Mocha.js Cheat Sheet

Mocha.js Cheat Sheet tries to provide a basic reference to help you using Mochajs**. 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.

Source: mochajs.org

Mocha.js: Getting Started

Installing

```
$ npm install mocha
$ mkdir test
$ $EDITOR test/test.js # or open with your favorite editor
```

In your editor:

```
var assert = require('assert');
describe('Array', function () {
  describe('#indexOf()', function () {
    it('should return -1 when the value is not present', function () {
      assert.equal([1, 2, 3].indexOf(4), -1);
    });
  });
});
```

Back in the terminal:

```
$ ./node_modules/mocha/bin/mocha

  Array
    #indexOf()
      ✓ should return -1 when the value is not present


  1 passing (9ms)
```

Set up a test script in package.json:

```
"scripts": {
  "test": "mocha"
}
```

Then run tests with:

```
$ npm test
```

Mocha.js: Run Cycle Overview

A brief outline on the order Mocha's components are executed. Worth noting that all hooks, `describe` and `it` callbacks are run in the order they are defined (i.e. found in the file).

```
run 'mocha spec.js'
|
spawn child process
```

```
|
|--------------> inside child process
  process and apply options
   |
   run spec file/s
   |
   |--------------> per spec file
     suite callbacks (e.g., 'describe')
      |
      'before' root-level pre-hook
      |
      'before' pre-hook
      |
      |--------------> per test
        'beforeEach' root-level pre-hook
        |
        'beforeEach' pre-hook
        |
        test callbacks (e.g., 'it')
        |
        'afterEach' post-hook
        |
        'afterEach' root-level post-hook
      |<-------------- per test end
      |
      'after' post-hook
      |
      'after' root-level post-hooks
    |<-------------- per spec file end
|<-------------- inside child process end
```

Mocha.js: before, beforeEach, after, afterEach

You can use before these to setup and restore any test helpers or components. You can even call these agains in nested describe.

```
describe('My feature test', function() {
    let classToTest;
    beforeEach(function() {
        classToTest = new ClassToTest();
    });

    afterEach(function() {
        classToTest = null;
    });
});
```

Mocha.js: Only and Skip

You use it to perform a test, it allows `it.only` will ensure that only this test will executed, `it.skip` will skip the test.

```
it.only('should be able to get valueA', function() {
    //..
});

it.skip('this test is going to be skipped', function() {
    //..
});
```


Mocha.js: Root Hook Plugins Cheat Sheet

Defining

A Root Hook Plugin file is a script which exports (via `module.exports`) a `mochaHooks` property. It is loaded via `--require <file>`.

With CommonJS

```
exports.mochaHooks = {
```

```
  beforeEach(done) {
    // do something before every test
    done();
  },
};
```

With ES Modules

```
export const mochaHooks = {
  beforeEach(done) {
    // do something before every test
    done();
  },
};
```

Available Root Hooks

`beforeAll`:

In **serial** mode (Mocha's default), *before all tests begin, once only*

In **parallel** mode, run *before all tests begin, for each file*

`beforeEach`:

In **both** modes, run *before each test*

`afterAll`:

In **serial** mode, run *after all tests end, once only*

In **parallel** mode, run *after all tests end, for each file*

`afterEach`:

In **both** modes, run *after every test*

Multiple Root Hooks In A Single Plugin

```
export const mochaHooks = {
  beforeEach: [
    function (done) {
      // do something before every test,
      // then run the next hook in this array
    },
    async function () {
      // async or Promise-returning functions allowed
    },
  ],
};
```

Multiple Root Hook Plugins

Multiple root hook plugins can be registered by using `--require` multiple times.

Migrating Tests To Use Root Hook Plugins

1. Find your root hooks (hooks defined *outside* of a suite–usually `describe()` callback).
2. Create a new file, e.g., `test/hooks.js`.
3. *Move* your root hooks into `test/hooks.js`.
4. In `test/hooks.js`, make your hooks a member of an exported `mochaHooks` property.
5. Use `--require test/hooks.js` (even better: use a config file with `{"require": "test/hooks.js"}`) when running your tests.

Mocha.js: Reporters Cheat Sheet

Spec

Alias: `Spec`, `spec`

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

Dot Matrix

Alias: `Dot`, `dot`

The Dot Matrix reporter is a series of characters which represent test cases. Failures highlight in red exclamation marks (`!`), pending tests with a blue comma (`,`), and slow tests as yellow. Good if you prefer minimal output.

Nyan

Alias: `Nyan`, `nyan`

Tap

Alias: `TAP`, `tap`

Landing Strip

Alias: `Landing`, `landing`

List

Alias: `List`, `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.

Progress

Alias: `Progress`, `progress`

The Progress reporter implements a simple progress-bar

Json

Alias: `JSON`, `json`

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

Json Stream

Alias: `JSONStream`, `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.

Min

Alias: `Min`, `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.

Doc

Alias: `Doc`, `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!

Markdown

Alias: `Markdown`, `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.

XUnit

Alias: `XUnit`, `xunit`

The XUnit reporter is also available. It outputs an XUnit-compatible XML document, often applicable in CI servers.

By default, it will output to the console. To write directly to a file, use `--reporter-option output=filename.xml`.

To specify a custom report title, use `--reporter-option suiteName="Custom name"`.

Html Reporter

Alias: `HTML`, `html`

The HTML reporter is not intended for use on the command-line.

Mocha.js: Interfaces Cheat Sheet

BDD

The **BDD** interface provides:

`describe()`

`context()`

`it()`

`specify()`

`before()`

`after()`

`beforeEach()`

`afterEach()`

TDD

The **TDD** interface provides:

`suite()`

`test()`

`suiteSetup()`

`suiteTeardown()`

`setup()`

`teardown()`

Exports

The keys `before`, `after`, `beforeEach`, and `afterEach` are special-cased, object values are suites, and function values.

QUnit

he 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()`

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.

Mocha.js: Running in the Browser Cheat Sheet

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://unpkg.com/mocha@5.2.0/mocha.css" rel="stylesheet" />
</head>
<body>
  <div id="mocha"></div>

  <script src="https://unpkg.com/chai/chai.js"></script>
  <script src="https://unpkg.com/mocha@5.2.0/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.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()`

Mocha.js: Desktop Notification Cheat Sheet

Node-based notifications

Enable Mocha's desktop notifications:

```
$ mocha --growl
```

Browser-based notifications

Enable Mocha's web notifications with a slight modification to your client-side mocha HTML. Add a call to `mocha.growl()` prior to running your tests:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
```

```
    <title>Mocha Tests</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="https://unpkg.com/mocha/mocha.css" />
  </head>
  <body>
    <div id="mocha"></div>

    <script src="https://unpkg.com/chai/chai.js"></script>
    <script src="https://unpkg.com/mocha/mocha.js"></script>

    <script class="mocha-init">
      mocha.setup('bdd');
      mocha.growl(); // <-- Enables web notifications
    </script>
    <script src="test.spec.js"></script>
    <script class="mocha-exec">
      mocha.run();
    </script>
  </body>
</html>
```

Mocha.js: The test/ Directory

By default, `mocha` looks for the glob `"./test/*.js"`, so you may want to put your tests in `test/` folder. If you want to include subdirectories, pass the `--recursive` option.

To configure where `mocha` looks for tests, you may pass your own glob:

```
$ mocha --recursive "./spec/*.js"
```

Some shells support recursive matching by using the globstar (`**`) wildcard. Bash >= 4.3 supports this with the `globstar` option which must be enabled to get the same results as passing the `--recursive` option (ZSH and Fish support this by default). With recursive matching enabled, the following is the same as passing `--recursive`:

```
$ mocha "./spec/**/*.js"
```

Note: Double quotes around the glob are recommended for portability.

Mocha.js: Error Codes Cheat Sheet

Code	Description
`ERR_MOCHA_INVALID_ARG_TYPE`	wrong type was passed for a given argument
`ERR_MOCHA_INVALID_ARG_VALUE`	invalid or unsupported value was passed for a given argument
`ERR_MOCHA_INVALID_EXCEPTION`	a falsy or otherwise underspecified exception was thrown
`ERR_MOCHA_INVALID_INTERFACE`	interface specified in options not found
`ERR_MOCHA_INVALID_REPORTER`	reporter specified in options not found
`ERR_MOCHA_NO_FILES_MATCH_PATTERN`	test file(s) could not be found
`ERR_MOCHA_UNSUPPORTED`	requested behavior, option, or parameter is unsupported

Mocha.js: Timeouts Cheat Sheet

Suite-level

```
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

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

Hook-level

```
describe('a suite of tests', function() {
  beforeEach(function(done) {
    this.timeout(3000); // A very long environment setup.
    setTimeout(done, 2500);
  });
});
```

Mocha.js: Usage Cheat Sheet

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

Option	Description
`-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 (default: "spec")
`-S, --sort`	sort test files
`-b, --bail`	bail after first test failure
`-d, --debug`	enable node's debugger, a 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 (default: [])
`-s, --slow <ms>`	specify "slow" test threshold in milliseconds (default: 75)
`-t, --timeout <ms>`	specify the test timeout threshold in milliseconds (default: 2000)
`-u, --ui <name>`	specify user-interface (bdd|tdd|qunit|exports) (default: "bdd")
`-w, --watch`	watch files in the current working directory for changes
`--check-leaks`	check for global variable leaks
`--full-trace`	display the full stack trace
`--compilers <ext>:<module>,...`	use the given module(s) to compile files (default: [])
`--debug-brk`	enable node's debugger breaking on the first line
`--globals <names>`	allow the given comma-delimited global [names] (default: [])
`--es_staging`	enable all staged features
`--harmony<_classes,_generators,...>`	all node `--harmony*` flags are available
`--preserve-symlinks`	Instructs the module loader to preserve symbolic links when resolving and caching modules
`--icu-data-dir`	include ICU data
	display actual/expected differences inline within each

`--inline-diffs`	string
`--no-diff`	do not show a diff on failure
`--inspect`	activate devtools in chrome
`--inspect-brk`	activate devtools in chrome and break on the first line
`--interfaces`	output provided interfaces and exit
`--no-deprecation`	silence deprecation warnings
`--exit force`	shutdown of the event loop after test run: mocha will call process.exit
`--no-timeouts`	disables timeouts, given implicitly with –debug/–inspect
`--no-warnings`	silence all node process warnings
`--opts <path>`	specify opts path (default: “test/mocha.opts”)
`--perf-basic-prof`	enable perf linux profiler (basic support)
`--napi-modules`	enable experimental NAPI modules
`--prof`	log statistical profiling information
`--log-timer-events`	Time events including external callbacks
`--recursive`	include sub directories
`--reporters`	output provided reporters and exit
`--retries <times>`	specify number of times to retry a failed test case (default: 0)
`--throw-deprecation`	throw an exception anytime a deprecated function is used
`--trace`	trace function calls
`--trace-deprecation`	show stack traces on deprecations
`--trace-warnings`	show stack traces on node process warnings
`--use_strict`	enforce strict mode
`--watch-extensions <ext>,...`	specify extensions to monitor with –watch (default: [“js”])
`--delay`	wait for async suite definition
`--allow-uncaught`	enable uncaught errors to propagate
`--forbid-only`	causes test marked with only to fail the suite
`--forbid-pending`	causes pending tests and test marked with skip to fail the suite
`--file <file>`	adds file be loaded prior to suite execution (default: [])
`--exclude <file>`	adds file or glob pattern to ignore (default: [])
`-h, --help`	output usage information

Mocha.js: Testing Mocha

To run Mocha's tests, you will need GNU Make or compatible; Cygwin should work.

```
$ cd /path/to/mocha
$ npm install
$ npm test
```

To use a different reporter:

```
$ REPORTER=nyan npm test
```

Database Cheat Sheet | Docker Compose Cheatsheet | Heroku Cheat Sheet | Apache Cheat Sheet | BabylonJS Cheat Sheet
Vue.js Cheat Sheet | MongoDB Cheat Sheet | C# Cheat Sheet | Emoji Cheat Sheet | DB2 Cheat Sheet | Nano Cheat Sheet - Mac
Python Cheat Sheet | WebStorm Cheat Sheet | Axios Cheat Sheet | VS Code Cheat Sheet | GraphQL Cheat Sheet
Org-mode Cheat Sheet | Swift Cheat Sheet | MySQL SQL Injection Cheat Sheet | Ack Cheat Sheet

Search ... SEARCH

To the top ↑