Playwright Cheat Sheet PDF

A Playwright cheat sheet lists the CLI commands to install, run, debug and report, the main locators, common actions such as click and fill, frequent assertions with expect, and key configuration options. Use it for fast lookup while writing tests, and check the official documentation for the current API.

Key takeaways

- Most daily work uses npx playwright test, codegen, show-report and show-trace.
- Prefer getByRole, getByLabel and getByText for locators.
- Use await expect(locator) assertions, which retry automatically.
- Config options such as retries, workers and trace control the run.
- Treat the sheet as a memory aid and confirm details in the official docs.

What is Playwright quick reference?

A Playwright cheat sheet is a compact reference of commands, locators, actions, assertions and settings that testers use when writing and running Playwright tests.

Which commands do I use most?

```
npm init playwright@latest          # create a project
npx playwright test                 # run all tests
npx playwright test login.spec.ts   # run one file
npx playwright test -g "checkout"   # run by title
npx playwright test --project=firefox
npx playwright test --headed        # show browser
npx playwright test --debug         # open inspector
npx playwright test --ui            # UI mode
npx playwright test --trace on
npx playwright test --update-snapshots
npx playwright codegen <url>        # record actions
npx playwright show-report
npx playwright show-trace trace.zip
```

Which locators should I memorise?

```
page.getByRole('button', { name: 'Save' })
page.getByLabel('Email')
page.getByPlaceholder('Search')
page.getByText('Welcome')
page.getByTestId('cart-count')
page.locator('css=.card').filter({ hasText: 'Pro' })
page.getByRole('row').nth(2)
```

Which actions are most common?

Action	Code
Open a page	`await page.goto('/path')`
Click	`await locator.click()`
Type text	`await locator.fill('text')`
Select option	`await locator.selectOption('value')`
Check box	`await locator.check()`
Press key	`await page.keyboard.press('Enter')`
Upload file	`await locator.setInputFiles('file.png')`
Hover	`await locator.hover()`
Screenshot	`await page.screenshot({ path: 'a.png' })`

Which assertions do I use daily?

```
await expect(locator).toBeVisible();
await expect(locator).toBeHidden();
await expect(locator).toHaveText('Done');
await expect(locator).toContainText('Done');
await expect(locator).toHaveValue('abc');
await expect(locator).toHaveCount(3);
await expect(locator).toBeEnabled();
await expect(page).toHaveURL(/dashboard/);
await expect(page).toHaveTitle('Home');
```

All of these retry until the timeout.

What does a test file skeleton look like?

```
import { test, expect } from '@playwright/test';

test.describe('cart', () => {
  test.beforeEach(async ({ page }) => { await page.goto('/'); });
  test('adds an item', async ({ page }) => {
    await page.getByRole('button', { name: 'Add to cart' }).click();
    await expect(page.getByTestId('cart-count')).toHaveText('1');
  });
});
```

Which config options are worth remembering?

`testDir`, `timeout`, `retries`, `workers`, `fullyParallel`, `reporter`, `use.baseURL`, `use.trace`, `use.screenshot`, `projects`, `webServer` and `forbidOnly`. Know what each does and when you would change it.

What helpers and hooks exist?

`test.beforeAll`, `test.beforeEach`, `test.afterEach`, `test.afterAll`, `test.skip`, `test.fixme`, `test.slow`, `test.step`, `test.use` and tags in titles for grep. Use `test.step` to make reports read like a user story.

What quick habits prevent bugs?

Use await on every Playwright call, prefer role locators, never sleep, keep tests independent, read the trace on failure and keep selectors in page objects when the suite grows.

How do I handle dialogs, frames and new tabs?

Dialogs such as alerts are handled with `page.on('dialog', d => d.accept())` registered before the action that opens them. Iframes are reached with `page.frameLocator('#payment').getByLabel('Card number')`. A new tab is captured with `const [popup] = await Promise.all([page.waitForEvent('popup'), link.click()])`, after which you use `popup` like a normal page. Keep these three patterns on your sheet because they appear in many interviews and real projects.

How do I deal with downloads and uploads?

For a download, wait for the event while clicking: `const [download] = await Promise.all([page.waitForEvent('download'), button.click()])`, then call `download.saveAs('file.pdf')` and check the name. For an upload, call `setInputFiles` on the file input with a path or a list of paths. Both patterns avoid fixed waits and keep the test short and reliable.

How do I authenticate once and reuse it?

Log in in a setup project, save state with `context.storageState({ path: 'auth.json' })`, and set `storageState` in the config for projects that need it. Tests start already signed in. For several roles, store one file per role. Never commit these files because they contain session cookies; add them to .gitignore.

What should I do when a command is not on the sheet?

Search the official API reference by class name such as Page, Locator or APIRequestContext. Read the example near the method, try it in a tiny test and then add a line to your own sheet. Over time your personal sheet becomes the most useful one because it reflects the problems you actually meet.

Which mistakes does a cheat sheet help avoid?

Forgetting await, using sleeps, choosing brittle selectors, registering route handlers too late, running a test with only mode left in the code and forgetting to install browsers on a new machine. Keep a short list of these on the sheet and read it before each coding session, especially when you are new to the tool.

How do I use the sheet for revision before an interview?

Cover the right-hand column of each table and try to write the command or code from memory, then check yourself. Repeat for the weak rows until you can produce them without hints. Pair this with a small project so that every line has been run at least once. Interviewers often ask for a short code sample on the spot, so practised recall pays off.

Frequently asked questions

What commands should I know first in Playwright?

Learn init, test, test with headed or debug flags, codegen, show-report and show-trace, since they cover most daily work.

Which locators does a cheat sheet include?

It lists getByRole, getByLabel, getByPlaceholder, getByText, getByTestId and filtering or chaining for narrowing a match.

How do I run a single test?

Pass the file name to npx playwright test, or use the grep option with part of the test title to select it.

Is a cheat sheet enough to learn Playwright?

No. It is a memory aid, so learn the concepts from the documentation and practise by writing tests on a real application.

Will the API change?

Playwright updates regularly, so check the official documentation for the current API when a command behaves differently.

Sources and further reading

- Playwright documentation
- Playwright API reference
- Playwright test CLI

About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550