



First, make an attempt to resist the temptation to dunk your delicious JavaScript in the milky DOM!

Specifying your JavaScript in a way that's independent from the exact markup it's used with has numerous benefits. Right out the gate, your JavaScript has two customers (your spec and your page), meaning that it's immediately reusable. With practice, it will probably be more modular and better abstracted, too!

*However*



But, in reality, the majority of people new to JavaScript use it only for the DOM interactions that can't be performed on the server-side. Especially when you're starting out, having a way to easily and safely inject HTML fixtures into the DOM is really important.

A lightweight script for injecting HTML

http://v.gd/fixture



Link expands to:
https://github.com/searls/jasmine-fixture

SongSpec.js — jasmine-standalone

jasmine-fixture.js | SpecHelper.js | SongSpec.js

```
describe('Song',function(){
  var song,$currentSongSpan;
  beforeEach(function(){
    song = new Song();
    song.title = 'Some Song Title';
    $.jasmine.inject('<span id="currentSong"></span>');
    $currentSongSpan = $('#currentSong');
  });

});
```

Line: 9 Column: 1 | JavaScript | Soft Tabs: 2 | Song

# Injecting HTML



We can simply inject some HTML into the DOM to facilitate our test.

SongSpec.js — jasmine-standalone

jasmine-fixture.js | SpecHelper.js | SongSpec.js

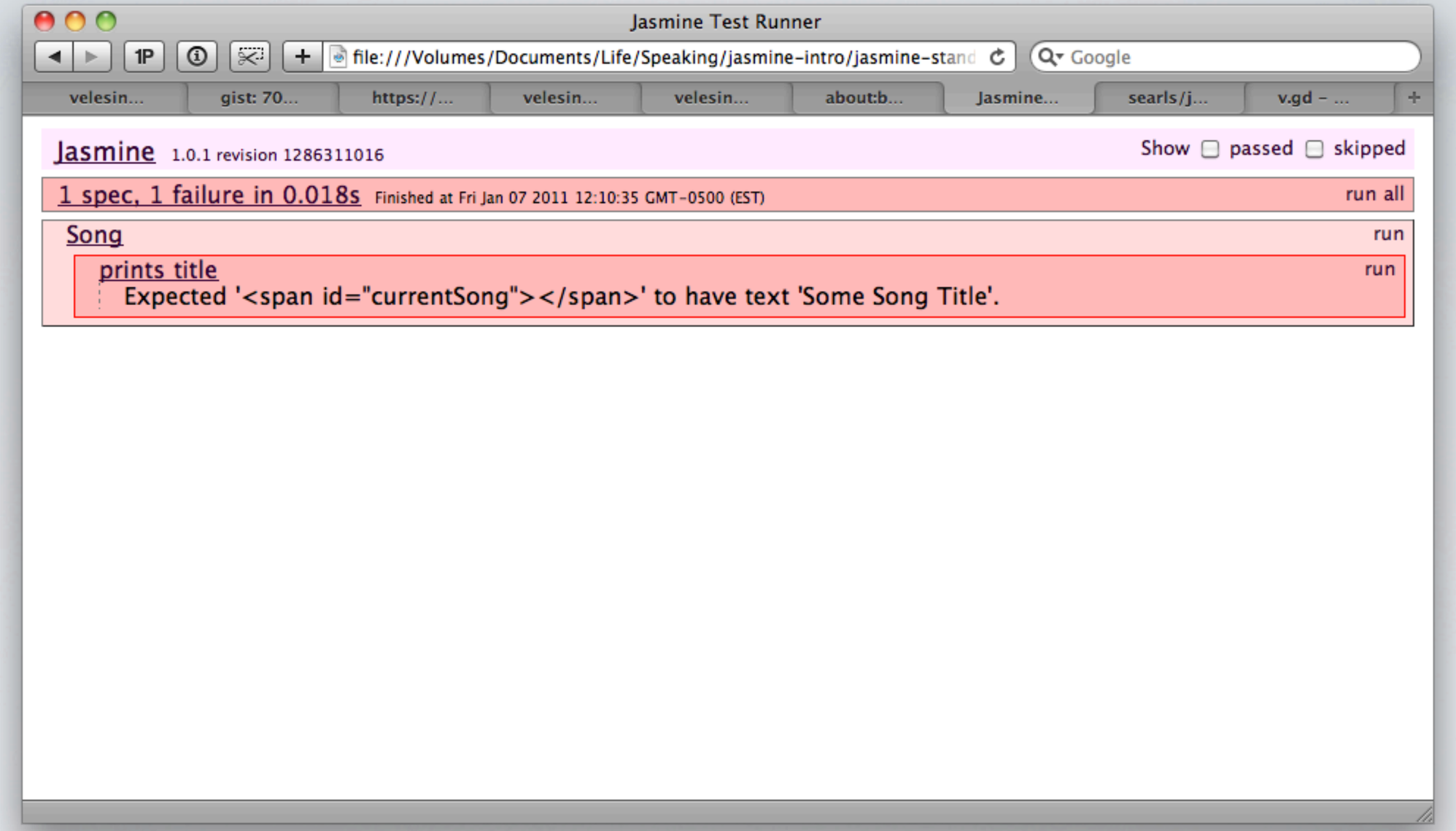
```
describe('Song',function(){
  var song,$currentSongSpan;
  beforeEach(function(){
    song = new Song();
    song.title = 'Some Song Title';
    $.jasmine.inject('<span id="currentSong"></span>');
    $currentSongSpan = $('#currentSong');
  });
  it('prints title',function(){
    song.printTitle();
    expect($currentSongSpan).toHaveText(song.title);
  })
});
```

Line: 13 Column: 4 | JavaScript | Soft Tabs: 2 | Song

# Injecting HTML



This way we can confidently specify the behavior of code interacting with the DOM, while minimizing the coupling to specific markup.




# Go red



Now we have a nice clear failure.

Song.js — jasmine-standalone

Song.js

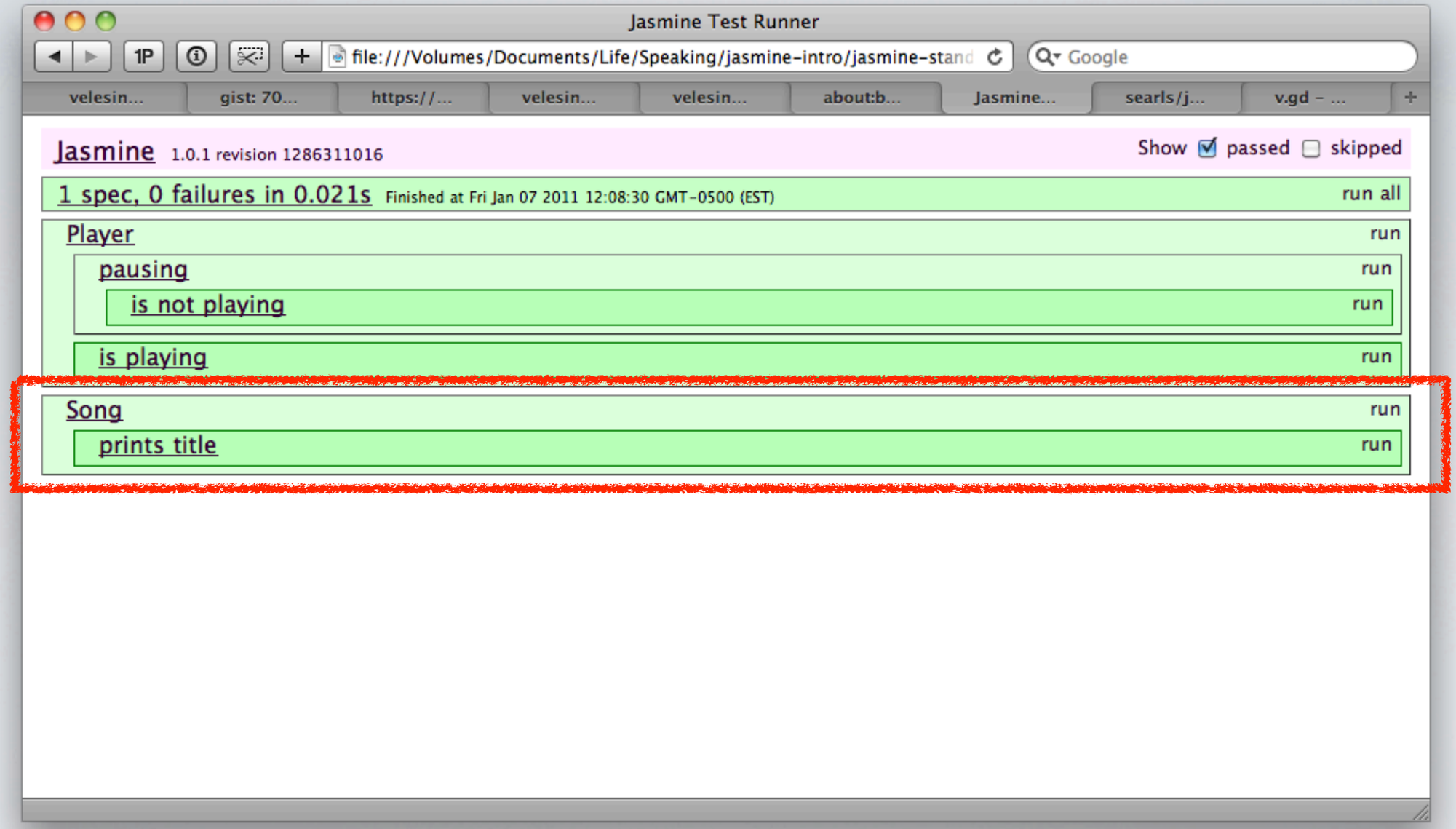
```
function Song() {
}

Song.prototype.printTitle = function(){
  $('#currentSong').text(this.title);
}
```

Line: 7  Column: 1  JavaScript  Soft Tabs: 2  printTitle

# Injecting HTML



Now our implementation is free to interact with the specific DOM element our test provides

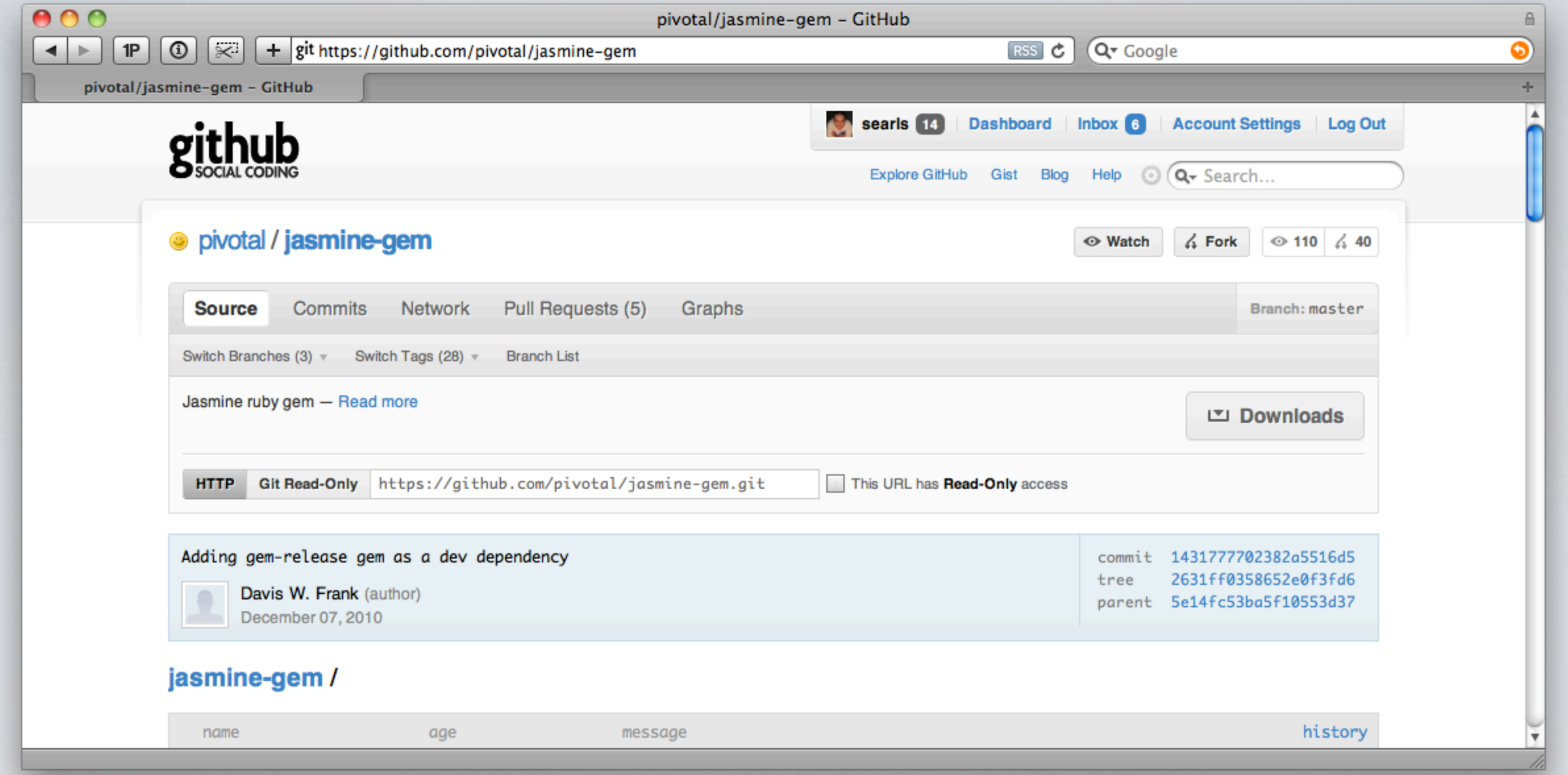



# Go green!

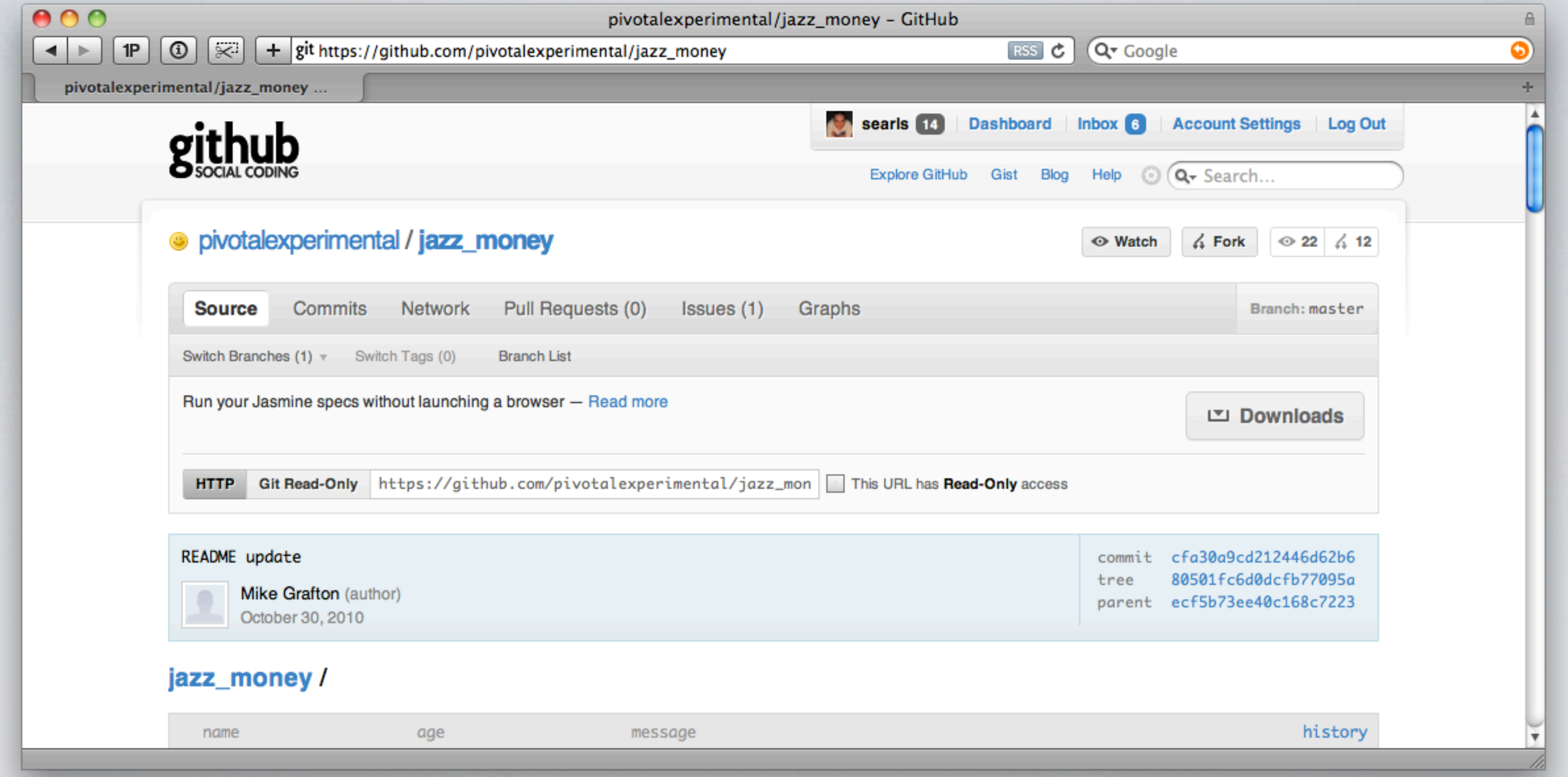


And our spec goes green!

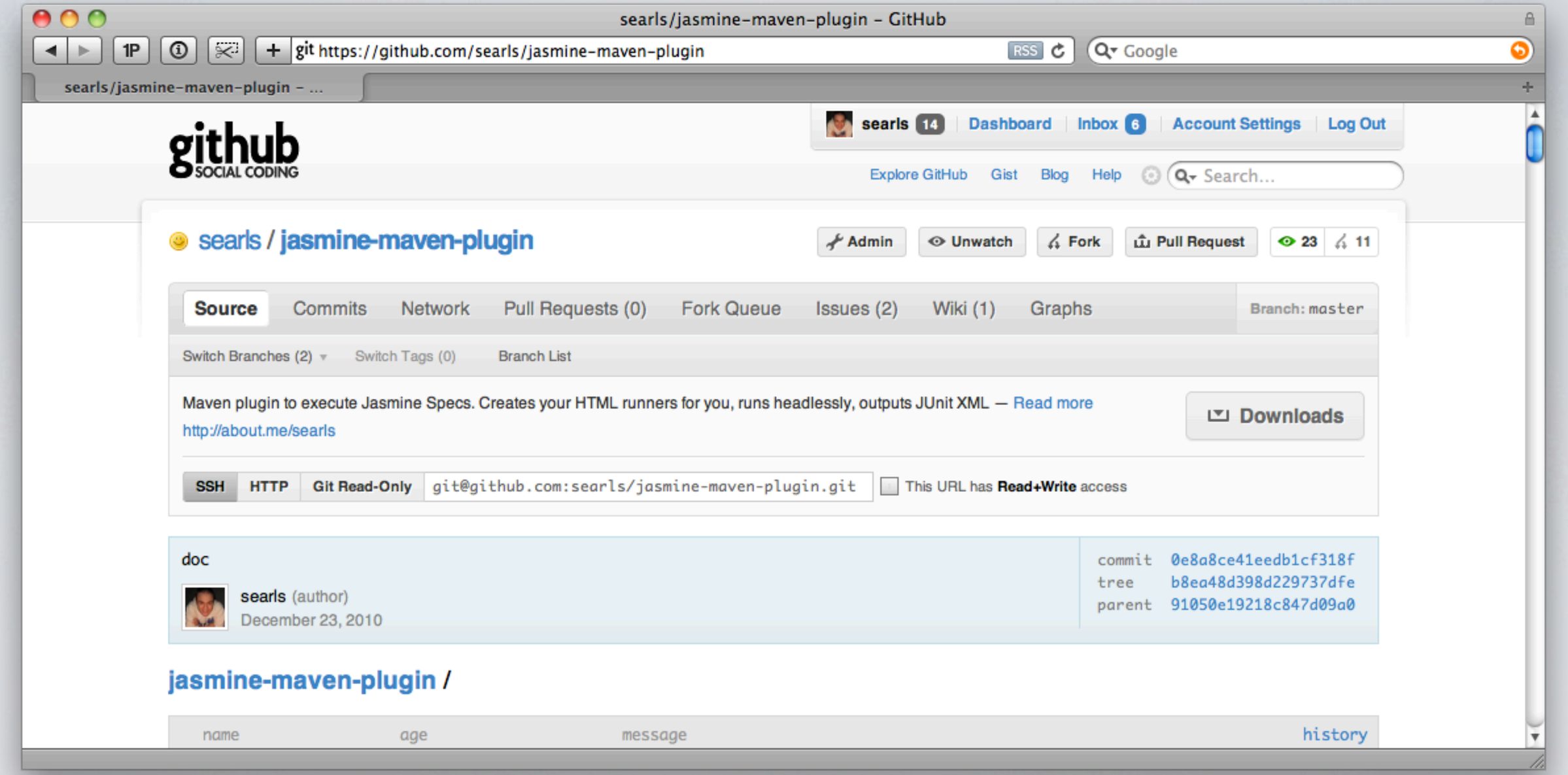
# 6. Continuous Integration / Project Comprehension



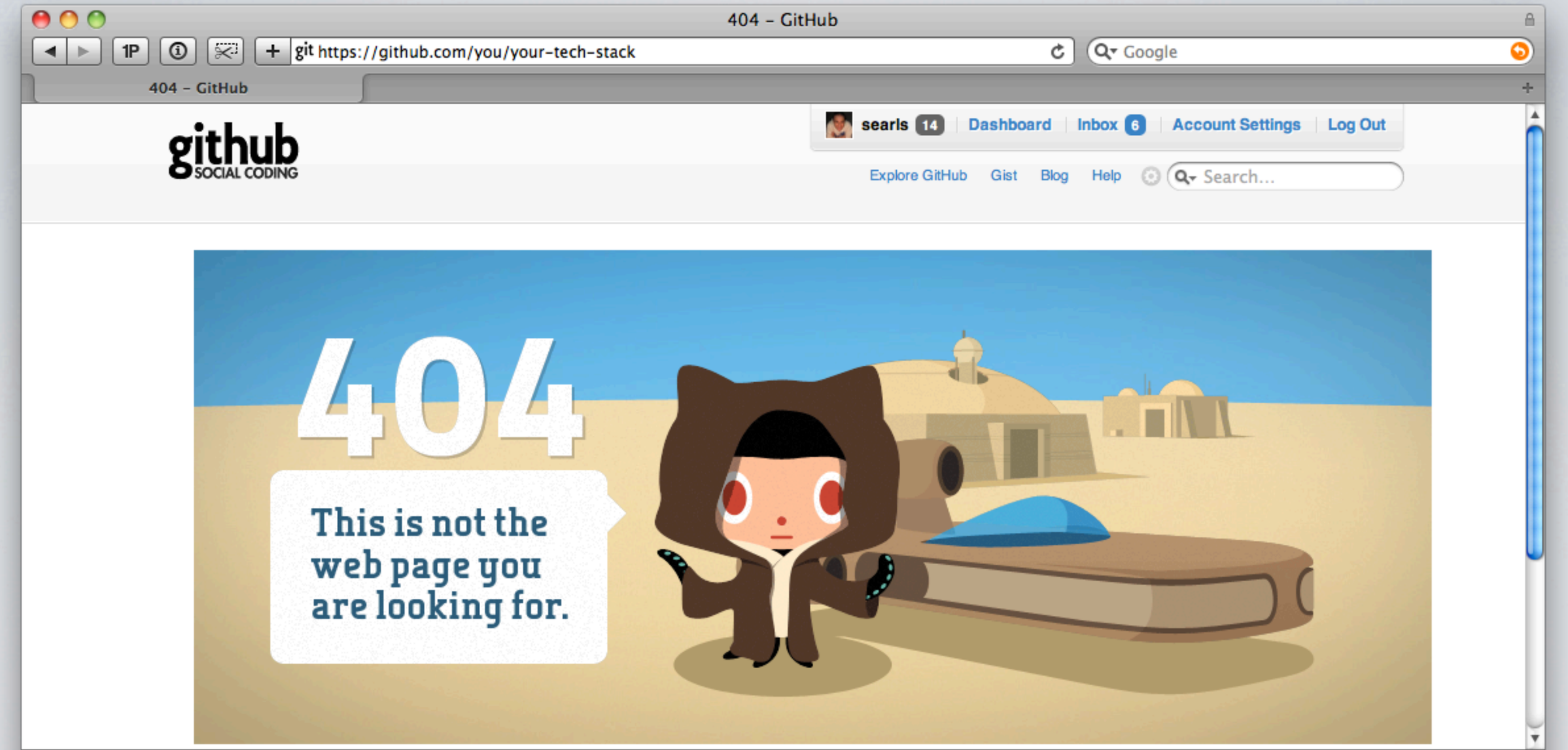

# jasmine-gem for Ruby/Rails under Selenium




# jazz_money for headless Ruby




# jasmine-maven-plugin for Java projects




# Let's build tools for your tech stack together!

# Image Credits

- http://www.sxc.hu/photo/708473
- http://www.sxc.hu/photo/859430
- http://www.sxc.hu/photo/1103790
- http://www.sxc.hu/photo/1309077
- http://www.flickr.com/photos/paperdollimages/5127685l8/
  (via *¦ ·sindorella ·¦* )