Version: 29.7

Getting Started

Install Jest using your favorite package manager:

npm | Yarn | pnpm

```
npm install --save-dev jest
```

Let's get started by writing a test for a hypothetical function that adds two numbers. First, create a `sum.js` file:

```
function sum(a, b) {
  return a + b;
}
module.exports = sum;
```

Then, create a file named `sum.test.js`. This will contain our actual test:

```
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Add the following section to your `package.json`:

```
{
  "scripts": {
    "test": "jest"
  }
}
```

Finally, run `yarn test` or `npm test` and Jest will print this message:

```
PASS  ./sum.test.js
✓ adds 1 + 2 to equal 3 (5ms)
```

You just successfully wrote your first test using Jest!

This test used `expect` and `toBe` to test that two values were exactly identical. To learn about the other things that Jest can test, see Using Matchers.

Running from command line

You can run Jest directly from the CLI (if it's globally available in your `PATH`, e.g. by `yarn global add jest` or `npm install jest --global`) with a variety of useful options.

Here's how to run Jest on files matching `my-test`, using `config.json` as a configuration file and display a native OS notification after the run:

```
jest my-test --notify --config=config.json
```

If you'd like to learn more about running `jest` through the command line, take a look at the Jest CLI Options page.

Additional Configuration

Generate a basic configuration file

Based on your project, Jest will ask you a few questions and will create a basic configuration file with a short description for each option:

npm Yarn pnpm

```
npm init jest@latest
```

Using Babel

To use Babel, install required dependencies:

npm Yarn pnpm

```
npm install --save-dev babel-jest @babel/core @babel/preset-env
```

Configure Babel to target your current version of Node by creating a `babel.config.js` file in the root of your project:

babel.config.js

```
module.exports = {
  presets: [['@babel/preset-env', {targets: {node: 'current'}}]],
};
```

The ideal configuration for Babel will depend on your project. See Babel's docs for more details.

Making your Babel config jest-aware

Using webpack

Jest can be used in projects that use webpack to manage assets, styles, and compilation. webpack does offer some unique challenges over other tools. Refer to the webpack guide to get started.

Using Vite

Jest can be used in projects that use vite to serve source code over native ESM to provide some frontend tooling, vite is an opinionated tool and does offer some out-of-the box workflows. Jest is not fully supported by vite due to how the plugin system from vite works, but there are some working examples for first-class jest integration using `vite-jest`, since this is not fully supported, you might as well read the limitation of the `vite-jest`. Refer to the vite guide to get started.

Using Parcel

Jest can be used in projects that use parcel-bundler to manage assets, styles, and compilation similar to webpack. Parcel requires zero configuration. Refer to the official docs to get started.

Using TypeScript

Via `babel`

Jest supports TypeScript, via Babel. First, make sure you followed the instructions on using Babel above. Next, install the `@babel/preset-typescript`:

npm Yarn pnpm

```
npm install --save-dev @babel/preset-typescript
```

Then add `@babel/preset-typescript` to the list of presets in your `babel.config.js`.

babel.config.js

```
module.exports = {
  presets: [
    ['@babel/preset-env', {targets: {node: 'current'}}],
    '@babel/preset-typescript',
  ],
};
```

However, there are some caveats to using TypeScript with Babel. Because TypeScript support in Babel is purely transpilation, Jest will not type-check your tests as they are run. If you want that, you can use ts-jest instead, or just run the TypeScript compiler tsc separately (or as part of your build process).

Via `ts-jest`

ts-jest is a TypeScript preprocessor with source map support for Jest that lets you use Jest to test projects written in TypeScript.

npm Yarn pnpm

```
npm install --save-dev ts-jest
```

In order for Jest to transpile TypeScript with `ts-jest`, you may also need to create a configuration file.

Type definitions

There are two ways to have Jest global APIs typed for test files written in TypeScript.

You can use type definitions which ships with Jest and will update each time you update Jest. Install the `@jest/globals` package:

npm Yarn pnpm

```
npm install --save-dev @jest/globals
```

And import the APIs from it:

sum.test.ts

```
import {describe, expect, test} from '@jest/globals';
import {sum} from './sum';

describe('sum module', () => {
  test('adds 1 + 2 to equal 3', () => {
    expect(sum(1, 2)).toBe(3);
  });
});
```

TIP

See the additional usage documentation of `describe.each`/`test.each` and `mock functions`.

Or you may choose to install the `@types/jest` package. It provides types for Jest globals without a need to import them.

npm Yarn pnpm

```
npm install --save-dev @types/jest
```

INFO

> `@types/jest` is a third party library maintained at DefinitelyTyped, hence the latest Jest features or versions may not be covered yet. Try to match versions of Jest and `@types/jest` as closely as possible. For example, if you are using Jest `27.4.0` then installing `27.4.x` of `@types/jest` is ideal.

Edit this page

*Last updated on **Sep 12, 2023** by **Simen Bekkhus***