

# Node.js Test Runner

Why is it game-changing?

# \$whoami

Senior Software **engineer**  
**Klarna.**



<https://{{twitter,instagram/github,youtube,linkedin}}.lsantos.dev>



**WHAT IS MY  
OBJECTIVE HERE?**

# WHAT IS MY OBJECTIVE HERE?

make you ditch Jest



# WHAT IS MY OBJECTIVE HERE?

make you ditch Jest

yes, it's that simple...



**and how am I GOING TO DO THAT?**

**and how am I GOING TO DO THAT?**

**by showing you something (possibly) better**

# WHY "POSSIBLY"?

# WHY "POSSIBLY"?

because the Node.js test runner is still being improved



# What's bad about current runners?

# What's bad about CURRENT RUNNERS?

# What's bad about CURRENT RUNNERS?

1. They are difficult to configure, especially for TypeScript

# What's bad about CURRENT RUNNERS?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow

# What's bad about CURRENT RUNNERS?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project

# WHAT'S BAD ABOUT CURRENT RUNNERS?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project
4. Test runners seem to not interoperate well with each other, making it hard to switch

# WHAT'S BAD ABOUT CURRENT RUNNERS?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project
4. Test runners seem to not interoperate well with each other, making it hard to switch
5. Most of them are pretty opinionated, some of them are just old



# WHAT'S BAD ABOUT CURRENT RUNNERS?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project
4. Test runners seem to not interoperate well with each other, making it hard to switch
5. Most of them are pretty opinionated, some of them are just old
6. Want TypeScript, but don't want to use ts-jest? Good luck!

# WHAT'S BAD ABOUT CURRENT RUNNERS?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project
4. Test runners seem to not interoperate well with each other, making it hard to switch
5. Most of them are pretty opinionated, some of them are just old
6. Want TypeScript, but don't want to use ts-jest? Good luck!
  1. Otherwise, **YAL**

# WHAT'S BAD ABOUT CURRENT RUNNERS?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project
4. Test runners seem to not interoperate well with each other, making it hard to switch
5. Most of them are pretty opinionated, some of them are just old
6. Want TypeScript, but don't want to use ts-jest? Good luck!
  1. Otherwise, **YAL**
7. There are SO MANY of them, it's hard to choose



EntERS THE Node.js Test

Runner\_

# IT'S NOT THAT new

## test\_runner: add initial CLI runner #42658

[Edit](#)[Code](#)**Merged**nodejs-github-bot merged 1 commit into nodejs:master from cjihrig:test-cli  on Apr 15, 2022[Conversation 73](#)[Commits 1](#)[Checks 0](#)[Files changed 23](#)[+669 -128](#) 

cjihrig commented on Apr 8, 2022

Member

...

This commit introduces an initial version of a CLI-based test runner.



11



nodejs-github-bot commented on Apr 8, 2022

Member

...

Review requested:

### Reviewers

juliangruber

ljharb

atian25

jasnell

bnb

targos

aduh95



# IMPROVEMENTS CAME SOON AFTER

## test\_runner: add TAP parser #43525

Edit

< > Code ▾

↳ Merged

nodejs-github-bot merged 98 commits into nodejs:main from manekinekko:tap-14-parser on Nov 22, 2022

Conversation 303

Commits 98

Checks 27

Files changed 19

+4,418 -31



manekinekko commented on Jun 21, 2022 ·  
edited

Contributor

...

This PR adds initial support for a TAP LL(1) parser. This implementation is based on the grammar for TAP14 from <https://testanything.org/tap-version-14-specification.html>

TODO:

- add a TAP checker (by design, the current parser does only parsing).

Reviewers



Trott



aduh95



kmannislands



Mifrill



fhinkel



cjihrig



# IT was soon STABLE

## Request: mark test runner stable in Node 20.0.0 #46642

Closed

19 comments · Fixed by #46983



cjihrig commented on Feb 13, 2023

Contributor

...

### What is the problem this feature will solve?

Having a stable test runner in core. People have been asking for it, and I feel the majority of the test runner is ready.

### What is the feature you are proposing to solve the problem?

I would like to stabilize the test runner for the 20.0.0 release. This currently would not include code coverage, which is behind a separate flag.

### What alternatives have you considered?

Not stabilizing the test runner.



32



# WHY IS IT DIFFERENT?

# WHY IS IT DIFFERENT?

- It's built-in, so no **YAL**

# WHY IS IT DIFFERENT?

- It's built-in, so no **YAL**
- It's fast

# WHY IS IT DIFFERENT?

- It's built-in, so no **YAL**
- It's fast
- No configuration needed, integrates seamlessly with Node.js

# WHY IS IT DIFFERENT?

- It's built-in, so no **YAL**
- It's fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library

# WHY IS IT DIFFERENT?

- It's built-in, so no **YAL**
- It's fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it's easy to integrate with other runners

# WHY IS IT DIFFERENT?

- It's built-in, so no **YAL**
- It's fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it's easy to integrate with other runners
- Always use up to date Node.js features

# WHY IS IT DIFFERENT?

- It's built-in, so no **YAL**
- It's fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it's easy to integrate with other runners
- Always use up to date Node.js features
- Consistency across projects

# WHY IS IT DIFFERENT?

- It's built-in, so no **YAL**
- It's fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it's easy to integrate with other runners
- Always use up to date Node.js features
- Consistency across projects
- No polyfills needed

# WHY IS IT DIFFERENT?

- It's built-in, so no **YAL**
- It's fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it's easy to integrate with other runners
- Always use up to date Node.js features
- Consistency across projects
- No polyfills needed
- Experimental support for code coverage and other reporters



# WHY IS IT DIFFERENT?

- It's built-in, so no **YAL**
- It's fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it's easy to integrate with other runners
- Always use up to date Node.js features
- Consistency across projects
- No polyfills needed
- Experimental support for code coverage and other reporters
- Supports TS out of the box



# GETTING STARTED

You don't need anything to get started. Import `node:test` and `node:assert` and you are good to go!

```
1 // @filename: index.js
2 export function sum(a, b) {
3     if (typeof a !== 'number' || typeof b !== 'number') {
4         throw new Error('Invalid input')
5     }
6     return a + b
7 }
8
9 // @filename: test.js
10 import { describe, it } from 'node:test'
11 import assert from 'node:assert'
12 import { sum } from './index.js'
13
14 describe('sum', () => {
15     it('should add two numbers', () => {
16         assert.strictEqual(sum(1, 2), 3)
17     })
18 })
19
```



# GETTING STARTED

You don't need anything to get started. Import `node:test` and `node:assert` and you are good to go!

```
1 // @filename: index.js
2 export function sum(a, b) {
3     if (typeof a !== 'number' || typeof b !== 'number') {
4         throw new Error('Invalid input')
5     }
6     return a + b
7 }
8
9 // @filename: test.js
10 import { describe, it } from 'node:test'
11 import assert from 'node:assert'
12 import { sum } from './index.js'
13
14 describe('sum', () => {
15     it('should add two numbers', () => {
16         assert.strictEqual(sum(1, 2), 3)
17     })
18 })
19
```



# GETTING STARTED

You don't need anything to get started. Import `node:test` and `node:assert` and you are good to go!

```
1 // @filename: index.js
2 export function sum(a, b) {
3     if (typeof a !== 'number' || typeof b !== 'number') {
4         throw new Error('Invalid input')
5     }
6     return a + b
7 }
8
9 // @filename: test.js
10 import { describe, it } from 'node:test'
11 import assert from 'node:assert'
12 import { sum } from './index.js'
13
14 describe('sum', () => {
15     it('should add two numbers', () => {
16         assert.strictEqual(sum(1, 2), 3)
17     })
18 })
19
```



# GETTING STARTED

You don't need anything to get started. Import `node:test` and `node:assert` and you are good to go!

```
1 // @filename: index.js
2 export function sum(a, b) {
3     if (typeof a !== 'number' || typeof b !== 'number') {
4         throw new Error('Invalid input')
5     }
6     return a + b
7 }
8
9 // @filename: test.js
10 import { describe, it } from 'node:test'
11 import assert from 'node:assert'
12 import { sum } from './index.js'
13
14 describe('sum', () => {
15     it('should add two numbers', () => {
16         assert.strictEqual(sum(1, 2), 3)
17     })
18 })
19
```

Let's break it down...



WRITING **tests\_**

# WRITING TESTS

```
1 import { describe, it } from 'node:test'  
2 import assert from 'node:assert'  
3 import { sum } from './index.js'  
4  
5 describe('sum', () => {  
6     it('should add two numbers', () => {  
7         assert.strictEqual(sum(1, 2), 3)  
8     })  
9 })  
10
```



# WRITING TESTS

```
1 import { describe, it } from 'node:test'  
2 import assert from 'node:assert'  
3 import { sum } from './index.js'  
4  
5 describe('sum', () => {  
6     it('should add two numbers', () => {  
7         assert.strictEqual(sum(1, 2), 3)  
8     })  
9 })  
10
```



# WRITING TESTS

```
1 import { describe, it } from 'node:test'  
2 import assert from 'node:assert'  
3 import { sum } from './index.js'  
4  
5 describe('sum', () => {  
6     it('should add two numbers', () => {  
7         assert.strictEqual(sum(1, 2), 3)  
8     })  
9 })  
10
```



# WRITING TESTS

```
1 import { describe, it } from 'node:test'  
2 import assert from 'node:assert'  
3 import { sum } from './index.js'  
4  
5 describe('sum', () => {  
6     it('should add two numbers', () => {  
7         assert.strictEqual(sum(1, 2), 3)  
8     })  
9 })  
10
```



# WRITING TESTS

```
1 import { describe, it } from 'node:test'  
2 import assert from 'node:assert'  
3 import { sum } from './index.js'  
4  
5 describe('sum', () => {  
6     it('should add two numbers', () => {  
7         assert.strictEqual(sum(1, 2), 3)  
8     })  
9 })  
10
```

Or you can use the `test` function

```
1 import test from 'node:test'  
2 import assert from 'node:assert'  
3  
4 test('sum', (t) => {  
5     t.test('should add two numbers', () => {  
6         assert.strictEqual(sum(1, 2), 3)  
7     })  
8 })  
9
```



# Assertions

Assertions are taken care of by the `assert` module, which is built-in

But you can use any other assertion library, like `chai` or `should`

```
1 import { describe, it } from 'node:test'  
2 import { expect } from 'chai'  
3  
4 describe('sum', () => {  
5     it('should add two numbers', () => {  
6         expect(sum(1, 2)).to.equal(3)  
7     })  
8 })  
9
```



# Assertions

Assertions are taken care of by the `assert` module, which is built-in

But you can use any other assertion library, like `chai` or `should`

```
1 import { describe, it } from 'node:test'  
2 import { expect } from 'chai'  
3  
4 describe('sum', () => {  
5   it('should add two numbers', () => {  
6     expect(sum(1, 2)).to.equal(3)  
7   })  
8 })  
9
```

**\_the important thing:** assertions are not tied to the runner



# Running\_

# RUNNING TESTS

Node has a built-in `--test` command that you can use to run your tests

```
1 node --test test.js  
2
```



# RUNNING TESTS

Node has a built-in `--test` command that you can use to run your tests

```
1 node --test test.js  
2
```

This will give you a TAP output:

```
1 > node --test test.js  
2   ► sum  
3     ✓ should add two numbers (0.130208ms)  
4   ► sum (0.949ms)  
5  
6   i tests 1  
7   i suites 1  
8   i pass 1  
9   i fail 0  
10  i cancelled 0  
11  i skipped 0  
12  i todo 0  
13  i duration_ms 49.5135  
14
```



# RUNNING TESTS

Node has a built-in `--test` command that you can use to run your tests

```
1 node --test test.js  
2
```

This will give you a TAP output:

```
1 > node --test test.js  
2   ▶ sum  
3     ✓ should add two numbers (0.130208ms)  
4   ▶ sum (0.949ms)  
5  
6   i tests 1  
7   i suites 1  
8   i pass 1  
9   i fail 0  
10  i cancelled 0  
11  i skipped 0  
12  i todo 0  
13  i duration_ms 49.5135  
14
```



# RUNNING TESTS

Node has a built-in `--test` command that you can use to run your tests

```
1 node --test test.js  
2
```

This will give you a TAP output:

```
1 > node --test test.js  
2   ▶ sum  
3     ✓ should add two numbers (0.130208ms)  
4   ▶ sum (0.949ms)  
5  
6   i tests 1  
7   i suites 1  
8   i pass 1  
9   i fail 0  
10  i cancelled 0  
11  i skipped 0  
12  i todo 0  
13  i duration_ms 49.5135  
14
```



# RUNNING TESTS

Node has a built-in `--test` command that you can use to run your tests

```
1 node --test test.js  
2
```

This will give you a TAP output:

```
1 > node --test test.js  
2   ▶ sum  
3     ✓ should add two numbers (0.130208ms)  
4   ▶ sum (0.949ms)  
5  
6   i tests 1  
7   i suites 1  
8   i pass 1  
9   i fail 0  
10  i cancelled 0  
11  i skipped 0  
12  i todo 0  
13  i duration_ms 49.5135  
14
```



# Mocks\_



# MOCKING

Today, Node support mocking of some built-in structures:

```
1 import test from 'node:test'  
2  
3 test('mocks', (ctx) => {  
4   ctx.mock.fn() // mock function  
5   ctx.mock.getter() // mock getter  
6   ctx.mock.setter() // mock setter  
7   ctx.mock.method() // mock method  
8 })  
9
```



# MOCKING

Today, Node support mocking of some built-in structures:

```
1 import test from 'node:test'  
2  
3 test('mocks', (ctx) => {  
4   ctx.mock.fn() // mock function  
5   ctx.mock.getter() // mock getter  
6   ctx.mock.setter() // mock setter  
7   ctx.mock.method() // mock method  
8 })  
9
```



# MOCKING

Today, Node support mocking of some built-in structures:

```
1 import test from 'node:test'  
2  
3 test('mocks', (ctx) => {  
4     ctx.mock.fn() // mock function  
5     ctx.mock.getter() // mock getter  
6     ctx.mock.setter() // mock setter  
7     ctx.mock.method() // mock method  
8 })  
9
```



# MOCKING

Today, Node support mocking of some built-in structures:

```
1 import test from 'node:test'  
2  
3 test('mocks', (ctx) => {  
4     ctx.mock.fn() // mock function  
5     ctx.mock.getter() // mock getter  
6     ctx.mock.setter() // mock setter  
7     ctx.mock.method() // mock method  
8 })  
9
```



# MOCKING

Today, Node support mocking of some built-in structures:

```
1 import test from 'node:test'  
2  
3 test('mocks', (ctx) => {  
4     ctx.mock.fn() // mock function  
5     ctx.mock.getter() // mock getter  
6     ctx.mock.setter() // mock setter  
7     ctx.mock.method() // mock method  
8 })  
9
```

It also allows you to mock two more complex structures



# Mocking

Timers ([link](#))

**test\_runner: introduces a new MockTimers API**  
#47775

Merged

nodejs-github-bot merged 57 commits into `nodejs:main` from `ErickWendel:test_runner/introduce-fake-timers` on Jun 22

Conversation 167 Commits 57 Checks 54 Files changed 6 +1,444 -2

ErickWendel commented on Apr 29 · edited Member ...

This PR introduces a new FakeTimers API for the native Node.js test runner.

I'm opening it for review as I've implemented all features for this initial version (it's still missing docs but I think we can start moving forward while I write them)

### Backlog

- ✓ - accept a list of "Which timers/things do I mock?"
- ✓ - add an experimental warning
- ✓ - tests
- ✓ - 100% code coverage

Reviewers

- ijharb ✓
- fatso83
- tniessen
- benjamingr ✓
- MoLow ✓
- aduh95 ●
- cjihrig ●

Assignees

- ErickWendel



# Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
1 import assert from 'node:assert'
2 import { describe, it } from 'node:test'
3
4 describe('setTimeout', () => {
5   it('should call the callback', (ctx) => {
6     const fn = ctx.mock.fn() // look ma, no jest!
7     ctx.mock.timers.enable({ apis: ['setTimeout'] })
8     setTimeout(fn, 99999)
9     assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11    ctx.mock.timers.tick(99999) // advance the timer
12    assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14    ctx.mock.timers.reset() // reset back to original state
15  })
16})
17
```



# Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
1 import assert from 'node:assert'
2 import { describe, it } from 'node:test'
3
4 describe('setTimeout', () => {
5   it('should call the callback', (ctx) => {
6     const fn = ctx.mock.fn() // look ma, no jest!
7     ctx.mock.timers.enable({ apis: ['setTimeout'] })
8     setTimeout(fn, 99999)
9     assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11    ctx.mock.timers.tick(99999) // advance the timer
12    assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14    ctx.mock.timers.reset() // reset back to original state
15  })
16})
17
```



# Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
1 import assert from 'node:assert'
2 import { describe, it } from 'node:test'
3
4 describe('setTimeout', () => {
5   it('should call the callback', (ctx) => {
6     const fn = ctx.mock.fn() // look ma, no jest!
7     ctx.mock.timers.enable({ apis: ['setTimeout'] })
8     setTimeout(fn, 99999)
9     assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11    ctx.mock.timers.tick(99999) // advance the timer
12    assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14    ctx.mock.timers.reset() // reset back to original state
15  })
16})
17
```



# Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
1 import assert from 'node:assert'
2 import { describe, it } from 'node:test'
3
4 describe('setTimeout', () => {
5   it('should call the callback', (ctx) => {
6     const fn = ctx.mock.fn() // look ma, no jest!
7     ctx.mock.timers.enable({ apis: ['setTimeout'] })
8     setTimeout(fn, 99999)
9     assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11    ctx.mock.timers.tick(99999) // advance the timer
12    assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14    ctx.mock.timers.reset() // reset back to original state
15  })
16})
17
```



# Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
1 import assert from 'node:assert'
2 import { describe, it } from 'node:test'
3
4 describe('setTimeout', () => {
5   it('should call the callback', (ctx) => {
6     const fn = ctx.mock.fn() // look ma, no jest!
7     ctx.mock.timers.enable({ apis: ['setTimeout'] })
8     setTimeout(fn, 99999)
9     assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11    ctx.mock.timers.tick(99999) // advance the timer
12    assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14    ctx.mock.timers.reset() // reset back to original state
15  })
16})
17
```



# Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
1 import assert from 'node:assert'
2 import { describe, it } from 'node:test'
3
4 describe('setTimeout', () => {
5   it('should call the callback', (ctx) => {
6     const fn = ctx.mock.fn() // look ma, no jest!
7     ctx.mock.timers.enable({ apis: ['setTimeout'] })
8     setTimeout(fn, 99999)
9     assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11    ctx.mock.timers.tick(99999) // advance the timer
12    assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14    ctx.mock.timers.reset() // reset back to original state
15  })
16})
17
```



# Mock Timers

You can also import `[mock]` directly from `[node:test]`

```
1  import assert from 'node:assert'
2  import { describe, it, mock } from 'node:test'
3
4  describe('setTimeout', () => {
5    it('should call the callback', (ctx) => {
6      const fn = mock.fn() // look ma, no jest!
7      mock.timers.enable({ apis: ['setTimeout'] })
8      setTimeout(fn, 99999)
9      assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11     mock.timers.tick(99999) // advance the timer
12     assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14     mock.timers.reset() // reset back to original state
15   })
16 })
17 }
```



# Mock Timers

You can also import `[mock]` directly from `[node:test]`

```
1 import assert from 'node:assert'
2 import { describe, it, mock } from 'node:test'
3
4 describe('setTimeout', () => {
5   it('should call the callback', (ctx) => {
6     const fn = mock.fn() // look ma, no jest!
7     mock.timers.enable({ apis: ['setTimeout'] })
8     setTimeout(fn, 99999)
9     assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11    mock.timers.tick(99999) // advance the timer
12    assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14    mock.timers.reset() // reset back to original state
15  })
16})
17
```



# Date mocking

Dates (by yours truly) ([link](#))

**test\_runner: Add Date to the supported mock APIs #48638**

Merged by nodejs-github-bot nodejs:main ← khaosdoctor:test\_runner/introduce\_improve\_fake\_timers on Oct 23, 2023

Conversation 191 Commits 4 Checks 25 Files changed 4

 **khaosdoctor** commented on Jul 2, 2023 · edited

This builds on top of [@ErickWendel's #47775](#), I saw the next steps would be to implement the mock timers for Date.now (and thus, the Date object) and performance.now.

This PR implements the Date.now mock, I'll also work on performance.now on another PR to make it simpler to review. This one includes the Docs already updated and the added tests.

This heavily builds on [Sinon's Fake Timers](#) for the base edge cases

**To-Do**

- Refactor the code to make `setTime` actually call the `tick` method and pass the time
- Make the initial time be 0 or accept an instance of `Date`, or a specific number to make its implementation be the same as [fake-timers](#)
- Mock the Date object completely
- Allow `enable` to accept multiple overloads

**Next iterations**

- Mock `performance.now`
- Mock `process.hrtime`

New **MockTimers API**



# MOCKING DATES

```
1 import assert from 'node:assert';
2 import { test } from 'node:test';
3
4 test('mocks the Date object', (context) => {
5   context.mock.timers.enable({ apis: ['Date'] });
6   assert.strictEqual(Date.now(), 0); // Date is mocked to 1970-01-01
7
8   // Advance in time will also advance the date
9   context.mock.timers.tick(9999);
10  assert.strictEqual(Date.now(), 9999); // Date is mocked to 1970-01-01 + 9999ms
11  context.mock.timers.reset() // reset back to original state
12});
13
```



# MOCKING DATES

```
1 import assert from 'node:assert';
2 import { test } from 'node:test';
3
4 test('mocks the Date object', (context) => {
5   context.mock.timers.enable({ apis: ['Date'] });
6   assert.strictEqual(Date.now(), 0); // Date is mocked to 1970-01-01
7
8   // Advance in time will also advance the date
9   context.mock.timers.tick(9999);
10  assert.strictEqual(Date.now(), 9999); // Date is mocked to 1970-01-01 + 9999ms
11  context.mock.timers.reset() // reset back to original state
12 });
13
```



# MOCKING DATES

```
1 import assert from 'node:assert';
2 import { test } from 'node:test';
3
4 test('mocks the Date object', (context) => {
5   context.mock.timers.enable({ apis: ['Date'] });
6   assert.strictEqual(Date.now(), 0); // Date is mocked to 1970-01-01
7
8   // Advance in time will also advance the date
9   context.mock.timers.tick(9999);
10  assert.strictEqual(Date.now(), 9999); // Date is mocked to 1970-01-01 + 9999ms
11  context.mock.timers.reset() // reset back to original state
12 });
13
```



# MOCKING DATES

```
1 import assert from 'node:assert';
2 import { test } from 'node:test';
3
4 test('mocks the Date object', (context) => {
5   context.mock.timers.enable({ apis: ['Date'] });
6   assert.strictEqual(Date.now(), 0); // Date is mocked to 1970-01-01
7
8   // Advance in time will also advance the date
9   context.mock.timers.tick(9999);
10  assert.strictEqual(Date.now(), 9999); // Date is mocked to 1970-01-01 + 9999ms
11  context.mock.timers.reset() // reset back to original state
12 });
13
```



# MOCKING DATES

```
1 import assert from 'node:assert';
2 import { test } from 'node:test';
3
4 test('mocks the Date object', (context) => {
5   context.mock.timers.enable({ apis: ['Date'] });
6   assert.strictEqual(Date.now(), 0); // Date is mocked to 1970-01-01
7
8   // Advance in time will also advance the date
9   context.mock.timers.tick(9999);
10  assert.strictEqual(Date.now(), 9999); // Date is mocked to 1970-01-01 + 9999ms
11  context.mock.timers.reset() // reset back to original state
12 });
13
```



# Code Coverage\_



# Code coverage

The test runner supports code coverage under an `--experimental-test-coverage` flag

```
1 node --test --experimental-test-coverage test.js  
2
```



# Code coverage

The test runner supports code coverage under an `--experimental-test-coverage` flag

```
1 node --test --experimental-test-coverage test.js  
2
```

As you can expect, it's experimental, so use with caution



# Code coverage

```
1  > node --test --experimental-test-coverage test.js
2  ▶ sum
3    ✓ should add two numbers (0.193833ms)
4  ▶ sum (1.418792ms)
5
6  i tests 1
7  i suites 1
8  i pass 1
9  i fail 0
10 i cancelled 0
11 i skipped `^` 
12 i todo 0
13 i duration_ms 63.954292
14 i start of coverage report
15 i _____
16 i file      | line % | branch % | funcs % | uncovered lines
17 i _____
18 i index.js | 66.67 | 66.67 | 100.00 | 3-4
19 i test.js  | 100.00 | 100.00 | 100.00 |
20 i _____
21 i all fil... | 86.67 | 83.33 | 100.00 |
22 i _____
23 i end of coverage report
24
```



# Code coverage

```
1  > node --test --experimental-test-coverage test.js
2  ▶ sum
3    ✓ should add two numbers (0.193833ms)
4  ▶ sum (1.418792ms)
5
6  i tests 1
7  i suites 1
8  i pass 1
9  i fail 0
10 i cancelled 0
11 i skipped `^`
12 i todo 0
13 i duration_ms 63.954292
14 i start of coverage report
15 i _____
16 i file      | line % | branch % | funcs % | uncovered lines
17 i _____
18 i index.js  | 66.67 | 66.67 | 100.00 | 3-4
19 i test.js   | 100.00 | 100.00 | 100.00 |
20 i _____
21 i all fil... | 86.67 | 83.33 | 100.00 |
22 i _____
23 i end of coverage report
24
```



# Code coverage

```
1  > node --test --experimental-test-coverage test.js
2  ▶ sum
3    ✓ should add two numbers (0.193833ms)
4  ▶ sum (1.418792ms)
5
6  i tests 1
7  i suites 1
8  i pass 1
9  i fail 0
10 i cancelled 0
11 i skipped `^`
12 i todo 0
13 i duration_ms 63.954292
14 i start of coverage report
15 i _____
16 i file      |  line % |  branch % |  funcs % |  uncovered lines
17 i _____
18 i index.js  |  66.67 |  66.67 |  100.00 |  3-4
19 i test.js   | 100.00 | 100.00 | 100.00 |
20 i _____
21 i all fil... |  86.67 |  83.33 |  100.00 |
22 i _____
23 i end of coverage report
24
```



# Code coverage

```
1  > node --test --experimental-test-coverage test.js
2  ▶ sum
3    ✓ should add two numbers (0.193833ms)
4  ▶ sum (1.418792ms)
5
6  i tests 1
7  i suites 1
8  i pass 1
9  i fail 0
10 i cancelled 0
11 i skipped `^`
12 i todo 0
13 i duration_ms 63.954292
14 i start of coverage report
15 i _____
16 i file      | line % | branch % | funcs % | uncovered lines
17 i _____
18 i index.js | 66.67 | 66.67 | 100.00 | 3-4
19 i test.js  | 100.00 | 100.00 | 100.00 |
20 i _____
21 i all fil... | 86.67 | 83.33 | 100.00 |
22 i _____
23 i end of coverage report
24
```



# Code coverage

Those uncovered lines are the throw statement in the `sum` function

```
1  export function sum(a, b) {  
2    if (typeof a !== 'number' || typeof b !== 'number') {  
3      throw new Error('Invalid input')  
4    }  
5    return a + b  
6  }  
7
```



# Coverage reporters

You can change the coverage reporters with the flag `--test-reporter`

```
1 node --test --experimental-test-coverage \
2   --test-reporter=lcov \
3   --test-reporter-destination=lcov.info \
4   test.js
5
```



# Coverage reporters

You can change the coverage reporters with the flag `--test-reporter`

```
1 node --test --experimental-test-coverage \
2   --test-reporter=lcov \
3   --test-reporter-destination=lcov.info \
4   test.js
5
```

This gets you an `lcov.info` file that you can import into any coverage tool

The test runner supports:

- `spec`
- `dot`
- `tap`
- `lcov`
- `junit`



# TypeScript ❤

# TypeScript SUPPORT

Node already supports what's called "importers".



# TypeScript SUPPORT

Node already supports what's called "importers".

- Importers (previously "loaders") are functions that execute before a module is loaded



# TypeScript SUPPORT

Node already supports what's called "importers".

- Importers (previously "loaders") are functions that execute before a module is loaded
- They can be used to transform the module before it's imported

# TypeScript SUPPORT

Node already supports what's called "importers".

- Importers (previously "loaders") are functions that execute before a module is loaded
- They can be used to transform the module before it's imported
- This means you can transpile TypeScript on the fly

# TypeScript SUPPORT

Node already supports what's called "importers".

- Importers (previously "loaders") are functions that execute before a module is loaded
- They can be used to transform the module before it's imported
- This means you can transpile TypeScript on the fly
- Common importers are `ts-node`, `tsx`, and `esbuild`



# TypeScript SUPPORT

Let's take this example:

```
1 // index.ts
2 export function sum(a: number, b: number) {
3     if (typeof a !== 'number' || typeof b !== 'number') {
4         throw new Error('Invalid input')
5     }
6     return a + b
7 }
8
9 console.log(sum(1, 2))
10
```



# TypeScript SUPPORT

Let's take this example:

```
1 // index.ts
2 export function sum(a: number, b: number) {
3     if (typeof a !== 'number' || typeof b !== 'number') {
4         throw new Error('Invalid input')
5     }
6     return a + b
7 }
8
9 console.log(sum(1, 2))
10
```

Install `tsx` with `npm install tsx` and run the file with:

```
1 node --import=tsx index.ts
2
```



# TypeScript SUPPORT

You will get this

```
1  ➜ node --import=tsx index.ts  
2  3  
3
```



# TypeScript SUPPORT

You will get this

```
1  ➜ node --import=tsx index.ts  
2  3  
3
```

WHAT IF I TOLD YOU...



# TypeScript SUPPORT

You will get this

```
1  ➜ node --import=tsx index.ts  
2  3  
3
```

WHAT IF I TOLD YOU...

you can use this with the test runner?



# TypeScript SUPPORT

Let's just take the previous example and change the file extension to `test.ts`

Then run it with the `--import` flag



# TypeScript SUPPORT

Let's just take the previous example and change the file extension to `test.ts`

Then run it with the `--import` flag

```
1  > node --import=tsx --test test.ts
2  3 # our console
3  ► sum
4  ✓ should add two numbers (0.128ms)
5  ► sum (0.72875ms)
6
7  i tests 1
8  i suites 1
9  i pass 1
10 i fail 0
11 i cancelled 0
12 i skipped 0
13 i todo 0
14 i duration_ms 157.107959
15
```



# TypeScript SUPPORT

Let's just take the previous example and change the file extension to `test.ts`

Then run it with the `--import` flag

```
1  > node --import=tsx --test test.ts
2  3 # our console
3  ► sum
4  ✓ should add two numbers (0.128ms)
5  ► sum (0.72875ms)
6
7  i tests 1
8  i suites 1
9  i pass 1
10 i fail 0
11 i cancelled 0
12 i skipped 0
13 i todo 0
14 i duration_ms 157.107959
15
```

Now look at that `jest.config.js` file you've been maintaining as a boilerplate for 5 years and tell me if this isn't just a bliss



# OTHER SUPPORTED FEATURES

# OTHER SUPPORTED FEATURES

- Aborting tests via abort signals

# OTHER SUPPORTED FEATURES

- Aborting tests via abort signals
- Skipping tests with `\t/it.skip`

# OTHER SUPPORTED FEATURES

- Aborting tests via abort signals
- Skipping tests with `\t/it.skip`
- Todo tests with `\t/it.todo`

# OTHER SUPPORTED FEATURES

- Aborting tests via abort signals
- Skipping tests with ``t/it.skip``
- Todo tests with ``t/it.todo``
- "only" flag to run only a specific test with ``t/it.only``

# OTHER SUPPORTED FEATURES

- Aborting tests via abort signals
- Skipping tests with ``t/it.skip``
- Todo tests with ``t/it.todo``
- "only" flag to run only a specific test with ``t/it.only``
- Lifecycle hooks like ``before``, ``after``, ``beforeEach``, ``afterEach``

# OTHER SUPPORTED FEATURES

- Aborting tests via abort signals
- Skipping tests with `t/it.skip`
- Todo tests with `t/it.todo`
- "only" flag to run only a specific test with `t/it.only`
- Lifecycle hooks like `before`, `after`, `beforeEach`, `afterEach`
- Test filtering with `--test-name-pattern` via RegExp

# OTHER SUPPORTED FEATURES

- Aborting tests via `abort` signals
- Skipping tests with `t/it.skip`
- Todo tests with `t/it.todo`
- "only" flag to run only a specific test with `t/it.only`
- Lifecycle hooks like `before`, `after`, `beforeEach`, `afterEach`
- Test filtering with `--test-name-pattern` via RegExp
- Watch mode (experimental)



# The FUTURE OF Node.js TESTING

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

## A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig  
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

### Improve Filtering



# The FUTURE OF Node.js TESTING

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon

## A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig

2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

### Improve Filtering



# The FUTURE OF Node.js TESTING

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering

## A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig  
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

### Improve Filtering



# The FUTURE OF Node.js TESTING

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering
- (maybe) More reporters

## A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig  
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

### Improve Filtering



# The FUTURE OF Node.js TESTING

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering
- (maybe) More reporters
- Snapshot testing

## A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig  
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

### Improve Filtering



# The FUTURE OF Node.js TESTING

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering
- (maybe) More reporters
- Snapshot testing
- Source map support

## A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig  
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

### Improve Filtering



# The FUTURE OF Node.js TESTING

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering
- (maybe) More reporters
- Snapshot testing
- Source map support
- MOAR mocks! (like `performance`)

## A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig  
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

### Improve Filtering



# The FUTURE OF Node.js TESTING

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering
- (maybe) More reporters
- Snapshot testing
- Source map support
- MOAR mocks! (like `performance`)
- The **definitive** downfall of Jest (just kidding)

## A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig  
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

### Improve Filtering



# Get involved

The test runner is still in active development, and you can get involved!



# Get involved

The test runner is still in active development, and you can get involved!

- Documentation is always a good place to start



# Get involved

The test runner is still in active development, and you can get involved!

- Documentation is always a good place to start
- You can also contribute to the code



# GET INVOLVED

The test runner is still in active development, and you can get involved!

- Documentation is always a good place to start
- You can also contribute to the code
  - Maybe making Colin happy with his wishlist



# GET INVOLVED

The test runner is still in active development, and you can get involved!

- Documentation is always a good place to start
- You can also contribute to the code
  - Maybe making Colin happy with his wishlist
- Or just use it and give feedback!



# GET INVOLVED

The test runner is still in active development, and you can get involved!

- Documentation is always a good place to start
- You can also contribute to the code
  - Maybe making Colin happy with his wishlist
- Or just use it and give feedback!
- Start shifting your projects to it, and help the community grow!



# THANK YOU!

<https://{{twitter,instagram/github,youtube,linkedin}}.lsantos.dev>



# See THIS TALK ON MY WEBSITE

<https://lsantos.dev/talks/nodejs-test-runner>

