@wordpress/jest-console

- Installation
  - Setup
  - Usage
  - .toHaveErrored()
  - .toHaveErroredWith(arg1, arg2, ...)
  - .toHaveInformed()
  - .toHaveInformedWith(arg1, arg2, ...)
  - .toHaveLogged()
  - .toHaveLoggedWith(arg1, arg2, ...)
  - .toHaveWarned()
  - .toHaveWarnedWith(arg1, arg2, ...)
- Contributing to this package

Custom Jest matchers for the Console
object to test JavaScript code in WordPress.

This package converts `console.error`, `console.info`, `console.log` and `console.warn` functions into mocks and tracks their calls.
It also enforces usage of one of the related matchers whenever tested code calls one of the mentioned `console` methods.
It means that you need to assert with `.toHaveErrored()` or `.toHaveErroredWith( arg1, arg2, ... )` when `console.error`
gets executed, and use the corresponding methods when `console.info`, `console.log` or `console.warn` are called.
Your test will fail otherwise! This is a conscious design decision which helps to detect deprecation warnings when
upgrading dependent libraries or smaller errors when refactoring code.

Installation

Install the module:

```
npm install @wordpress/jest-console --save-dev
```

Note: This package requires Node.js version with long-term support status (check Active LTS or Maintenance LTS releases). It is not compatible with older versions.

Setup

The simplest setup is to use Jest's `setupFilesAfterEnv` config option:

```
"jest": {
  "setupFilesAfterEnv": [
    "@wordpress/jest-console"
  ]
},
```

Usage

`.toHaveErrored()`

Use `.toHaveErrored` to ensure that `console.error` function was called.

For example, let's say you have a `drinkAll( flavor )` function that makes you drink all available beverages.
You might want to check if function calls `console.error` for `'octopus'` instead, because `'octopus'` flavor is really
weird and why would anything be octopus-flavored? You can do that with this test suite:

```
describe( 'drinkAll', () => {
    test( 'drinks something lemon-flavored', () => {
        drinkAll( 'lemon' );
        expect( console ).not.toHaveErrored();
    } );

    test( 'errors when something is octopus-flavored', () => {
        drinkAll( 'octopus' );
        expect( console ).toHaveErrored();
    } );
} );
```

`.toHaveErroredWith( arg1, arg2, ... )`

Use `.toHaveErroredWith` to ensure that `console.error` function was called with specific arguments.

For example, let's say you have a `drinkAll( flavor )` function again makes you drink all available beverages.
You might want to check if function calls `console.error` with a specific message for `'octopus'` instead, because
`'octopus'` flavor is really weird and why would anything be octopus-flavored? To make sure this works, you could write:

```
describe( 'drinkAll', () => {
    test( 'errors with message when something is octopus-flavored', () => {
        drinkAll( 'octopus' );
        expect( console ).toHaveErroredWith(
            'Should I really drink something that is octopus-flavored?'
        );
    } );
} );
```

`.toHaveInformed()`

Use `.toHaveInformed` to ensure that `console.info` function was called.

Almost identical usage as `.toHaveErrored()`.

`.toHaveInformedWith( arg1, arg2, ... )`

Use `.toHaveInformedWith` to ensure that `console.info` function was called with specific arguments.

Almost identical usage as `.toHaveErroredWith()`.

`.toHaveLogged()`

Use `.toHaveLogged` to ensure that `console.log` function was called.

Almost identical usage as `.toHaveErrored()`.

`.toHaveLoggedWith( arg1, arg2, ... )`

Use `.toHaveLoggedWith` to ensure that `console.log` function was called with specific arguments.

Almost identical usage as `.toHaveErroredWith()`.

`.toHaveWarned()`

Use `.toHaveWarned` to ensure that `console.warn` function was called.

Almost identical usage as `.toHaveErrored()`.

`.toHaveWarnedWith( arg1, arg2, ... )`

Use `.toHaveWarneddWith` to ensure that `console.warn` function was called with specific arguments.

Almost identical usage as `.toHaveErroredWith()`.

Contributing to this package

This is an individual package that's part of the Gutenberg project. The project is organized as a monorepo. It's made up of multiple self-contained software packages, each with a specific purpose.

The packages in this monorepo are published to npm and used by WordPress as well as other software projects.

To find out more about contributing to this package or Gutenberg as a whole, please read the project's main contributor guide.

□□□ #2
□ □□□ □□□ 2026-10-31 18:19:28 UTC
□ □□□ □□□ 2026-10-31 18:19:29 UTC