Reporters

Introduction

Playwright Test comes with a few built-in reporters for different needs and ability to provide custom reporters. The easiest way to try out built-in reporters is to pass `--reporter` command line option.

```
npx playwright test --reporter=line
```

For more control, you can specify reporters programmatically in the configuration file.

playwright.config.ts
```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: 'line',
});
```

Multiple reporters

You can use multiple reporters at the same time. For example you can use `'list'` for nice terminal output and `'json'` to get a comprehensive json file with the test results.

playwright.config.ts
```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: [
    ['list'],
    ['json', {  outputFile: 'test-results.json' }]
  ],
});
```

Reporters on CI

You can use different reporters locally and on CI. For example, using concise `'dot'` reporter avoids too much output. This is the default on CI.

playwright.config.ts
```
import { defineConfig } from '@playwright/test';
```

```ts
export default defineConfig({
  // Concise 'dot' for CI, default 'list' when running locally
  reporter: process.env.CI ? 'dot' : 'list',
});
```

Built-in reporters

All built-in reporters show detailed information about failures, and mostly differ in verbosity for successful runs.

List reporter

List reporter is default (except on CI where the `dot` reporter is default). It prints a line for each test being run.

```sh
npx playwright test --reporter=list
```

playwright.config.ts

```ts
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: 'list',
});
```

Here is an example output in the middle of a test run. Failures will be listed at the end.

```
npx playwright test --reporter=list
Running 124 tests using 6 workers

 1  ✓ should access error in env (438ms)
 2  ✓ handle long test names (515ms)
 3  x 1) render expected (691ms)
 4  ✓ should timeout (932ms)
 5    should repeat each:
 6  ✓ should respect enclosing .gitignore (569ms)
 7    should teardown env after timeout:
 8    should respect excluded tests:
 9  ✓ should handle env beforeEach error (638ms)
10    should respect enclosing .gitignore:
```

You can opt into the step rendering via passing the following config option:

playwright.config.ts

```ts
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: [['list', { printSteps: true }]],
});
```

List report supports the following configuration options and environment variables:

Environment Variable Name	Reporter Config Option	Description	Default
PLAYWRIGHT_LIST_PRINT_STEPS	printSteps	Whether to print each step on its own line.	false
PLAYWRIGHT_FORCE_TTY		Whether to produce output suitable for a live terminal. If a number is specified, it will also be used as the terminal width.	true when terminal is in TTY mode, false otherwise.
FORCE_COLOR		Whether to produce colored output.	true when terminal is in TTY mode, false otherwise.

Line reporter

Line reporter is more concise than the list reporter. It uses a single line to report last finished test, and prints failures when they occur. Line reporter is useful for large test suites where it shows the progress but does not spam the output by listing all the tests.

```
npx playwright test --reporter=line
```

playwright.config.ts

```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: 'line',
});
```

Here is an example output in the middle of a test run. Failures are reported inline.

```
npx playwright test --reporter=line
Running 124 tests using 6 workers
  1) dot-reporter.spec.ts:20:1 › render expected
=======================================================

    Error: expect(received).toBe(expected) // Object.is equality

    Expected: 1
    Received: 0

[23/124] gitignore.spec.ts - should respect nested .gitignore
```

Line report supports the following configuration options and environment variables:

Environment Variable Name	Reporter Config Option	Description	Default
PLAYWRIGHT_FORCE_TTY		Whether to produce output suitable for a live terminal. If a	true when terminal is in TTY

Environment Variable Name	Reporter Config Option	Description	Default
		number is specified, it will also be used as the terminal width.	mode, `false` otherwise.
`FORCE_COLOR`		Whether to produce colored output.	`true` when terminal is in TTY mode, `false` otherwise.

Dot reporter

Dot reporter is very concise - it only produces a single character per successful test run. It is the default on CI and useful where you don't want a lot of output.

```
npx playwright test --reporter=dot
```

playwright.config.ts

```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: 'dot',
});
```

Here is an example output in the middle of a test run. Failures will be listed at the end.

```
npx playwright test --reporter=dot
Running 124 tests using 6 workers
......F.........................................
```

One character is displayed for each test that has run, indicating its status:

Character	Description
`·`	Passed
`F`	Failed
`×`	Failed or timed out - and will be retried
`±`	Passed on retry (flaky)
`T`	Timed out
`°`	Skipped

Dot report supports the following configuration options and environment variables:

Environment Variable Name	Reporter Config Option	Description	Default
`PLAYWRIGHT_FORCE_TTY`		Whether to produce output suitable for a live terminal. If a number is specified, it will also be used as the terminal width.	`true` when terminal is in TTY mode, `false` otherwise.
`FORCE_COLOR`		Whether to produce colored output.	`true` when terminal is in TTY

Environment Variable Name	Reporter Config Option	Description	Default
			mode, `false` otherwise.

HTML reporter

HTML reporter produces a self-contained folder that contains report for the test run that can be served as a web page.

```
npx playwright test --reporter=html
```

By default, HTML report is opened automatically if some of the tests failed. You can control this behavior via the `open` property in the Playwright config or the `PLAYWRIGHT_HTML_OPEN` environmental variable. The possible values for that property are `always`, `never` and `on-failure` (default).

You can also configure `host` and `port` that are used to serve the HTML report.

playwright.config.ts
```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: [['html', { open: 'never' }]],
});
```

By default, report is written into the `playwright-report` folder in the current working directory. One can override that location using the `PLAYWRIGHT_HTML_OUTPUT_DIR` environment variable or a reporter configuration.

In configuration file, pass options directly:

playwright.config.ts
```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: [['html', { outputFolder: 'my-report' }]],
});
```

If you are uploading attachments from data folder to other location, you can use `attachmentsBaseURL` option to let html report where to look for them.

playwright.config.ts
```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: [['html', { attachmentsBaseURL: 'https://external-storage.com/'
}]],
});
```

A quick way of opening the last test run report is:

```
npx playwright show-report
```

Or if there is a custom folder name:

```
npx playwright show-report my-report
```

HTML report supports the following configuration options and environment variables:

Environment Variable Name	Reporter Config Option	Description	Default
`PLAYWRIGHT_HTML_OUTPUT_DIR`	`outputFolder`	Directory to save the report to.	`playwright-report`
`PLAYWRIGHT_HTML_OPEN`	`open`	When to open the html report in the browser, one of `'always'`, `'never'` or `'on-failure'`	`'on-failure'`
`PLAYWRIGHT_HTML_HOST`	`host`	When report opens in the browser, it will be served bound to this hostname.	`localhost`
`PLAYWRIGHT_HTML_PORT`	`port`	When report opens in the browser, it will be served on this port.	`9323` or any available port when `9323` is not available.
`PLAYWRIGHT_HTML_ATTACHMENTS_BASE_URL`	`attachmentsBaseURL`	A separate location where attachments from the `data` subdirectory are uploaded. Only needed	`data/`

Environment Variable Name	Reporter Config Option	Description	Default
		when you upload report and `data` separately to different locations.	

Blob reporter

Blob reports contain all the details about the test run and can be used later to produce any other report. Their primary function is to facilitate the merging of reports from sharded tests.

```
npx playwright test --reporter=blob
```

By default, the report is written into the `blob-report` directory in the package.json directory or current working directory (if no package.json is found). The report file name looks like `report-<hash>.zip` or `report-<hash>-<shard_number>.zip` when sharding is used. The hash is an optional value computed from `--grep`, `--grepInverted`, `--project` and file filters passed as command line arguments. The hash guarantees that running Playwright with different command line options will produce different but stable between runs report names. The output file name can be overridden in the configuration file or pass as `'PLAYWRIGHT_BLOB_OUTPUT_FILE'` environment variable.

playwright.config.ts

```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: [['blob', { outputFile: `./blob-report/report-
${os.platform()}.zip` }]],
});
```

Blob report supports following configuration options and environment variables:

Environment Variable Name	Reporter Config Option	Description	Default
`PLAYWRIGHT_BLOB_OUTPUT_DIR`	`outputDir`	Directory to save the output. Existing content is deleted before writing the new report.	`blob-report`
`PLAYWRIGHT_BLOB_OUTPUT_NAME`	`fileName`	Report file name.	`report-<project>-<hash>-<shard_number>.zip`
`PLAYWRIGHT_BLOB_OUTPUT_FILE`	`outputFile`	Full path to the output file. If defined, `outputDir` and	`undefined`

Environment Variable Name	Reporter Config Option	Description	Default
		`fileName` will be ignored.	

JSON reporter

JSON reporter produces an object with all information about the test run.

Most likely you want to write the JSON to a file. When running with `--reporter=json`, use `PLAYWRIGHT_JSON_OUTPUT_NAME` environment variable:

- Bash
- PowerShell
- Batch

```
PLAYWRIGHT_JSON_OUTPUT_NAME=results.json npx playwright test --
reporter=json
$env:PLAYWRIGHT_JSON_OUTPUT_NAME="results.json"
npx playwright test --reporter=json
set PLAYWRIGHT_JSON_OUTPUT_NAME=results.json
npx playwright test --reporter=json
```

In configuration file, pass options directly:

playwright.config.ts

```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: [['json', { outputFile: 'results.json' }]],
});
```

JSON report supports following configuration options and environment variables:

Environment Variable Name	Reporter Config Option	Description	Default
`PLAYWRIGHT_JSON_OUTPUT_DIR`		Directory to save the output file. Ignored if output file is specified.	`cwd` or config directory.
`PLAYWRIGHT_JSON_OUTPUT_NAME`	`outputFile`	Base file name for the output, relative to the output dir.	JSON report is printed to the stdout.
`PLAYWRIGHT_JSON_OUTPUT_FILE`	`outputFile`	Full path to the output file. If defined, `PLAYWRIGHT_JSON_OUTPUT_DIR` and	JSON report is printed to the stdout.

Environment Variable Name	Reporter Config Option	Description	Default
		`PLAYWRIGHT_JSON_OUTPUT_NAME` will be ignored.	

JUnit reporter

JUnit reporter produces a JUnit-style xml report.

Most likely you want to write the report to an xml file. When running with `--reporter=junit`, use `PLAYWRIGHT_JUNIT_OUTPUT_NAME` environment variable:

- Bash
- PowerShell
- Batch

```
PLAYWRIGHT_JUNIT_OUTPUT_NAME=results.xml npx playwright test --
reporter=junit
$env:PLAYWRIGHT_JUNIT_OUTPUT_NAME="results.xml"
npx playwright test --reporter=junit
set PLAYWRIGHT_JUNIT_OUTPUT_NAME=results.xml
npx playwright test --reporter=junit
```

In configuration file, pass options directly:

playwright.config.ts

```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: [['junit', { outputFile: 'results.xml' }]],
});
```

JUnit report supports following configuration options and environment variables:

Environment Variable Name	Reporter Config Option	Descripti
`PLAYWRIGHT_JUNIT_OUTPUT_DIR`		Directory to save the Ignored if output file specified.
`PLAYWRIGHT_JUNIT_OUTPUT_NAME`	`outputFile`	Base file name for th relative to the output
`PLAYWRIGHT_JUNIT_OUTPUT_FILE`	`outputFile`	Full path to the outpu defined, `PLAYWRIGHT_JUNIT_` and `PLAYWRIGHT_JUNIT_` will be ignored.

Environment Variable Name	Reporter Config Option	Descripti
`PLAYWRIGHT_JUNIT_STRIP_ANSI`	`stripANSIControlSequences`	Whether to remove A sequences from the t writing it in the repo
`PLAYWRIGHT_JUNIT_INCLUDE_PROJECT_IN_TEST_NAME`	`includeProjectInTestName`	Whether to include P project name in every a name prefix.
`PLAYWRIGHT_JUNIT_SUITE_ID`		Value of the `id` attrib root `<testsuites/>`
`PLAYWRIGHT_JUNIT_SUITE_NAME`		Value of the `name` att root `<testsuites/>`

GitHub Actions annotations

You can use the built in `github` reporter to get automatic failure annotations when running in GitHub actions.

Note that all other reporters work on GitHub Actions as well, but do not provide annotations. Also, it is not recommended to use this annotation type if running your tests with a matrix strategy as the stack trace failures will multiply and obscure the GitHub file view.

playwright.config.ts

```ts
import { defineConfig } from '@playwright/test';

export default defineConfig({
  // 'github' for GitHub Actions CI to generate annotations, plus a concise 'dot'
  // default 'list' when running locally
  reporter: process.env.CI ? 'github' : 'list',
});
```

Custom reporters

You can create a custom reporter by implementing a class with some of the reporter methods. Learn more about the Reporter API.

my-awesome-reporter.ts

```ts
import type {
  FullConfig, FullResult, Reporter, Suite, TestCase, TestResult
} from '@playwright/test/reporter';

class MyReporter implements Reporter {
  onBegin(config: FullConfig, suite: Suite) {
    console.log(`Starting the run with ${suite.allTests().length} tests`);
  }

  onTestBegin(test: TestCase, result: TestResult) {
    console.log(`Starting test ${test.title}`);
```

```
  }

  onTestEnd(test: TestCase, result: TestResult) {
    console.log(`Finished test ${test.title}: ${result.status}`);
  }

  onEnd(result: FullResult) {
    console.log(`Finished the run: ${result.status}`);
  }
}

export default MyReporter;
```

Now use this reporter with testConfig.reporter.

playwright.config.ts

```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: './my-awesome-reporter.ts',
});
```

Or just pass the reporter file path as `--reporter` command line option:

```
npx playwright test --reporter="./myreporter/my-awesome-reporter.ts"
```

Third party reporter showcase

- Allure
- Argos Visual Testing
- Currents
- GitHub Actions Reporter
- GitHub Pull Request Comment
- Mail Reporter
- Microsoft Teams Reporter
- Monocart
- ReportPortal
- Serenity/JS
- Testmo
- Testomat.io
- Tesults

Previous
Projects

Next
Retries

- Built-in reporters
  - List reporter
  - Line reporter
  - Dot reporter
  - HTML reporter
  - Blob reporter
  - JSON reporter
  - JUnit reporter
  - GitHub Actions annotations
- Custom reporters
- Third party reporter showcase

Learn

- Getting started
- Playwright Training
- Learn Videos
- Feature Videos

Community

- Stack Overflow
- Discord
- Twitter
- LinkedIn

More

- GitHub
- YouTube
- Blog
- Ambassadors