readme.md for @git.zone/tstest

□□ **A powerful, modern test runner for TypeScript** — beautiful output, multi-runtime support, and a batteries-included test framework that makes testing actually enjoyable.

Availability and Links

- npmjs.org (npm package)
- code.foss.global (source)

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Why tstest?

Most TypeScript test runners feel like an afterthought — clunky configuration, ugly output, and poor TypeScript support. **tstest** was built from the ground up for TypeScript developers who want:

- □□ **Zero config** — Point it at your test directory and go
- □□ **Multi-runtime** — Run the same tests on Node.js, Deno, Bun, and Chromium
- □□ **Beautiful output** — Color-coded results with emojis, progress bars, and visual diffs
- ⚡ **Built-in everything** — Assertions, snapshots, fixtures, retries, timeouts, parallel execution
- □□ **Server-side tooling** — Free ports, HTTPS certs, ephemeral databases, S3 storage — all out of the box

Installation

```
pnpm install --save-dev @git.zone/tstest
```

Module Exports

tstest ships as four modules, each optimized for a different use case:

Export Path	Environment	Purpose
`@git.zone/tstest`	CLI	Test runner — discovers and executes test files
`@git.zone/tstest/tapbundle`	Browser + Node	Core test framework — `tap`, `expect`, lifecycle hooks
`@git.zone/tstest/tapbundle_serverside`	Node.js only	Server-side utilities — ports, certs, databases, shell
`@git.zone/tstest/tapbundle_protocol`	Isomorphic	TAP Protocol V2 — structured metadata, events, diffs

Quick Start

1. Write a test

```
// test/test.math.ts
import { tap, expect } from '@git.zone/tstest/tapbundle';

tap.test('should add numbers', async () => {
  expect(2 + 2).toEqual(4);
});

tap.test('should handle async operations', async (tools) => {
  await tools.delayFor(100);
  const result = await fetchData();
  expect(result).toBeTruthy();
});
```

```
export default tap.start();
```

2. Run it

```
# Run all tests
tstest test/

# Run a specific file
tstest test/test.math.ts

# Use glob patterns
tstest "test/**/*.ts"

# Verbose mode (shows console output)
tstest test/ --verbose

# Watch mode
tstest test/ --watch
```

3. See beautiful output

```
Test Discovery
   Mode: directory
   Pattern: test
   Found: 4 test file(s)

▶  test/test.math.ts (1/4)
   Runtime: Node.js
    should add numbers (2ms)
    should handle async operations (105ms)
   Summary: 2/2 PASSED in 1.2s

Test Summary
┌─────────────────────────────────┐
│ Total Files:                4 │
│ Total Tests:                8 │
```

```
| Passed:                     8 |
| Failed:                     0 |
| Duration:                 2.4s |
└──────────────────────────────────┘


ALL TESTS PASSED! □□
```

Multi-Runtime Architecture

tstest supports running your tests across **four JavaScript runtimes**, letting you verify cross-platform compatibility with zero extra effort.

Test File Naming Convention

Name your test files with runtime specifiers to control where they run:

Pattern	Runtimes	Example
`*.ts`	Node.js (default)	`test.api.ts`
`*.node.ts`	Node.js only	`test.server.node.ts`
`*.chromium.ts`	Chromium browser	`test.dom.chromium.ts`
`*.deno.ts`	Deno	`test.http.deno.ts`
`*.bun.ts`	Bun	`test.fast.bun.ts`
`*.all.ts`	All runtimes	`test.universal.all.ts`
`*.node+chromium.ts`	Node.js + Chromium	`test.isomorphic.node+chromium.ts`
`*.node+deno.ts`	Node.js + Deno	`test.cross.node+deno.ts`
`*.chromium.nonci.ts`	Chromium, skip in CI	`test.visual.chromium.nonci.ts`

Runtime Execution Order

When multiple runtimes are specified, tests execute in this order: **Node.js → Bun → Deno → Chromium**

Migration from Legacy Naming

```
# Dry run — see what would change
tstest migrate --dry-run

# Apply migrations (uses git mv to preserve history)
tstest migrate --write
```

Legacy Pattern	Modern Equivalent
`*.browser.ts`	`*.chromium.ts`
`*.both.ts`	`*.node+chromium.ts`

CLI Options

Option	Description
`--quiet`, `-q`	Minimal output — perfect for CI
`--verbose`, `-v`	Show all console output from tests
`--no-color`	Disable colored output
`--json`	Output results as JSON (CI/CD pipelines)
`--logfile`	Save detailed logs with error/diff tracking
`--tags <tags>`	Run only tests with specific tags
`--timeout <seconds>`	Timeout test files after N seconds
`--startFrom <n>`	Start from test file number N
`--stopAt <n>`	Stop at test file number N
`--watch`, `-w`	Re-run tests on file changes
`--watch-ignore <patterns>`	Ignore patterns in watch mode
`--only`	Run only tests marked with `.only`

Writing Tests with tapbundle

Basic Syntax

```
import { tap, expect } from '@git.zone/tstest/tapbundle';
```

```
tap.test('basic test', async () => {
  expect(2 + 2).toEqual(4);
});

tap.test('with tools', async (tools) => {
  await tools.delayFor(100);
  tools.timeout(5000);
  expect(true).toBeTrue();
});

export default tap.start();
```

Test Modifiers

```
// Skip
tap.skip.test('not ready yet', async () => { /* skipped */ });

// Only (exclusive)
tap.only.test('focus on this', async () => { /* only this runs */ });

// Todo
tap.todo.test('implement later', async () => { /* marked as todo */ });

// Fluent chaining
tap.timeout(5000)
   .retry(3)
   .tags('api', 'integration')
   .test('complex test', async (tools) => { /* configured */ });
```

Test Organization with describe()

```
tap.describe('User Management', () => {
  tap.beforeEach(async () => {
    // setup before each test
  });

  tap.afterEach(async () => {
```

```
    // cleanup after each test
  });

  tap.test('should create user', async () => { /* ... */ });
  tap.test('should delete user', async () => { /* ... */ });

  tap.describe('Permissions', () => {
    tap.test('should set admin role', async () => { /* ... */ });
  });
});
```

Pre-Tasks and Post-Tasks

```
tap.preTask('setup database', async () => {
  await initializeDatabase();
});

tap.test('uses the database', async () => { /* ... */ });

tap.postTask('cleanup database', async () => {
  await cleanupDatabase();
});
```

Test Tools

Every test function receives a `tools` parameter packed with utilities:

```
tap.test('tools demo', async (tools) => {
  // □□ Delays
  await tools.delayFor(1000);
  await tools.delayForRandom(100, 500);

  // □□ Skip
  tools.skipIf(process.env.CI === 'true', 'Skipping in CI');
  tools.skip('reason');

  // □□Retry & timeout
  tools.retry(3);
```

```
  tools.timeout(10000);

  // □□Context sharing between tests
  tools.context.set('userId', 12345);
  const userId = tools.context.get('userId');

  // □□Deferred promises
  const deferred = tools.defer();
  setTimeout(() => deferred.resolve('done'), 100);
  await deferred.promise;

  // □□Error capture
  const error = await tools.returnError(async () => {
    throw new Error('Expected error');
  });
  expect(error).toBeInstanceOf(Error);

  // □ Allow failure (test won't fail the suite)
  tools.allowFailure();
});
```

Snapshot Testing

```
tap.test('snapshot test', async (tools) => {
  const output = generateComplexOutput();
  await tools.matchSnapshot(output);
  await tools.matchSnapshot(output.header, 'header');
});

// Update snapshots: UPDATE_SNAPSHOTS=true tstest test/
```

Test Fixtures

```
tap.defineFixture('testUser', async (data) => ({
  id: Date.now(),
  name: data?.name || 'Test User',
  email: data?.email || 'test@example.com',
```

```
}));

tap.test('fixture test', async (tools) => {
  const user = await tools.fixture('testUser', { name: 'John' });
  expect(user.name).toEqual('John');

  // Factory pattern for multiple instances
  const users = await tools.factory('testUser').createMany(5);
  expect(users).toHaveLength(5);
});
```

Parallel Execution

```
// Within a file
tap.parallel().test('parallel test 1', async () => { /* ... */ });
tap.parallel().test('parallel test 2', async () => { /* ... */ });

// Across files — same suffix = parallel group
// test.api.para__1.ts  ←— run together
// test.db.para__1.ts   ←— run together
// test.auth.para__2.ts ←— runs after para__1 completes
```

Assertions (expect)

tapbundle uses @push.rocks/smartexpect for assertions with automatic diff generation on failures:

```
// Equality
expect(value).toEqual(5);
expect(obj).toDeepEqual({ a: 1, b: 2 });

// Types
expect('hello').toBeTypeofString();
expect(42).toBeTypeofNumber();
expect([]).toBeArray();

// Comparisons
expect(5).toBeGreaterThan(3);
```

```
expect(0.1 + 0.2).toBeCloseTo(0.3, 10);

// Truthiness
expect(true).toBeTrue();
expect(null).toBeNull();
expect(undefined).toBeUndefined();

// Strings
expect('hello world').toStartWith('hello');
expect('hello world').toEndWith('world');
expect('hello world').toInclude('lo wo');
expect('hello world').toMatch(/^hello/);

// Arrays
expect([1, 2, 3]).toContain(2);
expect([1, 2, 3]).toContainAll([1, 3]);
expect([1, 2, 3]).toHaveLength(3);

// Objects
expect(obj).toHaveProperty('name');
expect(obj).toMatchObject({ name: 'John' });

// Functions & Promises
expect(() => { throw new Error(); }).toThrow();
await expect(Promise.resolve('val')).resolves.toEqual('val');
await expect(Promise.reject(new Error())).rejects.toThrow();

// Custom
expect(7).customAssertion(v => v % 2 === 1, 'Value is not odd');
```

Server-Side Tools (tapbundle_serverside)

For Node.js-only tests, import server-side utilities:

```
import { tapNodeTools } from '@git.zone/tstest/tapbundle_serverside';
import { tap, expect } from '@git.zone/tstest/tapbundle';
```

□□ Network Utilities

Find free local ports for test servers — no more port conflicts:

```
tap.test('should start server on free port', async () => {
  // Single free port (random in range 3000–60000)
  const port = await tapNodeTools.findFreePort();

  // Custom range
  const port2 = await tapNodeTools.findFreePort({ startPort: 8000, endPort: 9000 });

  // With exclusions
  const port3 = await tapNodeTools.findFreePort({ exclude: [8080, 8443] });
});

tap.test('should allocate multiple ports', async () => {
  // Multiple distinct ports
  const [httpPort, wsPort, adminPort] = await tapNodeTools.findFreePorts(3);

  // Consecutive port range (e.g., 4000, 4001, 4002)
  const portRange = await tapNodeTools.findFreePortRange(3, {
    startPort: 20000,
    endPort: 30000,
  });
});
```

□□ HTTPS Certificates

Generate self-signed certs for testing secure connections:

```
tap.test('should serve over HTTPS', async () => {
  const { key, cert } = await tapNodeTools.createHttpsCert('localhost');
  const server = https.createServer({ key, cert }, handler);
  server.listen(port);
});
```

□□ Shell Commands

```
const result = await tapNodeTools.runCommand('ls -la');
console.log(result.exitCode); // 0
```

□□ Environment Variables

```
const apiKey = await tapNodeTools.getEnvVarOnDemand('GITHUB_API_KEY');
// Prompts if not set, stores in .nogit/.env for future use
```

□□ Ephemeral MongoDB

```
const mongo = await tapNodeTools.createSmartmongo();
// ... run database tests ...
await mongo.stop();
```

□□ Local S3 Storage

```
const s3 = await tapNodeTools.createSmarts3();
// ... run object storage tests ...
await s3.stop();
```

Test File Directives

Control runtime behavior directly from your test files using special comment directives at the top of the file. Directives must appear before any `import` statements.

Deno Permissions

By default, Deno tests run with `--allow-read`, `--allow-env`, `--allow-net`, `--allow-write`, `--allow-sys`, and `--allow-import`. Add directives to request additional permissions:

```
// tstest:deno:allowAll
import { tap, expect } from '@git.zone/tstest/tapbundle';
```

```
tap.test('test with full Deno permissions', async () => {
  // Runs with --allow-all (e.g., for FFI, subprocess spawning, etc.)
});

export default tap.start();
```

Available Directives

Directive	Effect
`// tstest:deno:allowAll`	Grants all Deno permissions (`--allow-all`)
`// tstest:deno:allowRun`	Adds `--allow-run` for subprocess spawning
`// tstest:deno:allowFfi`	Adds `--allow-ffi` for native library calls
`// tstest:deno:allowHrtime`	Adds `--allow-hrtime` for high-res timers
`// tstest:deno:flag:--unstable-ffi`	Passes any arbitrary Deno flag
`// tstest:node:flag:--max-old-space-size=4096`	Passes flags to Node.js
`// tstest:bun:flag:--smol`	Passes flags to Bun

Multiple Directives

Combine as many directives as needed:

```
// tstest:deno:allowRun
// tstest:deno:allowFfi
// tstest:deno:flag:--unstable-ffi
import { tap, expect } from '@git.zone/tstest/tapbundle';

tap.test('test with Rust FFI', async () => {
  // Has --allow-run, --allow-ffi, and --unstable-ffi in addition to defaults
});

export default tap.start();
```

Shared Directives via 00init.ts

Directives in a `00init.ts` file apply to all test files in that directory. Test file directives are merged with (and extend) init file directives.

```
// test/00init.ts
// tstest:deno:allowRun
```

```
// test/test.mytest.deno.ts
// tstest:deno:allowFfi
// Both --allow-run (from 00init.ts) and --allow-ffi are active
import { tap, expect } from '@git.zone/tstest/tapbundle';
```

Advanced Features

Watch Mode

```
tstest test/ --watch
tstest test/ --watch --watch-ignore "dist/**,coverage/**"
```

- □□ Shows which files triggered the re-run
- □ 300ms debouncing to batch rapid changes
- □□ Clears console between runs

Visual Diffs

When assertions fail, you get beautiful diffs:

```
□ should return correct user data

   Object Diff:
   {
     name: "John",
   - age: 30,
   + age: 31,
     email: "john@example.com"
   }
```

Enhanced Logging

```
tstest test/ --logfile
```

Folder	Contents
`.nogit/testlogs/`	Current run logs
`.nogit/testlogs/previous/`	Previous run logs
`.nogit/testlogs/00err/`	Failed test logs
`.nogit/testlogs/00diff/`	Changed output diffs

JSON Output (CI/CD)

```
tstest test/ --json > test-results.json
```

```
{"event":"discovery","count":4,"pattern":"test","executionMode":"directory"}
{"event":"testResult","testName":"prepare test","passed":true,"duration":1}
{"event":"summary","summary":{"totalFiles":4,"totalTests":4,"totalPassed":4,"totalFailed":0}}
```

Tag Filtering

```
tap.tags('unit', 'api').test('api unit test', async () => { /* ... */ });
```

```
tstest test/ --tags unit,api
```

Test File Range

```
tstest test/ --startFrom 5 --stopAt 10  # Run files 5-10 only
```

Browser Testing with webhelpers

```
import { tap, webhelpers } from '@git.zone/tstest/tapbundle';

tap.test('DOM test', async () => {
```

```
  const element = await webhelpers.fixture(webhelpers.html`
    <div class="container">
      <h1>Hello</h1>
    </div>
  `);
  expect(element.querySelector('h1').textContent).toEqual('Hello');
});
```

TapWrap (Global Lifecycle)

```
import { TapWrap } from '@git.zone/tstest/tapbundle';

const tapWrap = new TapWrap({
  before: async () => { await globalSetup(); },
  after: async () => { await globalCleanup(); },
});
```

tapbundle Protocol V2

tstest includes an enhanced TAP protocol that extends TAP 13 with structured metadata while staying backwards compatible. Protocol markers (`⟦TSTEST:...⟧`) are invisible to standard TAP parsers.

```
import { ProtocolEmitter, ProtocolParser } from '@git.zone/tstest/tapbundle_protocol';

// Emit
const emitter = new ProtocolEmitter();
console.log(emitter.emitProtocolHeader());  // ⟦TSTEST:PROTOCOL:2.0.0⟧
console.log(emitter.emitTest({
  ok: true, testNumber: 1, description: 'test',
  metadata: { time: 42, tags: ['unit'] }
}).join('\n'));

// Parse
const parser = new ProtocolParser();
const messages = parser.parseLine('ok 1 - test ⟦TSTEST:time:42⟧');
```

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the license file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #5
Created 2026-03-28 10:49:33 UTC by foss.global Team
Updated 2026-03-28 12:15:31 UTC by foss.global Team