Cypress Commands & Methods – Usage Guide

1. Test Structure & Hooks

`describe()` – Test Suite

Groups related tests.

```
describe('Login flow', () => {
  // tests here
})
```

Use when: You want to organize tests by feature/module.

`it()` – Individual Test

Defines a single test case.

```
it('logs in with valid credentials', () => {
  // steps & assertions
})
```

Use when: You express a single behavior you want to verify.

Hooks: `before`, `beforeEach`, `afterEach`, `after`

Run setup/teardown logic.

```
before(() => {
  // runs once before all tests
})

beforeEach(() => {
```

```
  // runs before every test
})

afterEach(() => {
  // runs after every test
})

after(() => {
  // runs once after all tests
})
```

Typical uses:

- `before`: seed DB, log in via API.
- `beforeEach`: visit page, reset state, set cookies.
- `after/afterEach`: cleanup, log out, reset data.

---

2. Navigation & Page-Level Commands

`cy.visit()` – Load a Page (Cypress Documentation)

Navigates to a URL (absolute or relative to `baseUrl`).

```
cy.visit('/')                  // uses baseUrl
cy.visit('/login')             // /login on baseUrl
cy.visit('http://localhost:3000/dashboard') // full URL
```

Options example:

```
cy.visit('/users', {
  qs: { page: '1', role: 'admin' }, // query params
  failOnStatusCode: false
})
```

Use when: Starting any UI test or navigating to specific routes.

`cy.reload()` – Reload Current Page

```
cy.reload()          // normal reload
cy.reload(true)      // reload without using cache
```

Use when: You want to re-trigger page load or verify persistence.

`cy.url()` / `cy.title()`

```
cy.url().should('include', '/dashboard')
cy.title().should('eq', 'My App – Dashboard')
```

Use when: Verifying redirects, routing, or page titles.

---

3. Querying the DOM (Getting Elements)

`cy.get()` – Find by Selector (Cypress Documentation)

Gets one or more elements using a CSS selector or alias.

```
cy.get('input[name="email"]')
cy.get('.btn-primary')
cy.get('@userRow')   // using alias
```

Use when: You know the CSS selector / data attribute.

> 🔑 Best practice: Use `data-cy` or `data-test` attributes for stable selectors.

`cy.contains()` – Find by Text (Cypress Documentation)

Gets element containing specific text.

```
cy.contains('Submit')          // any element containing text
cy.contains('button', 'Save')  // button with text Save
```

```
cy.contains('li', /user-\d+/)  // using RegExp
```

Use when: Locating buttons/links by visible label (more user-like).

`cy.find()` – Find Inside an Existing Subject

Used after `cy.get()` to narrow search.

```
cy.get('form').find('input[type="password"]')
```

Use when: You want to search *within* a specific container.

`cy.within()` – Scoped Queries

All `cy.get()` calls inside `within` are scoped to that element.

```
cy.get('form#login').within(() => {
  cy.get('input[name="email"]').type('[email protected]')
  cy.get('input[name="password"]').type('secret')
  cy.contains('button', 'Login').click()
})
```

Use when: Page has repeated patterns (cards, forms, tables).

---

4. User Actions (Interacting with Elements)

These are usually chained off a `cy.get()` or `cy.contains()`.

`.click()`

```
cy.get('button[type="submit"]').click()
cy.contains('Delete').click()
```

Options (e.g., force click):

```
cy.get('button.hidden').click({ force: true })
```

`.type()`

Types into input/textarea.

```
cy.get('input[name="email"]').type('[email protected]')
cy.get('textarea').type('Some multi-line\ntext')
```

With options:

```
cy.get('#search').type('Cypress{enter}')
```

`.clear()`

```
cy.get('input[name="email"]').clear().type('[email protected]')
```

`.check()` / `.uncheck()` – Checkboxes & Radios

```
cy.get('input[type="checkbox"]').check()
cy.get('input[type="checkbox"]').uncheck()
cy.get('input[name="gender"]').check('male')   // value="male"
```

`.select()` – `<select>` Dropdown

```
cy.get('select#country').select('India')     // by text
cy.get('select#country').select('IN')       // by value
```

Scroll & Viewport

```
cy.get('#footer').scrollIntoView()
cy.scrollTo('bottom')
```

---

5. Assertions

`.should()` – Chained Assertions (LambdaTest)

```
cy.get('h1').should('contain', 'Dashboard')
cy.get('input').should('be.visible').and('be.enabled')
cy.get('@userCount').should('be.gt', 0)
```

Common built-in assertions:

- State: `be.visible`, `exist`, `be.disabled`, `be.checked`
- Text/value: `have.text`, `contain`, `have.value`
- Attributes/CSS: `have.attr`, `have.class`, `have.css`
- Length: `have.length`, `have.length.at.least`, etc.

`.and()` – Chain More Assertions

```
cy.get('button')
  .should('be.visible')
  .and('have.text', 'Save')
```

`expect()` – Explicit Assertions (Chai)

Used inside `.then()` callbacks.

```
cy.get('@userResponse').then((res) => {
  expect(res.status).to.eq(200)
  expect(res.body).to.have.property('id')
})
```

---

6. Timing, Waits & Retry-ability

Cypress automatically retries commands and assertions until timeout. Use manual waits sparingly.

`cy.wait()` – Time-based or Alias-based

```
cy.wait(1000)          // not recommended usually

// better – wait on network call
cy.intercept('GET', '/api/users').as('getUsers')
cy.visit('/users')
cy.wait('@getUsers')
```

Use when: You need to wait for network calls or debounced UI changes.

`cy.clock()` & `cy.tick()`

Control the browser clock (for timers, polling, etc.).

```
cy.clock()
// some code that sets setTimeout...
cy.tick(5000)  // move time forward 5s
```

---

7. Aliases, Variables & Chaining

`.as()` – Create Alias

Alias DOM elements, routes, or arbitrary values.

```
cy.get('table#users').as('usersTable')
cy.get('@usersTable').should('be.visible')

cy.intercept('GET', '/api/users').as('getUsers')
cy.wait('@getUsers')

cy.wrap({ role: 'admin' }).as('userData')
```

`cy.wrap()` – Wrap JS Value into Cypress Chain

```
const user = { name: 'Murali' }
cy.wrap(user).its('name').should('eq', 'Murali')
```

`.then()` – Work with Resolved Values

```
cy.get('input[name="email"]').then(($input) => {
  const val = $input.val()
  expect(val).to.include('@')
})
```

Use when: You need imperative logic, conditionals, or to bridge Cypress chain with plain JS.

---

8. Network & API Testing

`cy.request()` – HTTP Requests (Reflect)

Make API calls without using the UI.

```
cy.request('GET', '/api/users').then((response) => {
  expect(response.status).to.eq(200)
  expect(response.body).to.have.length(5)
})

// with body
cy.request('POST', '/api/login', {
  email: '[email protected]',
  password: 'secret'
})
```

Use when: Login via API, seeding data, pure API tests.

`cy.intercept()` – Stub / Spy Network Calls (Cypress Documentation)

Intercept and control XHR/fetch.

```
cy.intercept('GET', '/api/users').as('getUsers')
cy.visit('/users')
```

```
cy.wait('@getUsers').its('response.statusCode').should('eq', 200)
```

Stub response:

```
cy.intercept('GET', '/api/users', {
  statusCode: 200,
  body: [{ id: 1, name: 'Test User' }]
})
cy.visit('/users')
```

Use when: Mocking backend, testing error states, speeding up tests.

---

9. Fixtures & Test Data

`cy.fixture()`

Load static JSON or text from `cypress/fixtures`.

```
cy.fixture('user.json').then((user) => {
  cy.get('input[name="email"]').type(user.email)
})
```

Combine with `cy.intercept()`:

```
cy.fixture('users.json').then((users) => {
  cy.intercept('GET', '/api/users', { body: users })
})
```

---

10. Window, Document & Application State

`cy.window()` & `cy.document()` (GeeksforGeeks)

Access the underlying `window` or `document` object.

```
cy.window().then((win) => {
  expect(win.localStorage.getItem('token')).to.exist
```

```
})

cy.document().its('contentType').should('eq', 'text/html')
```

Use when: Working with storage, global variables, or low-level DOM details.

`cy.viewport()`

Change browser viewport size.

```
cy.viewport(1280, 720)
cy.viewport('iphone-6')   // preset
```

Use when: Testing responsive layouts.

---

11. System & Node Side

`cy.exec()`

Run system commands on the machine running Cypress.

```
cy.exec('node scripts/reset-db.js')
  .its('code')
  .should('eq', 0)
```

`cy.task()`

Run custom Node code defined in `setupNodeEvents`.

```
// cypress.config.js (in setupNodeEvents)
on('task', {
  log(message) {
    console.log(message)
    return null
  },
})
```

```
// test
cy.task('log', 'Running test for user creation')
```

Use when: You need backend helpers, DB access, file system, etc.

---

12. Custom Commands

`Cypress.Commands.add()` / `.overwrite()` (Cypress Documentation)

Create reusable commands.

```
// cypress/support/commands.js
Cypress.Commands.add('login', (email, password) => {
  cy.visit('/login')
  cy.get('input[name="email"]').type(email)
  cy.get('input[name="password"]').type(password)
  cy.contains('button', 'Login').click()
})

// usage
cy.login('[email protected]', 'secret')
```

Overwrite:

```
Cypress.Commands.overwrite('visit', (originalFn, url, options) => {
  // add auth header or logging
  return originalFn(url, options)
})
```

Use when: You see repeated patterns in tests; want to build a domain-specific language (DSL) for your app.

---

13. Common Patterns & Best Practices (Quick Notes)

- Prefer `cy.contains()` **+ semantics** instead of brittle CSS selectors.

- Use **aliases** (`.as()`) for:
    - network calls (`@getUsers`)
    - frequently used elements (`@submitBtn`)
    - complex data (`@user`)
- Avoid `cy.wait(5000)` unless absolutely necessary; prefer waiting on intercept aliases or assertions.
- Bring logic into **custom commands** instead of long, repetitive test bodies.
- Keep **test code readable**: “Given–When–Then” structure, clear intent in test names.

If you tell me your audience level (absolute beginners vs experienced testers), I can trim this into a **1–2 page printable PDF style** or expand with **advanced methods** (sessions, origin, shadow DOM, iframes, etc.).