Jest JavaScript test runner

START HERE

`npx jest`
Run all tests

`npx jest --coverage`
Coverage report

`npx jest <path>`
Run matching files

`npx jest -u`
Update snapshots

`npx jest -t "<name>"`
Run matching tests

`npx jest --ci`
CI mode

`npx jest --watch`
Rerun on change

`npx jest --bail`
Stop on failure

Find tests testMatch → `transform` → **Run in workers** per file → `expect` → **Report** results, coverage

CONCEPTS

Through npm, put -- before flags: npm test -- -t "login". Flags override the config file.

Each test file runs in its own sandbox, so mocks and module state do not leak between files.

RUN TESTS

`npx jest`
Run every test

`npx jest <pattern>`
Run test files matching a regex

`npx jest -t "<name>"`
Run tests whose name matches
Matches the describe and test names joined together; the regex is case-sensitive.

`npx jest --watch`
Rerun tests related to changes
Needs git or hg; use --watchAll to rerun everything.

`npx jest --watchAll`
Rerun all tests on any change

`npx jest --onlyChanged`
Only tests for uncommitted changes

`npx jest --changedSince=<ref>`
Tests affected since a branch or commit

`npx jest --findRelatedTests <files>`
Tests that cover these source files

`npx jest --listTests`
List test files and exit

`npx jest --bail`
Stop after the first failing suite

`npx jest --config <path>`
Use a specific config file

OUTPUT AND SPEED

`npx jest --coverage`
Collect and print coverage

`npx jest --verbose`
List every test with its result

`npx jest --silent`
Hide console output from tests

`npx jest --ci`
Never write new snapshots

`npx jest --runInBand`
Run serially in one process
Also -i. Best for debugging and small CI machines.

`npx jest --maxWorkers=<n>`
Limit parallel workers (n or 50%)

OUTPUT AND SPEED (cont.)

`npx jest --shard=<i>/<n>`
Run one slice of the suite

`npx jest --testTimeout=<ms>`
Per-test timeout, default 5000

`npx jest --detectOpenHandles`
Find what keeps Jest from exiting
Implies --runInBand and is slow.

`npx jest --env=<env>`
Test environment, e.g. jsdom

`npx jest --clearCache`
Delete the Jest cache

SNAPSHOTS

`npx jest -u`
Re-record failing snapshots

`npx jest -u -t "<name>"`
Update snapshots of matching tests only

`expect(<value>).toMatchSnapshot()`
Compare against a stored snapshot

`expect(<value>).toMatchInlineSnapshot()`
Store the snapshot in the test file

Review snapshot diffs in code review: -u accepts whatever the code produces now.

WRITING TESTS

`test("<name>", () ⇒ { ... })`
Define a test (alias: it)

`describe("<name>", () ⇒ { ... })`
Group tests

`test.only("<name>", fn)`
Run only this test in the file

`test.skip("<name>", fn)`
Skip this test

`test.each([[a, b], ...])("<name>", fn)`
Run one test per table row

`beforeEach(fn) / afterEach(fn)`
Run before or after every test

`beforeAll(fn) / afterAll(fn)`
Run once per file

`expect(<value>).toBe(<x>)`
Strict equality (Object.is)

`expect(<value>).toEqual(<x>)`
Deep equality

WRITING TESTS (cont.)

`expect(<value>).toStrictEqual(<x>)`
Deep equality incl. undefined and types

`expect(<value>).toBeTruthy()`
Truthy check

`expect(<value>).toContain(<x>)`
Array item or substring

`expect(<value>).toMatchObject({...})`
Subset of properties matches

`expect(<value>).toHaveLength(<n>)`
Length check

`expect(<fn>).toThrow(<msg>)`
Function throws

`await expect(<promise>).resolves.toBe(<x>)`
Assert on a resolved promise

`await expect(<promise>).rejects.toThrow()`
Assert on a rejected promise

`expect(<value>).not.toBe(<x>)`
Negate any matcher

`expect.assertions(<n>)`
Require n assertions to run

Async tests must return or await the promise, or Jest finishes before the assertions run.

MOCKS

`jest.fn()`
Create a mock function

`jest.fn(<impl>)`
Mock with an implementation

`jest.spyOn(<obj>, "<method>")`
Spy on a method, keep the original

`jest.mock("<module>")`
Auto-mock a module (hoisted)

`jest.mock("<module>", () ⇒ ({...}))`
Mock a module with a factory

`mockFn.mockReturnValue(<x>)`
Always return x

`mockFn.mockResolvedValue(<x>)`
Return a promise resolving to x

`mockFn.mockRejectedValue(<e>)`
Return a promise rejecting with e

`mockFn.mockImplementation(<fn>)`
Set the implementation

MOCKS (cont.)

`mockFn.mockReturnValueOnce(<x>)`
Return x for the next call only

`toHaveBeenCalledWith(<args>)`
Assert on call arguments

`expect(mockFn).toHaveBeenCalledTimes(<n>)`
Assert on call count

`mockFn.mock.calls`
Arguments of every call

`mockFn.mockClear()`
Forget calls, keep implementation

`mockFn.mockRestore()`
Reset and restore the original (spyOn)

`jest.restoreAllMocks()`
Restore every spyOn mock

`jest.requireActual("<module>")`
Load the real module inside a mock

`jest.useFakeTimers()`
Fake setTimeout and friends

`jest.advanceTimersByTime(<ms>)`
Move fake time forward

CONFIG

`npm init jest@latest`
Create a config interactively

`testEnvironment: "jsdom"`
Browser-like DOM (needs jest-environment-jsdom)

`testMatch: ["**/*.test.js"]`
Which files count as tests

RECIPES

Debug one failing test

```
npx jest path/to/file.test.js -t "<name>" --runInBand
# attach a debugger:
node --inspect-brk node_modules/.bin/jest --runInBand
```

Pre-commit: test what you changed

```
npx jest --bail --findRelatedTests src/a.js src/b.js
```

CI run with coverage

```
npx jest --ci --coverage --maxWorkers=50%
# split over 3 machines:
npx jest --ci --shard=1/3
```

Mock a module and assert

```
jest.mock("./api");
const api = require("./api");
api.getUser.mockResolvedValue({ id: 1 });
await run();
expect(api.getUser).toHaveBeenCalledWith(1);
```