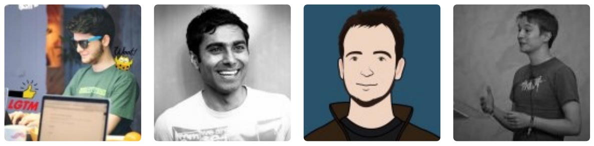
MOCHA simple, flexible, fun

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 4 sponsors 2

Backers

Support us with a monthly donation and help us continue our activities. [**Become a backer**]



Become a backer

Sponsors

Is your company using Mocha? Ask your manager to support us. Your company logo will show up here and on our **Github page**. [**Become a sponsor**]

SAUCELABS

Auth0 Token Based Authentication

open collective Become a sponsor

Features

- browser support
- simple async support, including promises
- test coverage reporting
- string diff support
- javascript API for running tests
- proper exit status for CI support etc
- auto-detects and disables coloring for non-ttys
- maps uncaught exceptions to the correct test case
- async test timeout support
- test retry support
- test-specific timeouts
- growl notification support
- reports test durations
- highlights slow tests
- file watcher support
- global variable leak detection
- optionally run tests that match a regexp

- auto-exit to prevent “hanging” with an active loop
- easily meta-generate suites & test-cases
- mocha.opts file support
- clickable suite titles to filter test execution
- node debugger support
- detects multiple calls to `done()`
- use any assertion library you want
- extensible reporting, bundled with 9+ reporters
- extensible test DSLs or “interfaces”
- before, after, before each, after each hooks
- arbitrary transpiler support (coffee-script etc)
- TextMate bundle
- and more!

Table of Contents

Installation

Install with **npm**:

```
$ npm install -g mocha
```

Getting Started

```
$ npm install -g mocha
$ mkdir test
$ $EDITOR test/test.js
```

In your editor:

```
var assert = require('chai').assert;
describe('Array', function() {
  describe('#indexOf()', function () {
    it('should return -1 when the value is not present',
function () {
      assert.equal(-1, [1,2,3].indexOf(5));
      assert.equal(-1, [1,2,3].indexOf(0));
    });
  });
});
```

Back in the terminal:

```
$  mocha

  .
  ✓ 1 test complete (1ms)
```

Assertions

Mocha allows you to use any assertion library you want, if it throws an error, it will work! This means you can utilize libraries such as:

should.js BDD style shown throughout these docs	**better-assert** c-style self-documenting assert()
expect.js expect() style assertions	**unexpected** the extensible BDD assertion toolkit
chai expect(), assert() and should style assertions	

Synchronous Code

When testing synchronous code, omit the callback and Mocha will automatically continue on to the next test.

```
describe('Array', function() {
  describe('#indexOf()', function() {
    it('should return -1 when the value is not present',
function() {
      [1,2,3].indexOf(5).should.equal(-1);
      [1,2,3].indexOf(0).should.equal(-1);
    });
  });
});
```

Asynchronous Code

Testing asynchronous code with Mocha could not be simpler! Simply invoke the callback when your test is complete. By adding a callback (usually named `done`) to `it()` Mocha will know that it should wait for completion.

```
describe('User', function() {
  describe('#save()', function() {
    it('should save without error', function(done) {
      var user = new User('Luna');
      user.save(function(err) {
        if (err) throw err;
        done();
      });
    });
  });
});
```

To make things even easier, the `done()` callback accepts an error, so we may use this directly:

```
describe('User', function() {
  describe('#save()', function() {
    it('should save without error', function(done) {
      var user = new User('Luna');
      user.save(done);
    });
  });
});
```

Working with Promises

Alternately, instead of using the `done()` callback, you may return a **Promise**. This is useful if the APIs you are testing return promises instead of taking callbacks:

```
beforeEach(function() {
  return db.clear()
    .then(function() {
      return db.save([tobi, loki, jane]);
    });
});

describe('#find()', function() {
  it('respond with matching records', function() {
    return db.find({ type: 'User'
}).should.eventually.have.length(3);
  });
});
```

(The latter example uses **Chai as Promised** for fluent promise assertions.)

Arrow Functions

Passing **arrow functions** to Mocha is discouraged. Their lexical binding of the `this` value makes them unable to access the Mocha context, and statements like `this.timeout(1000);` will not work inside an arrow function.

Hooks

Mocha provides the hooks `before()`, `after()`, `beforeEach()`, and `afterEach()`, which can be used to set up preconditions and clean up after your tests.

```
describe('hooks', function() {

  before(function() {
    // runs before all tests in this block
  });

  after(function() {
    // runs after all tests in this block
  });

  beforeEach(function() {
    // runs before each test in this block
  });

  afterEach(function() {
    // runs after each test in this block
  });

  // test cases
});
```

Describing Hooks

All hooks can be invoked with an optional description, making it easier to pinpoint errors in your tests. If hooks are given named functions, those names will be used if no description is supplied.

```
beforeEach(function() {
  // beforeEach hook
});

beforeEach(function namedFun() {
  // beforeEach:namedFun
});

beforeEach('some description', function() {
  // beforeEach:some description
});
```

Asynchronous Hooks

All “hooks” (`before()`, `after()`, `beforeEach()`, `afterEach()`) may be sync or async as well, behaving much like a regular test-case. For example, you may wish to populate database with dummy content before each test:

```
describe('Connection', function() {
  var db = new Connection,
    tobi = new User('tobi'),
    loki = new User('loki'),
    jane = new User('jane');

  beforeEach(function(done) {
    db.clear(function(err) {
      if (err) return done(err);
      db.save([tobi, loki, jane], done);
    });
  });

  describe('#find()', function() {
    it('respond with matching records', function(done) {
      db.find({type: 'User'}, function(err, res) {
        if (err) return done(err);
        res.should.have.length(3);
        done();
      });
    });
  });
});
```

Root-Level Hooks

You may also pick any file and add “root”-level hooks. For example, add `beforeEach()` outside of all `describe()` blocks. This will cause the callback to `beforeEach()` to run before any test case, regardless of the file it lives in (this is because Mocha has a hidden `describe()` block, called the “root suite”).

```
beforeEach(function() {
  console.log('before every test in every file');
});
```

Delayed Root Suite

If you need to perform asynchronous operations before any of your suites are run, you may delay the root suite. Simply run Mocha with the `--delay` flag. This will provide a special function, `run()`, in the global context.

```
setTimeout(function() {
  // do some setup

  describe('my suite', function() {
    // ...
  });

  run();
}, 5000);
```

Pending Tests

“Pending”–as in “someone should write these test cases eventually”–test-cases are simply those without a callback:

```
describe('Array', function() {
  describe('#indexOf()', function() {
    // pending test below
    it('should return -1 when the value is not present');
  });
});
```

Pending tests will be reported as such.

Exclusive Tests

The exclusivity feature allows you to run *only* the specified suite or test-case by appending `.only()` to the function. Here's an example of executing only a particular suite:

```
describe('Array', function() {
  describe.only('#indexOf()', function() {
    // ...
  });
});
```

Note: All nested suites will still be executed.

Here's an example of executing a particular test case:

```
describe('Array', function() {
  describe('#indexOf()', function() {
    it.only('should return -1 unless present', function() {
      // ...
    });

    it('should return the index when present', function() {
      // ...
    });
  });
});
```

Note: Hooks, if present, will still be executed.

Warning: Having more than one call to `.only()` in your tests or suites may result in unexpected behavior.

Inclusive Tests

This feature is the inverse of `.only()`. By appending `.skip()`, you may tell Mocha to simply ignore these suite(s) and test case(s). Anything skipped will be marked as **pending**, and reported as such. Here's an example of skipping an entire suite:

```
describe('Array', function() {
  describe.skip('#indexOf()', function() {
    // ...
  });
});
```

> *Best practice*: Use `.skip()` instead of commenting tests out.

Or a specific test-case:

```
describe('Array', function() {
  describe('#indexOf()', function() {
    it.skip('should return -1 unless present', function() {
      // ...
    });

    it('should return the index when present', function() {
      // ...
    });
  });
});
```

Retry Tests

You can choose to retry failed tests up to a certain number of times. This feature is designed to handle end-to-end tests (functional tests/Selenium…) where resources cannot be easily mocked/stubbed. **It's**

not recommended to use this feature for unit tests.

This feature does re-run `beforeEach/afterEach` hooks but not `before/after` hooks.

NOTE: Example below was written using Selenium webdriver (which **overwrites global Mocha hooks** for `Promise` chain).

```
describe('retries', function() {
  // Retry all tests in this suite up to 4 times
  this.retries(4);

  beforeEach(function () {
    browser.get('http://www.yahoo.com');
  });

  it('should succeed on the 3rd try', function () {
    // Specify this test to only retry up to 2 times
    this.retries(2);
    expect($('.foo').isDisplayed()).to.eventually.be.true;
  });
});
```

Dynamically Generating Tests

Given Mocha's use of `Function.prototype.call` and function expressions to define suites and test cases, it's straightforward to generate your tests dynamically. No special syntax is required — plain ol' JavaScript can be used to achieve functionality similar to "parameterized" tests, which you may have seen in other frameworks.

Take the following example:

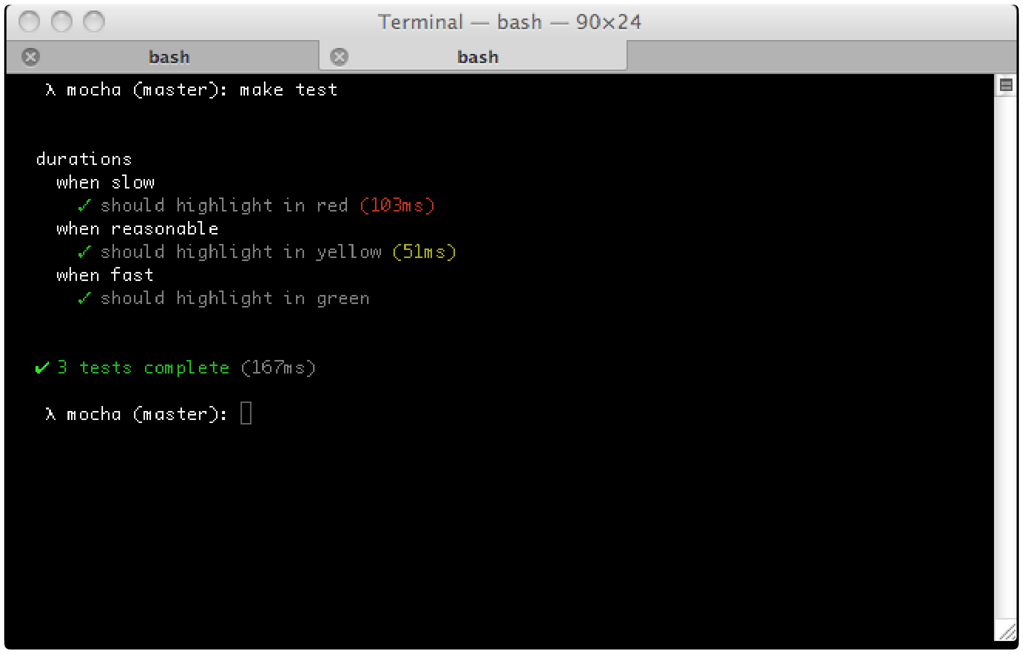
```
var assert = require('chai').assert;

function add() {
  return
Array.prototype.slice.call(arguments).reduce(function(prev,
curr) {
    return prev + curr;
  }, 0);
}

describe('add()', function() {
  var tests = [
    {args: [1, 2],       expected: 3},
    {args: [1, 2, 3],    expected: 6},
    {args: [1, 2, 3, 4], expected: 10}
  ];

  tests.forEach(function(test) {
    it('correctly adds ' + test.args.length + ' args',
function() {
      var res = add.apply(null, test.args);
      assert.equal(res, test.expected);
    });
  });
});
```

The above code will produce a suite with three specs:

```
$ mocha

  add()
    ✓ correctly adds 2 args
    ✓ correctly adds 3 args
    ✓ correctly adds 4 args
```

Test duration

Many reporters will display test duration, as well as flagging tests that are slow, as shown here with the “spec” reporter:



```
 λ mocha (master): make test


durations
  when slow
    ✓ should highlight in red (103ms)
  when reasonable
    ✓ should highlight in yellow (51ms)
  when fast
    ✓ should highlight in green


✔ 3 tests complete (167ms)

 λ mocha (master):
```

To tweak what's considered "slow", you can use the `slow()` method:

```
describe('something slow', function() {
  this.slow(10000);

  it('should take long enough for me to go make a sandwich', function() {
    // ...
  });
});
```

Timeouts

Suite-level

Suite-level timeouts may be applied to entire test "suites", or disabled via `this.timeout(0)`. This will be inherited by all nested suites and test-cases that do not override the value.

```
describe('a suite of tests', function() {
  this.timeout(500);

  it('should take less than 500ms', function(done){
    setTimeout(done, 300);
  });

  it('should take less than 500ms as well', function(done){
    setTimeout(done, 250);
  });
})
```

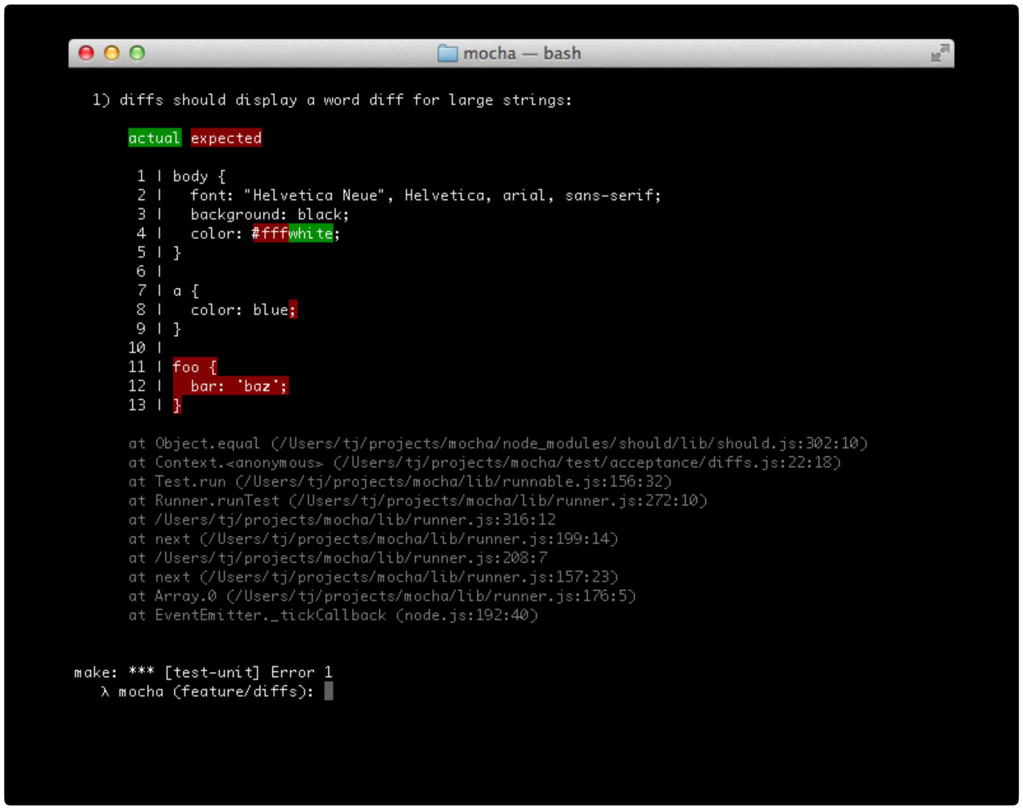
Test-level

Test-specific timeouts may also be applied, or the use of `this.timeout(0)` to disable timeouts all together:

```
it('should take less than 500ms', function(done){
  this.timeout(500);
  setTimeout(done, 300);
});
```

Diffs

Mocha supports the `err.expected` and `err.actual` properties of any thrown `AssertionError`s from an assertion library. Mocha will attempt to display the difference between what was expected, and what the assertion actually saw. Here's an example of a "string" diff:



```
  1) diffs should display a word diff for large strings:

      actual expected

       1 | body {
       2 |   font: "Helvetica Neue", Helvetica, arial, sans-serif;
       3 |   background: black;
       4 |   color: #fffwhite;
       5 | }
       6 |
       7 | a {
       8 |   color: blue;
       9 | }
      10 |
      11 | foo {
      12 |   bar: 'baz';
      13 | }

      at Object.equal (/Users/tj/projects/mocha/node_modules/should/lib/should.js:302:10)
      at Context.<anonymous> (/Users/tj/projects/mocha/test/acceptance/diffs.js:22:18)
      at Test.run (/Users/tj/projects/mocha/lib/runnable.js:156:32)
      at Runner.runTest (/Users/tj/projects/mocha/lib/runner.js:272:10)
      at /Users/tj/projects/mocha/lib/runner.js:316:12
      at next (/Users/tj/projects/mocha/lib/runner.js:199:14)
      at /Users/tj/projects/mocha/lib/runner.js:208:7
      at next (/Users/tj/projects/mocha/lib/runner.js:157:23)
      at Array.0 (/Users/tj/projects/mocha/lib/runner.js:176:5)
      at EventEmitter._tickCallback (node.js:192:40)


make: *** [test-unit] Error 1
   λ mocha (feature/diffs):
```

Usage

```
Usage: mocha [debug] [options] [files]


Commands:


  init <path>
  initialize a client-side mocha setup at <path>


Options:


 -h, --help                                output usage
information
 -V, --version                             output the version
number
 -A, --async-only                          force all tests to
```

```
take a callback (async) or return a promise
 -c, --colors                              force enabling of
colors
 -C, --no-colors                           force disabling of
colors
 -G, --growl                               enable growl
notification support
 -O, --reporter-options <k=v,k2=v2,...>  reporter-specific
options
 -R, --reporter <name>                     specify the
reporter to use
 -S, --sort                                sort test files
 -b, --bail                                bail after first
test failure
 -d, --debug                               enable node's
debugger, synonym for node --debug
 -g, --grep <pattern>                      only run tests
matching <pattern>
 -f, --fgrep <string>                      only run tests
containing <string>
 -gc, --expose-gc                          expose gc
extension
 -i, --invert                              inverts --grep and
--fgrep matches
 -r, --require <name>                      require the given
module
 -s, --slow <ms>                           "slow" test
threshold in milliseconds [75]
 -t, --timeout <ms>                        set test-case
timeout in milliseconds [2000]
 -u, --ui <name>                           specify user-
interface (bdd|tdd|exports)
 -w, --watch                               watch files for
changes
 --check-leaks                             check for global
variable leaks
 --compilers <ext>:<module>,...            use the given
module(s) to compile files
 --debug-brk                               enable node's
debugger breaking on the first line
 --delay                                   wait for async
```

```
suite definition
 --es_staging                          enable all staged
features
 --full-trace                          display the full
stack trace
 --globals <names>                     allow the given
comma-delimited global [names]
 --harmony                             enable all harmony
features (except typeof)
 --harmony-collections                 enable harmony
collections (sets, maps, and weak maps)
 --harmony-generators                  enable harmony
generators
 --harmony-proxies                     enable harmony
proxies
 --harmony_arrow_functions             enable "harmony
arrow functions" (iojs)
 --harmony_classes                     enable "harmony
classes" (iojs)
 --harmony_proxies                     enable "harmony
proxies" (iojs)
 --harmony_shipping                    enable all shipped
harmony features (iojs)
 --inline-diffs                        display
actual/expected differences inline within each string
 --interfaces                          display available
interfaces
 --no-deprecation                      silence
deprecation warnings
 --no-exit                             require a clean
shutdown of the event loop: mocha will not call
process.exit
 --no-timeouts                         disables timeouts,
given implicitly with --debug
 --opts <path>                         specify opts path
 --prof                                log statistical
profiling information
 --recursive                           include sub
directories
 --reporters                           display available
reporters
```

```
 --retries                               set number of
times to retry failed test cases
 --throw-deprecation                     throw an exception
anytime a deprecated function is used
 --trace                                 trace function
calls
 --trace-deprecation                     show stack traces
on deprecations
 --watch-extensions <ext>,...            additional
extensions to monitor with --watch
```

`-w, --watch`

Executes tests on changes to JavaScript in the CWD, and once initially.

`--compilers`

CoffeeScript is no longer supported out of the box. CS and similar transpilers may be used by mapping the file extensions (for use with `--watch`) and the module name. For example `--compilers coffee:coffee-script` with CoffeeScript 1.6- or `--compilers coffee:coffee-script/register` with CoffeeScript 1.7+.

`-b, --bail`

Only interested in the first exception? use `--bail`!

`-d, --debug`

Enables node's debugger support, this executes your script(s) with `node debug <file ...>` allowing you to step through code and break with the `debugger` statement. Note the difference between `mocha debug` and `mocha --debug`: `mocha debug` will fire up node's built-in debug client, `mocha --debug` will allow you to use a different interface — such as the Blink Developer Tools.

`--globals <names>`

Accepts a comma-delimited list of accepted global variable names. For example, suppose your app deliberately exposes a global named `app` and `YUI`, you may want to add `--globals app,YUI`. It also accepts wildcards. You could do `--globals '*bar'` and it would match `foobar`, `barbar`, etc. You can also simply pass in `'*'` to ignore all globals.

`--check-leaks`

By default, Mocha will not check for global variables leaked while running tests, to enable this pass `--check-leaks`, to specify globals that are acceptable use `--globals`, for example `--globals jQuery,MyLib`.

`-r, --require <module-name>`

The `--require` option is useful for libraries such as **should.js**, so you may simply `--require should` instead of manually invoking `require('should')` within each test file. Note that this works well for `should` as it augments `Object.prototype`, however if you wish to access a module's exports you will have to require them, for example `var should = require('should')`. Furthermore, it can be used with relative paths, e.g. `--require ./test/helper.js`

`-u, --ui <name>`

The `--ui` option lets you specify the interface to use, defaulting to "bdd".

`-R, --reporter <name>`

The `--reporter` option allows you to specify the reporter that will be used, defaulting to "spec". This flag may also be used to utilize third-party reporters. For example if you `npm install mocha-lcov-reporter` you may then do `--reporter mocha-lcov-reporter`.

`-t, --timeout <ms>`

Specifies the test-case timeout, defaulting to 2 seconds. To override you may pass the timeout in milliseconds, or a value with the `s` suffix, ex: `--`

`timeout 2s` or `--timeout 2000` would be equivalent.

`-s, --slow <ms>`

Specify the “slow” test threshold, defaulting to 75ms. Mocha uses this to highlight test-cases that are taking too long.

`-g, --grep <pattern>`

The `--grep` option when specified will trigger mocha to only run tests matching the given `pattern` which is internally compiled to a `RegExp`.

Suppose, for example, you have “api” related tests, as well as “app” related tests, as shown in the following snippet; One could use `--grep api` or `--grep app` to run one or the other. The same goes for any other part of a suite or test-case title, `--grep users` would be valid as well, or even `--grep GET`.

```
describe('api', function() {
  describe('GET /api/users', function() {
    it('respond with an array of users', function() {
      // ...
    });
  });
});

describe('app', function() {
  describe('GET /users', function() {
    it('respond with an array of users', function() {
      // ...
    });
  });
});
```

Interfaces

Mocha's "interface" system allows developers to choose their style of DSL. Mocha has **BDD**, **TDD**, **Exports**, **QUnit** and **Require**-style interfaces.

BDD

The **BDD** interface provides `describe()`, `context()`, `it()`, `specify()`, `before()`, `after()`, `beforeEach()`, and `afterEach()`.

`context()` is just an alias for `describe()`, and behaves the same way; it just provides a way to keep tests easier to read and organized. Similarly, `specify()` is an alias for `it()`.

> All of the previous examples were written using the **BDD** interface.

```
describe('Array', function() {
  before(function() {
    // ...
  });

  describe('#indexOf()', function() {
    context('when not present', function() {
      it('should not throw an error', function() {
        (function() {
          [1,2,3].indexOf(4);
        }).should.not.throw();
      });
      it('should return -1', function() {
        [1,2,3].indexOf(4).should.equal(-1);
      });
    });
    context('when present', function() {
      it('should return the index where the element first
appears in the array', function() {
        [1,2,3].indexOf(3).should.equal(2);
      });
    });
  });
  });
```

TDD

The **TDD** interface provides `suite()`, `test()`, `suiteSetup()`, `suiteTeardown()`, `setup()`, and `teardown()`:

```
suite('Array', function() {
  setup(function() {
    // ...
  });

  suite('#indexOf()', function() {
    test('should return -1 when not present', function() {
      assert.equal(-1, [1,2,3].indexOf(4));
    });
  });
});
```

Exports

The **Exports** interface is much like Mocha's predecessor **expresso**. The keys `before`, `after`, `beforeEach`, and `afterEach` are special-cased, object values are suites, and function values are test-cases:

```
module.exports = {
  before: function() {
    // ...
  },

  'Array': {
    '#indexOf()': {
      'should return -1 when not present': function() {
        [1,2,3].indexOf(4).should.equal(-1);
      }
    }
  }
};
```

QUnit

The **QUnit**-inspired interface matches the “flat” look of QUnit, where the test suite title is simply defined before the test-cases. Like TDD, it uses `suite()` and `test()`, but resembling BDD, it also contains `before()`, `after()`, `beforeEach()`, and `afterEach()`.

```
function ok(expr, msg) {
  if (!expr) throw new Error(msg);
}

suite('Array');

test('#length', function() {
  var arr = [1,2,3];
  ok(arr.length == 3);
});

test('#indexOf()', function() {
  var arr = [1,2,3];
  ok(arr.indexOf(1) == 0);
  ok(arr.indexOf(2) == 1);
  ok(arr.indexOf(3) == 2);
});

suite('String');

test('#length', function() {
  ok('foo'.length == 3);
});
```

Require

The `require` interface allows you to require the `describe` and friend words directly using `require` and call them whatever you want. This interface is also useful if you want to avoid global variables in your tests.

Note: The `require` interface cannot be run via the `node` executable, and must be run via `mocha`.

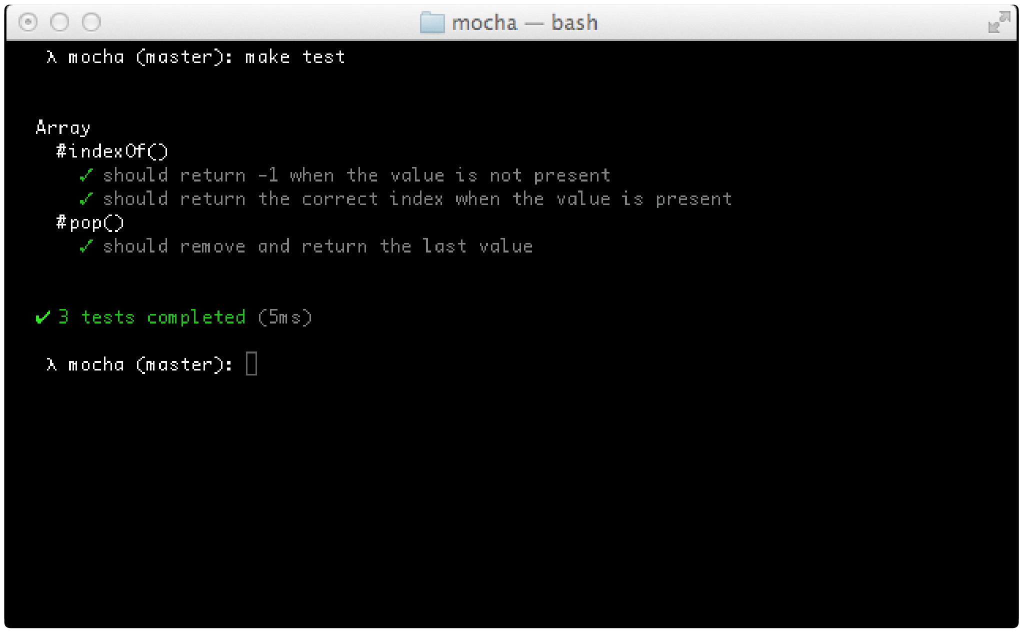
```
var testCase = require('mocha').describe;
var pre = require('mocha').before;
var assertions = require('mocha').it;
var assert = require('chai').assert;

testCase('Array', function() {
  pre(function() {
    // ...
  });

  testCase('#indexOf()', function() {
    assertions('should return -1 when not present',
function() {
      assert.equal([1,2,3].indexOf(4), -1);
    });
  });
});
```

Reporters

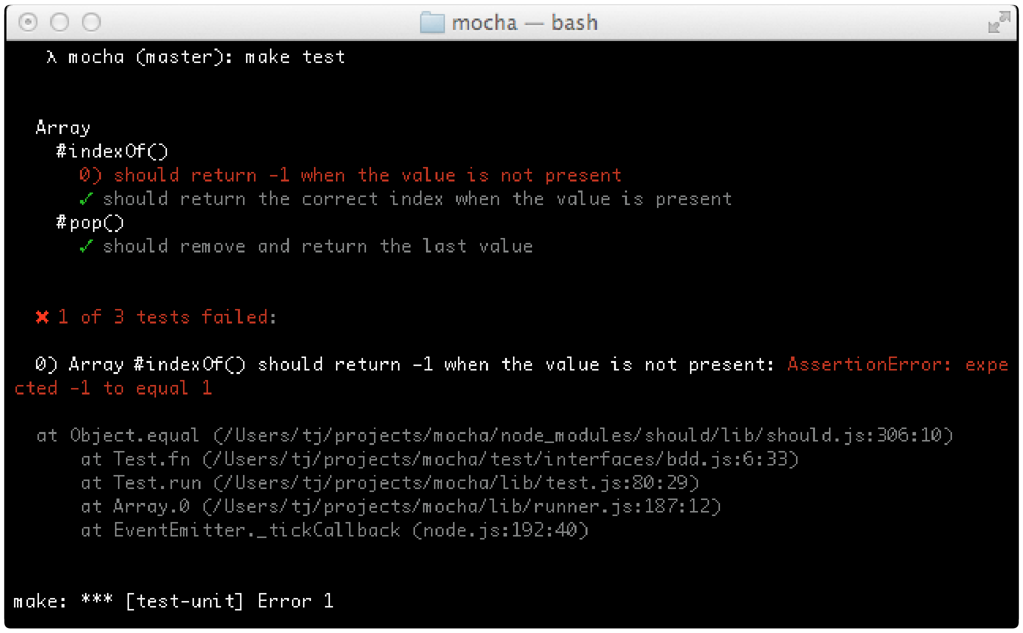
Mocha reporters adjust to the terminal window, and always disable ANSI-escape coloring when the stdio streams are not associated with a TTY.

Spec

This is the default reporter. The “spec” reporter outputs a hierarchical view nested just as the test cases are.



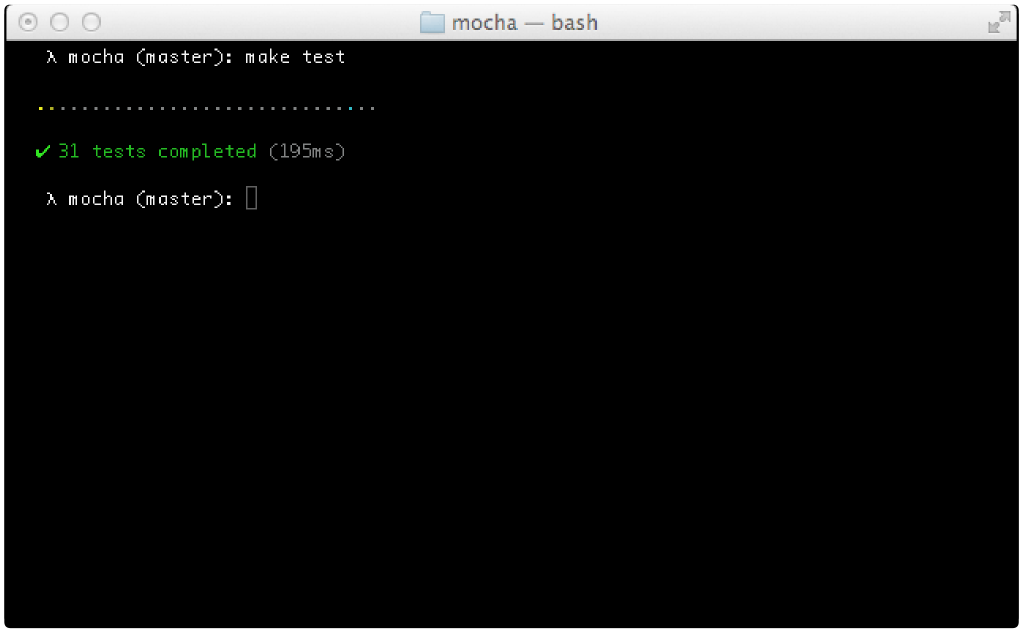
```
mocha — bash
 λ mocha (master): make test


Array
  #indexOf()
    ✓ should return -1 when the value is not present
    ✓ should return the correct index when the value is present
  #pop()
    ✓ should remove and return the last value


✓ 3 tests completed (5ms)

 λ mocha (master):
```



```
mocha — bash
 λ mocha (master): make test


Array
  #indexOf()
    0) should return -1 when the value is not present
    ✓ should return the correct index when the value is present
  #pop()
    ✓ should remove and return the last value


✗ 1 of 3 tests failed:

0) Array #indexOf() should return -1 when the value is not present: AssertionError: expe
cted -1 to equal 1

  at Object.equal (/Users/tj/projects/mocha/node_modules/should/lib/should.js:306:10)
      at Test.fn (/Users/tj/projects/mocha/test/interfaces/bdd.js:6:33)
      at Test.run (/Users/tj/projects/mocha/lib/test.js:80:29)
      at Array.0 (/Users/tj/projects/mocha/lib/runner.js:187:12)
      at EventEmitter._tickCallback (node.js:192:40)


make: *** [test-unit] Error 1
```

Dot Matrix

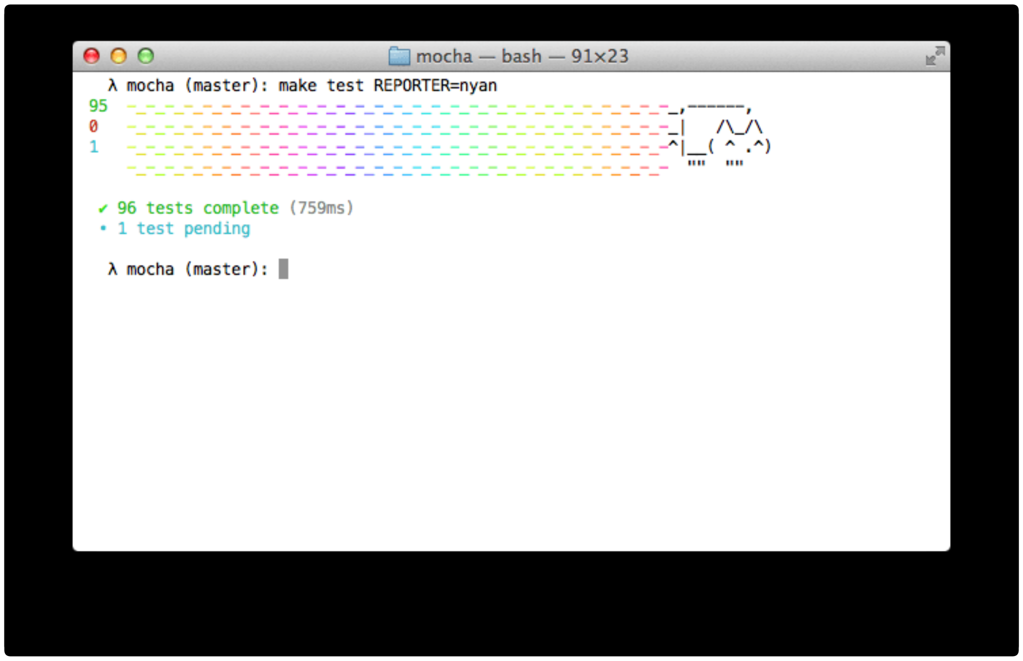
The dot matrix (or “dot”) reporter is simply a series of dots that represent test cases, failures highlight in red, pending in blue, slow as yellow. Good if you would like minimal output.



```
λ mocha (master): make test

..............................

✔ 31 tests completed (195ms)

λ mocha (master):
```

Nyan

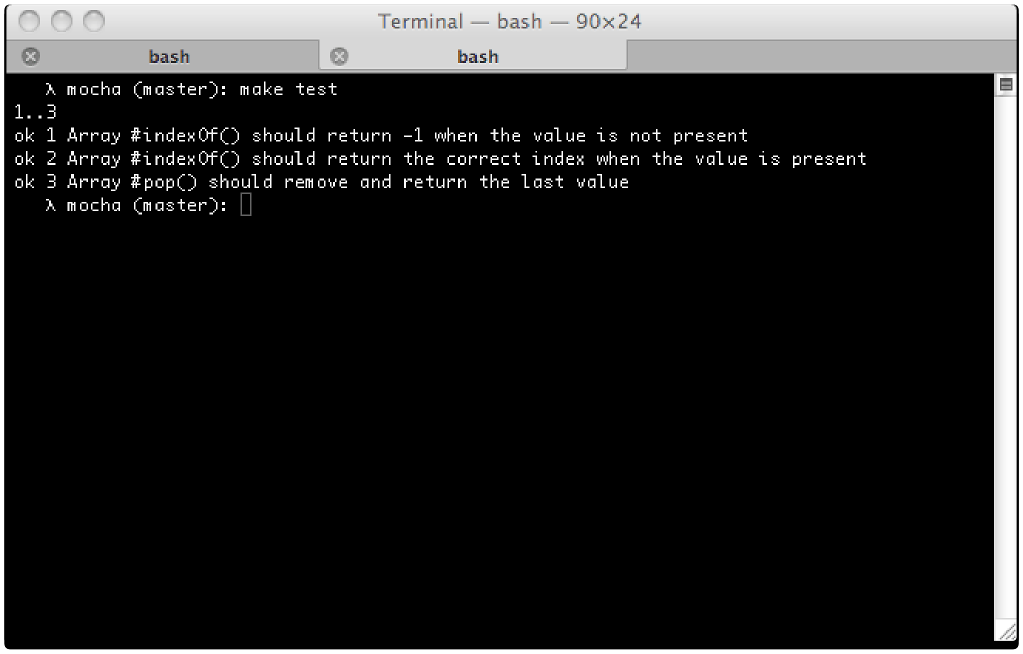
The “nyan” reporter is exactly what you might expect:



```
 λ mocha (master): make test REPORTER=nyan
95  -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_,------,
0   -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_|   /\_/\
1   -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-^|__( ^ .^)
    -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-  ""  ""

  ✔ 96 tests complete (759ms)
  • 1 test pending

 λ mocha (master):
```

TAP

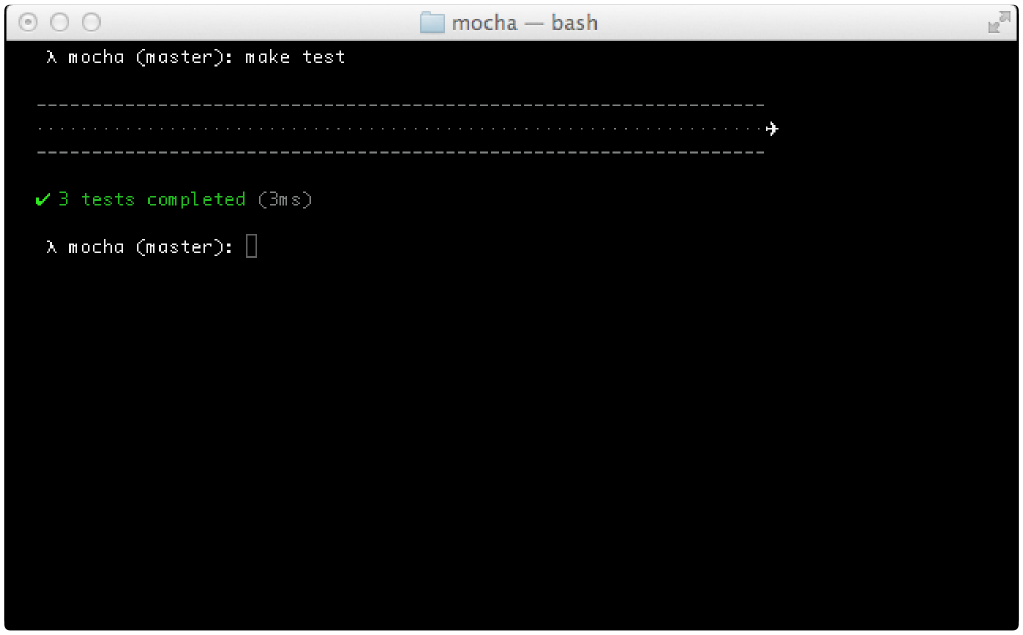
The TAP reporter emits lines for a **Test-Anything-Protocol** consumer.



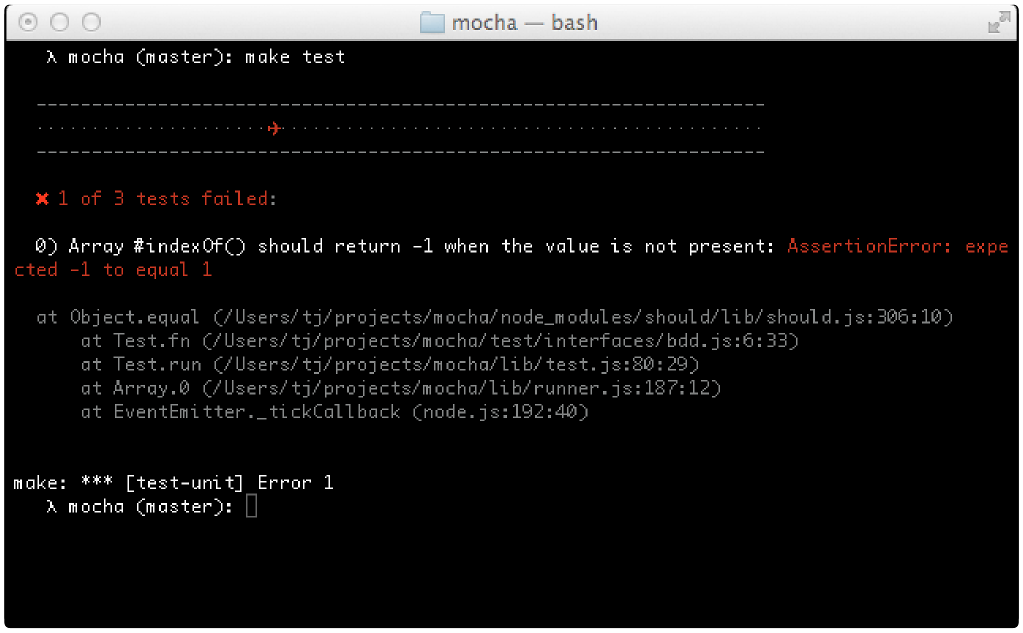
```
   λ mocha (master): make test
1..3
ok 1 Array #indexOf() should return -1 when the value is not present
ok 2 Array #indexOf() should return the correct index when the value is present
ok 3 Array #pop() should remove and return the last value
   λ mocha (master):
```

Landing Strip

The 'Landing Strip' reporter is a gimmicky test reporter simulating a plane landing :) unicode ftw



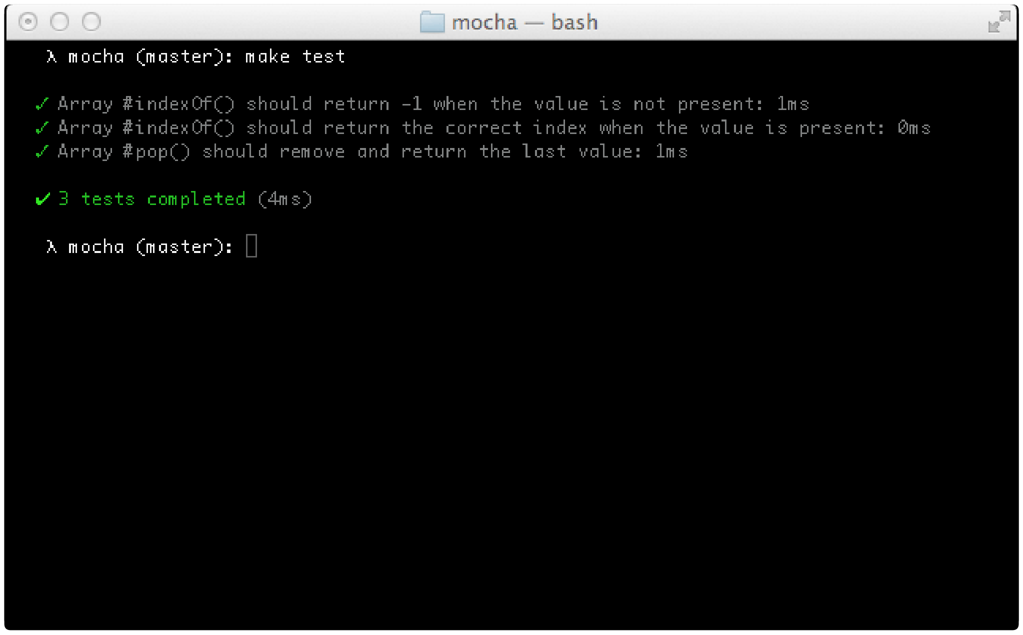
```
   λ mocha (master): make test

  ------------------------------------------------------------------
  ··································································✈
  ------------------------------------------------------------------

  ✔ 3 tests completed (3ms)

   λ mocha (master):
```



```
  λ mocha (master): make test

  ------------------------------------------------------------------
  ·····················✈·············································
  ------------------------------------------------------------------

  ✖ 1 of 3 tests failed:

  0) Array #indexOf() should return -1 when the value is not present: AssertionError: expe
cted -1 to equal 1

  at Object.equal (/Users/tj/projects/mocha/node_modules/should/lib/should.js:306:10)
      at Test.fn (/Users/tj/projects/mocha/test/interfaces/bdd.js:6:33)
      at Test.run (/Users/tj/projects/mocha/lib/test.js:80:29)
      at Array.0 (/Users/tj/projects/mocha/lib/runner.js:187:12)
      at EventEmitter._tickCallback (node.js:192:40)


make: *** [test-unit] Error 1
  λ mocha (master): 
```

List

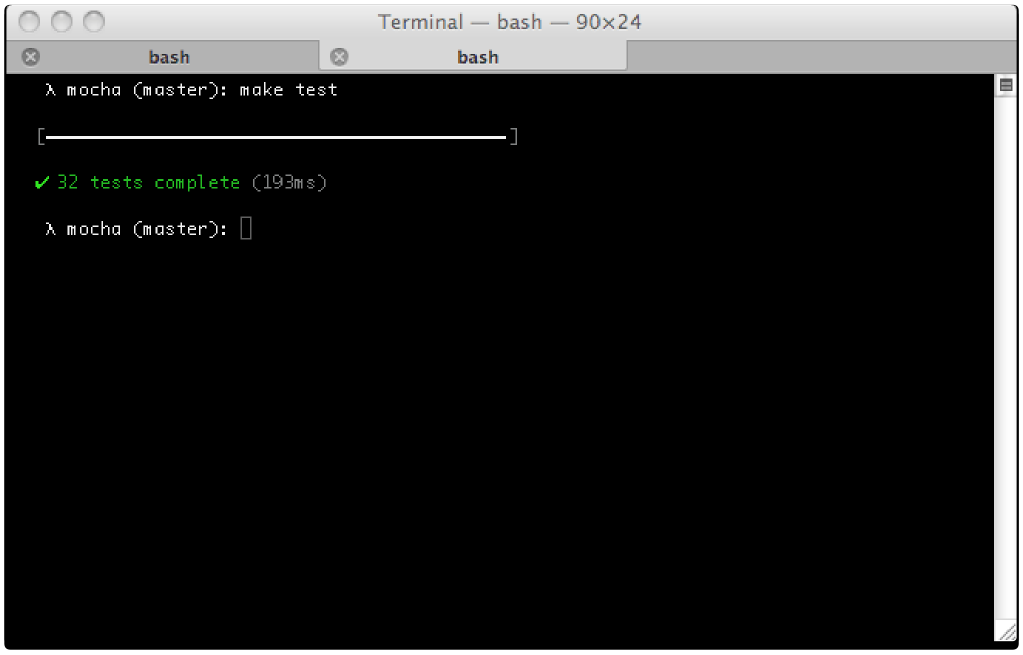
The “list” reporter outputs a simple specifications list as test cases pass or fail, outputting the failure details at the bottom of the output.



```
  λ mocha (master): make test

  ✓ Array #indexOf() should return -1 when the value is not present: 1ms
  ✓ Array #indexOf() should return the correct index when the value is present: 0ms
  ✓ Array #pop() should remove and return the last value: 1ms

  ✔ 3 tests completed (4ms)

  λ mocha (master): 
```

Progress

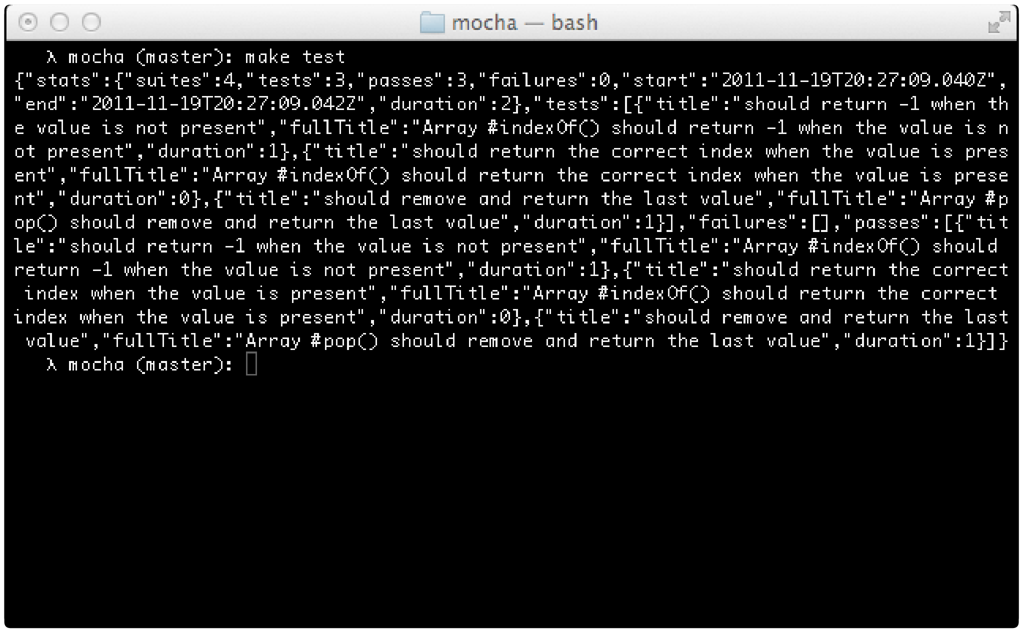
The “progress” reporter implements a simple progress-bar:



```
 λ mocha (master): make test

[━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━]

✔ 32 tests complete (193ms)

 λ mocha (master):
```

JSON

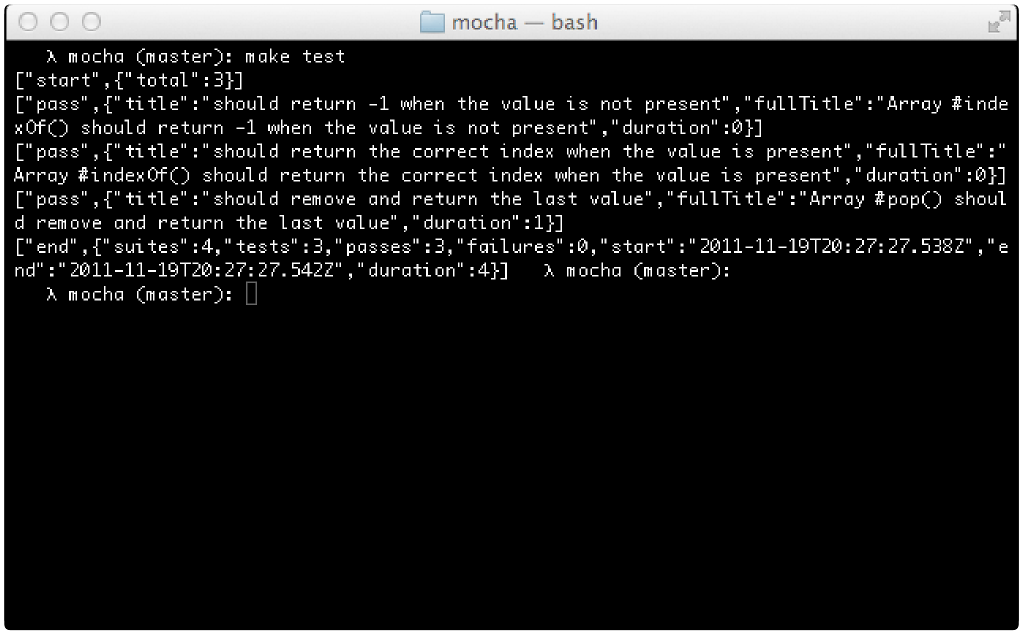
The “JSON” reporter outputs a single large JSON object when the tests have completed (failures or not).



```
   λ mocha (master): make test
{"stats":{"suites":4,"tests":3,"passes":3,"failures":0,"start":"2011-11-19T20:27:09.040Z",
"end":"2011-11-19T20:27:09.042Z","duration":2},"tests":[{"title":"should return -1 when th
e value is not present","fullTitle":"Array #indexOf() should return -1 when the value is n
ot present","duration":1},{"title":"should return the correct index when the value is pres
ent","fullTitle":"Array #indexOf() should return the correct index when the value is prese
nt","duration":0},{"title":"should remove and return the last value","fullTitle":"Array #p
op() should remove and return the last value","duration":1}],"failures":[],"passes":[{"tit
le":"should return -1 when the value is not present","fullTitle":"Array #indexOf() should
return -1 when the value is not present","duration":1},{"title":"should return the correct
 index when the value is present","fullTitle":"Array #indexOf() should return the correct
index when the value is present","duration":0},{"title":"should remove and return the last
 value","fullTitle":"Array #pop() should remove and return the last value","duration":1}]}
   λ mocha (master):
```

JSON Stream

The “JSON stream” reporter outputs newline-delimited JSON “events” as they occur, beginning with a “start” event, followed by test passes or failures, and then the final “end” event.



```
   λ mocha (master): make test
["start",{"total":3}]
["pass",{"title":"should return -1 when the value is not present","fullTitle":"Array #inde
xOf() should return -1 when the value is not present","duration":0}]
["pass",{"title":"should return the correct index when the value is present","fullTitle":"
Array #indexOf() should return the correct index when the value is present","duration":0}]
["pass",{"title":"should remove and return the last value","fullTitle":"Array #pop() shoul
d remove and return the last value","duration":1}]
["end",{"suites":4,"tests":3,"passes":3,"failures":0,"start":"2011-11-19T20:27:27.538Z","e
nd":"2011-11-19T20:27:27.542Z","duration":4}]   λ mocha (master):
   λ mocha (master):
```

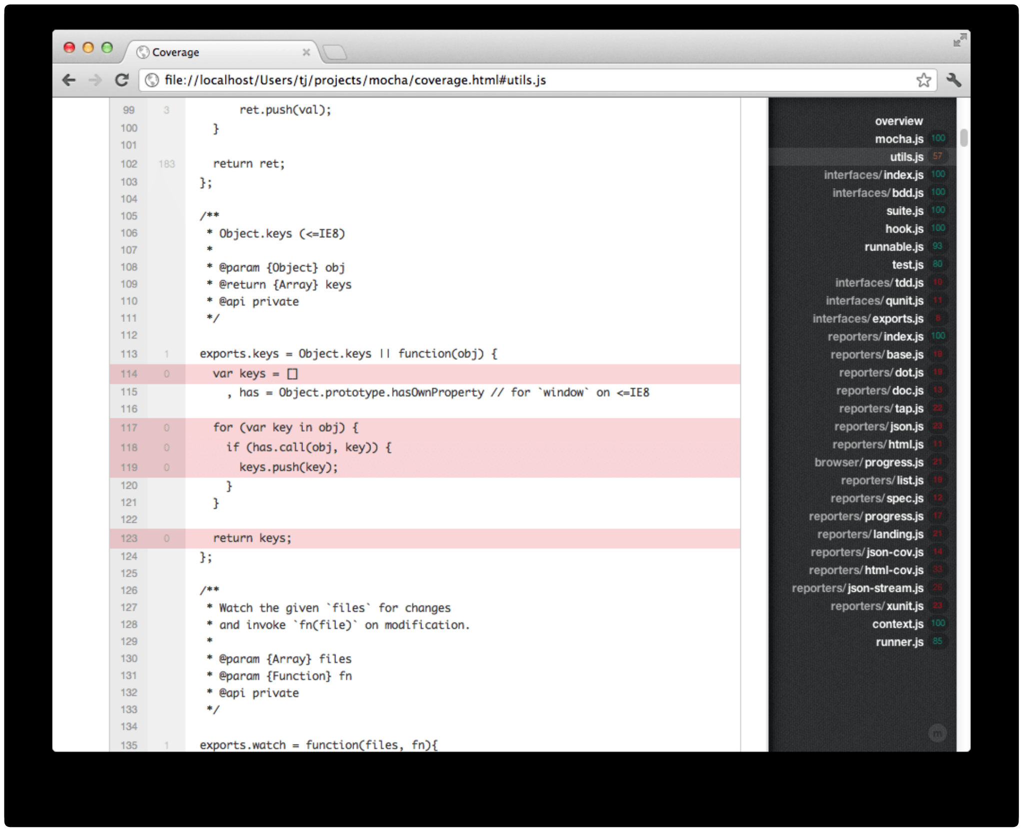
JSONCov

The “JSONCov” reporter is similar to the JSON reporter, however when run against a library instrumented by **node-jscoverage** it will produce coverage output.

HTMLCov

The “HTMLCov” reporter extends the JSONCov reporter. The library being tested should first be instrumented by **node-jscoverage**, this allows Mocha to capture the coverage information necessary to produce a single-page HTML report.

For an integration example, view the mocha test coverage support **commit** for Express.

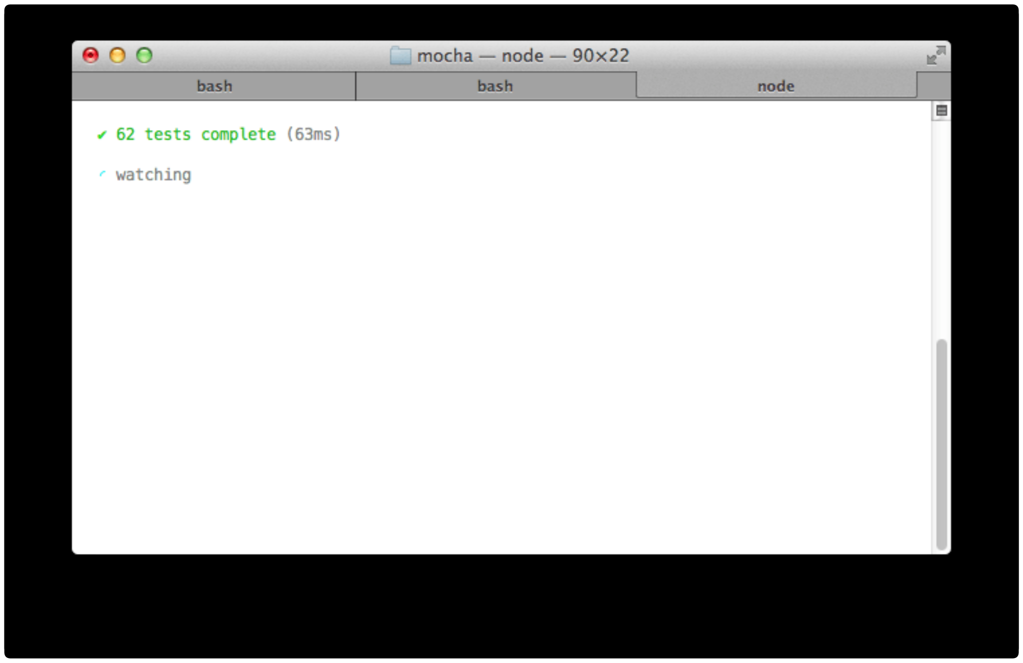

Coverage
file://localhost/Users/tj/projects/mocha/coverage.html#utils.js

```
      ret.push(val);
    }

    return ret;
  };

  /**
   * Object.keys (<=IE8)
   *
   * @param {Object} obj
   * @return {Array} keys
   * @api private
   */

  exports.keys = Object.keys || function(obj) {
    var keys = []
      , has = Object.prototype.hasOwnProperty // for `window` on <=IE8

    for (var key in obj) {
      if (has.call(obj, key)) {
        keys.push(key);
      }
    }

    return keys;
  };

  /**
   * Watch the given `files` for changes
   * and invoke `fn(file)` on modification.
   *
   * @param {Array} files
   * @param {Function} fn
   * @api private
   */

  exports.watch = function(files, fn){
```

```
                       overview
                       mocha.js 100
                       utils.js 57
          interfaces/index.js 100
            interfaces/bdd.js 100
                       suite.js 100
                        hook.js 100
                    runnable.js 93
                        test.js 80
            interfaces/tdd.js 10
          interfaces/qunit.js 11
        interfaces/exports.js 8
           reporters/index.js 100
            reporters/base.js 19
             reporters/dot.js 19
             reporters/doc.js 13
             reporters/tap.js 22
            reporters/json.js 23
            reporters/html.js 11
          browser/progress.js 21
            reporters/list.js 19
            reporters/spec.js 12
        reporters/progress.js 17
         reporters/landing.js 21
        reporters/json-cov.js 14
        reporters/html-cov.js 33
     reporters/json-stream.js 26
           reporters/xunit.js 23
                     context.js 100
                      runner.js 85
```

Min

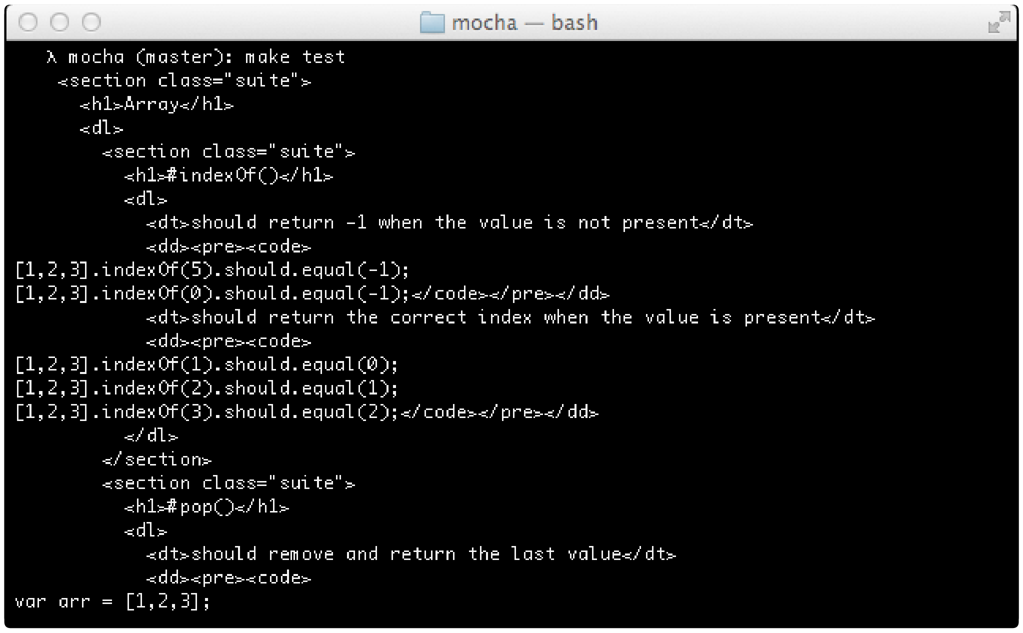
The “min” reporter displays the summary only, while still outputting errors on failure. This reporter works great with `--watch` as it clears the terminal in order to keep your test summary at the top.



```
✔ 62 tests complete (63ms)

◜ watching
```

Doc

The “doc” reporter outputs a hierarchical HTML body representation of your tests. Wrap it with a header, footer, and some styling, then you have some fantastic documentation!



```
  λ mocha (master): make test
    <section class="suite">
      <h1>Array</h1>
      <dl>
        <section class="suite">
          <h1>#indexOf()</h1>
          <dl>
            <dt>should return -1 when the value is not present</dt>
            <dd><pre><code>
[1,2,3].indexOf(5).should.equal(-1);
[1,2,3].indexOf(0).should.equal(-1);</code></pre></dd>
            <dt>should return the correct index when the value is present</dt>
            <dd><pre><code>
[1,2,3].indexOf(1).should.equal(0);
[1,2,3].indexOf(2).should.equal(1);
[1,2,3].indexOf(3).should.equal(2);</code></pre></dd>
          </dl>
        </section>
        <section class="suite">
          <h1>#pop()</h1>
          <dl>
            <dt>should remove and return the last value</dt>
            <dd><pre><code>
var arr = [1,2,3];
```

For example, suppose you have the following JavaScript:

```
describe('Array', function() {
  describe('#indexOf()', function() {
    it('should return -1 when the value is not present',
function() {
      [1,2,3].indexOf(5).should.equal(-1);
      [1,2,3].indexOf(0).should.equal(-1);
    });
  });
});
```

The command `mocha --reporter doc array` would yield:

```
<section class="suite">
  <h1>Array</h1>
  <dl>
    <section class="suite">
      <h1>#indexOf()</h1>
      <dl>
      <dt>should return -1 when the value is not
present</dt>
```

```
      <dd><pre><code>[1,2,3].indexOf(5).should.equal(-1);
[1,2,3].indexOf(0).should.equal(-1);</code></pre></dd>
      </dl>
    </section>
  </dl>
</section>
```

The SuperAgent request library **test documentation** was generated with Mocha's doc reporter using this simple make target:

```
test-docs:
	$(MAKE) test REPORTER=doc \
		| cat docs/head.html - docs/tail.html \
		> docs/test.html
```

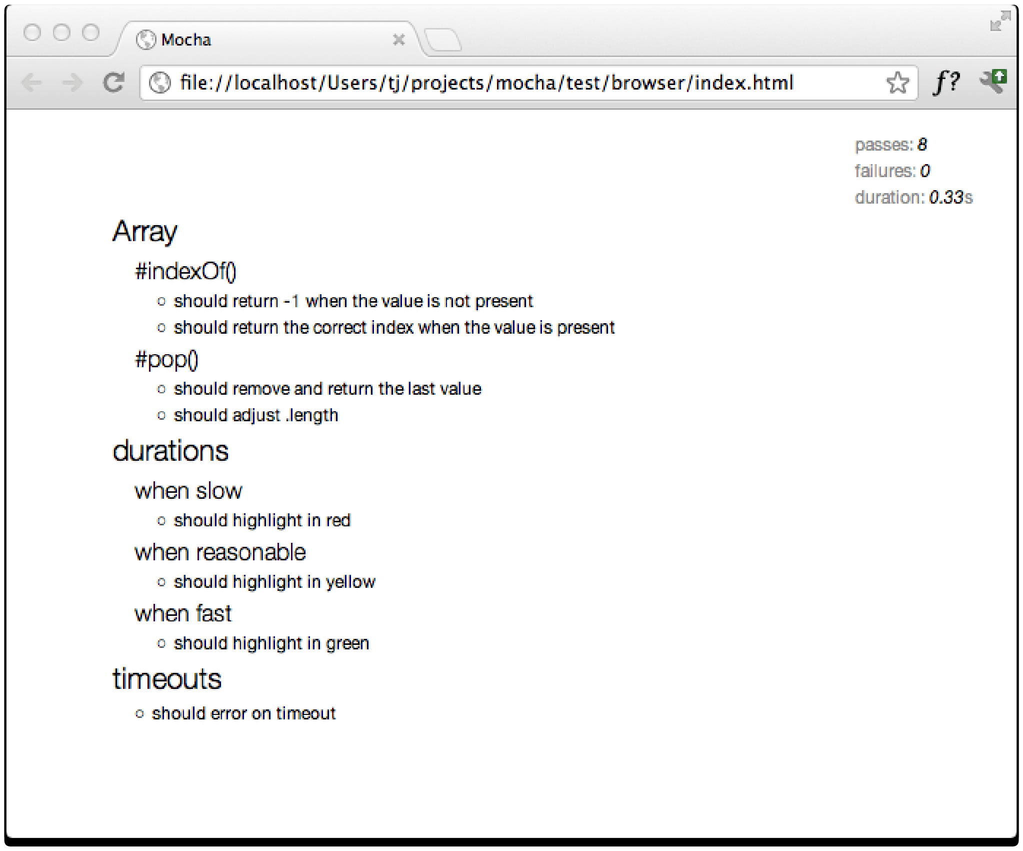
View the entire **Makefile** for reference.

Markdown

The "markdown" reporter generates a markdown TOC and body for your test suite. This is great if you want to use the tests as documentation within a Github wiki page, or a markdown file in the repository that Github can render. For example here is the Connect **test output**.

HTML

The "HTML" reporter is currently the only browser reporter supported by Mocha, and it looks like this:


Mocha
file://localhost/Users/tj/projects/mocha/test/browser/index.html

passes: 8
failures: 0
duration: 0.33s

```
Array
  #indexOf()
    ◦ should return -1 when the value is not present
    ◦ should return the correct index when the value is present
  #pop()
    ◦ should remove and return the last value
    ◦ should adjust .length
durations
  when slow
    ◦ should highlight in red
  when reasonable
    ◦ should highlight in yellow
  when fast
    ◦ should highlight in green
timeouts
  ◦ should error on timeout
```

Undocumented Reporters

The “XUnit” and “TeamCity” reporters are also available, but someone needs to write the documentation.

Running Mocha in the Browser

Mocha runs in the browser. Every release of Mocha will have new builds of `./mocha.js` and `./mocha.css` for use in the browser.

Browser-specific methods

The following method(s) *only* function in a browser context:

`mocha.allowUncaught()` : If called, uncaught errors will not be absorbed by the error handler.

A typical setup might look something like the following, where we call `mocha.setup('bdd')` to use the **BDD** interface before loading the test scripts, running them `onload` with `mocha.run()`.

```
<html>
<head>
  <meta charset="utf-8">
  <title>Mocha Tests</title>
  <link
href="https://cdn.rawgit.com/mochajs/mocha/2.2.5/mocha.css"
rel="stylesheet" />
</head>
<body>
  <div id="mocha"></div>

  <script
src="https://cdn.rawgit.com/jquery/jquery/2.1.4/dist/jquery.mi
</script>
  <script
src="https://cdn.rawgit.com/Automattic/expect.js/0.3.1/index.j
</script>
  <script
src="https://cdn.rawgit.com/mochajs/mocha/2.2.5/mocha.js">
</script>

  <script>mocha.setup('bdd')</script>
  <script src="test.array.js"></script>
  <script src="test.object.js"></script>
  <script src="test.xhr.js"></script>
  <script>
    mocha.checkLeaks();
    mocha.globals(['jQuery']);
    mocha.run();
  </script>
</body>
</html>
```

Grep

The browser may use the `--grep` as functionality. Append a query-string to your URL: `?grep=api`.

Browser Configuration

Mocha options can be set via `mocha.setup()`. Examples:

```
// Use "tdd" interface.  This is a shortcut to setting the
interface;
// any other options must be passed via an object.
mocha.setup('tdd');

// This is equivalent to the above.
mocha.setup({
  ui: 'tdd'
});

// Use "tdd" interface, ignore leaks, and force all tests
to be asynchronous
mocha.setup({
  ui: 'tdd',
  ignoreLeaks: true,
  asyncOnly: true
});
```

Browser-specific Option(s)

The following option(s) *only* function in a browser context:

`noHighlighting`: If set to `true`, do not attempt to use syntax highlighting on output test code.

`mocha.opts`

Back on the server, Mocha will attempt to load `./test/mocha.opts` as a configuration file of sorts. The lines in this file are combined with any command-line arguments. The command-line arguments take precedence. For example, suppose you have the following `mocha.opts` file:

```
--require should
--reporter dot
--ui bdd
```

This will default the reporter to `dot`, require the `should` library, and use `bdd` as the interface. With this, you may then invoke `mocha` with additional arguments, here enabling **Growl** support, and changing the reporter to `list`:

```
$ mocha --reporter list --growl
```

The `test/` Directory

By default, `mocha` looks for the glob `./test/*.js` and `./test/*.coffee`, so you may want to put your tests in `test/` folder.

Editor Plugins

The following editor-related packages are available:

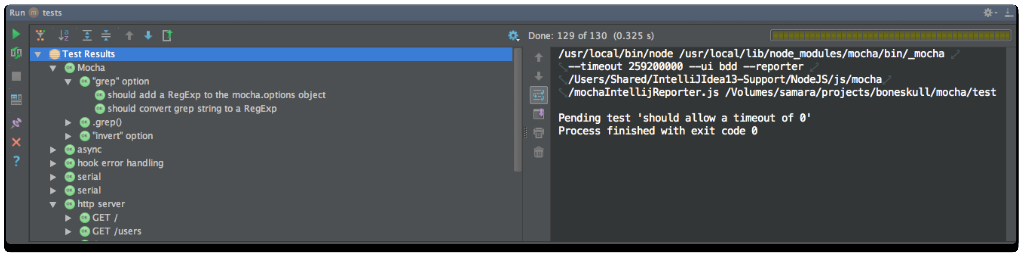
TextMate

The Mocha TextMate bundle includes snippets to make writing tests quicker and more enjoyable. To install the bundle, clone a copy of the **Mocha repo**, and run:

```
$ make tm
```

JetBrains

JetBrains provides a **NodeJS plugin** for its suite of IDEs (IntelliJ IDEA, WebStorm, etc.), which contains a Mocha test runner, among other things.



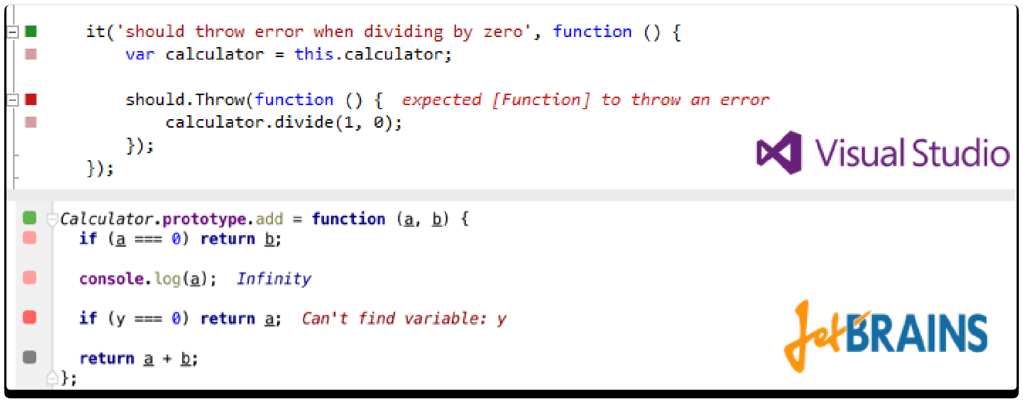
```
Test Results
  Mocha
    "grep" option
      should add a RegExp to the mocha.options object
      should convert grep string to a RegExp
    .grep()
    "invert" option
  async
  hook error handling
  serial
  serial
  http server
    GET /
    GET /users
```

```
/usr/local/bin/node /usr/local/lib/node_modules/mocha/bin/_mocha
 --timeout 259200000 --ui bdd --reporter
 /Users/Shared/IntelliJIdea13-Support/NodeJS/js/mocha
 /mochaIntellijReporter.js /Volumes/samara/projects/boneskull/mocha/test

Pending test 'should allow a timeout of 0'
Process finished with exit code 0
```

The plugin is titled **NodeJS**, and can be installed via **Preferences** > **Plugins**, assuming your license allows it.

Wallaby.js

Wallaby.js is a continuous testing tool that enables real-time code coverage for Mocha with any assertion library in JetBrains IDEs (IntelliJ IDEA, WebStorm, etc.) and Visual Studio for both browser and node.js projects.

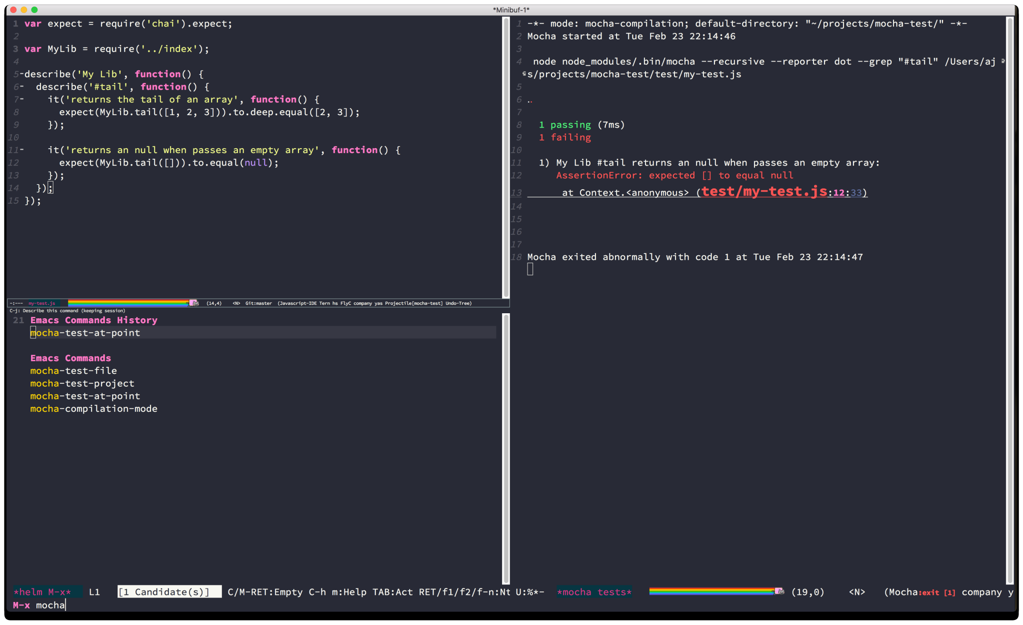


```
it('should throw error when dividing by zero', function () {
    var calculator = this.calculator;

    should.Throw(function () {  expected [Function] to throw an error
        calculator.divide(1, 0);
    });
});
```

```
Calculator.prototype.add = function (a, b) {
  if (a === 0) return b;

  console.log(a);  Infinity

  if (y === 0) return a;  Can't find variable: y

  return a + b;
};
```

Emacs

Emacs support for running Mocha tests is available via a 3rd party package **mocha.el**. The package is available on MELPA, and can be installed via `M-x package-install mocha`.



```
var expect = require('chai').expect;

var MyLib = require('../index');

-describe('My Lib', function() {
-  describe('#tail', function() {
-    it('returns the tail of an array', function() {
       expect(MyLib.tail([1, 2, 3])).to.deep.equal([2, 3]);
     });

-    it('returns an null when passes an empty array', function() {
       expect(MyLib.tail([])).to.equal(null);
     });
   });
 });
```

```
-:---  my-test.js    (14,4)    <N>  Git:master  (Javascript-IDE Tern hs FlyC company yas Projectile[mocha-test] Undo-Tree)
C-j: Describe this command (keeping session)
Emacs Commands History
mocha-test-at-point

Emacs Commands
mocha-test-file
mocha-test-project
mocha-test-at-point
mocha-compilation-mode
```

```
-*- mode: mocha-compilation; default-directory: "~/projects/mocha-test/" -*-
Mocha started at Tue Feb 23 22:14:46

 node node_modules/.bin/mocha --recursive --reporter dot --grep "#tail" /Users/aj
s/projects/mocha-test/test/my-test.js

..

  1 passing (7ms)
  1 failing

  1) My Lib #tail returns an null when passes an empty array:
     AssertionError: expected [] to equal null
      at Context.<anonymous> (test/my-test.js:12:33)




Mocha exited abnormally with code 1 at Tue Feb 23 22:14:47
```

```
*helm M-x*  L1   [1 Candidate(s)]  C/M-RET:Empty C-h m:Help TAB:Act RET/f1/f2/f-n:Nt U:%*-  *mocha tests*    (19,0)    <N>   (Mocha:exit [1] company y
M-x mocha
```

Examples

Real live example code:

Express	**WebSocket.io**
Connect	**Mocha**
SuperAgent	

Testing Mocha

To run Mocha's tests:

```
$ make test
```

Run all tests, including interfaces:

```
$ make test-all
```

Alter the reporter:

```
$ make test REPORTER=list
```

More Information

In addition to chatting with us on **Gitter**, for additional information such as using spies, mocking, and shared behaviours be sure to check out the **Mocha Wiki** on GitHub. For discussions join the **Google Group**. For a running example of Mocha, view **example/tests.html**. For the JavaScript API, view the **source**.