# Cypress Commands & Best Practices

Updated quick-reference handout • January 25, 2026

A concise cheat sheet for modern Cypress end-to-end and component testing: the commands you use every day, the patterns that keep tests stable, and the anti-patterns that cause flake.

### At a glance

- **Prefer automatic waiting + explicit alias waits** (avoid fixed time waits).
- **Use stable selectors** (data-* attributes) and keep tests user-focused.
- **Stub/spy network calls with cy.intercept()** for speed and determinism.
- **Cache authentication with cy.session()** via a login helper/custom command.
- **Keep specs independent** and seed/reset data in a predictable way.

## Contents

# 1. Quick start (CLI) and structure

Typical setup uses npm/yarn/pnpm with Cypress installed as a dev dependency. Run in interactive mode for local development and headless mode in CI.

| | |
|---|---|
| **npx cypress open**<br>Example | Launch the Cypress App (interactive runner).<br>`npx cypress open` |
| **npx cypress run**<br>Example | Run all specs headlessly (CI-friendly).<br>`npx cypress run` |
| **--browser**<br>Example | Choose a browser (e.g., chrome, edge, electron).<br>`npx cypress run --browser chrome` |
| **--spec**<br>Example | Run a specific spec file or pattern.<br>`npx cypress run --spec "cypress/e2e/login.cy.ts"` |
| **--config key=value**<br>Example | Override config values for a run.<br>`npx cypress run --config baseUrl=https://staging.example.com` |
| **CYPRESS_* env vars**<br>Example | Set env/config via environment variables.<br>`CYPRESS_baseUrl=https://staging.example.com npx cypress run` |

## Recommended folder layout

```
cypress/
  e2e/                # end-to-end specs
  component/          # component specs (if enabled)
  fixtures/           # static test data (JSON, images)
  support/
    e2e.ts            # global hooks + custom commands
    commands.ts
  downloads/          # generated at runtime
  screenshots/        # generated at runtime
  videos/             # generated at runtime
cypress.config.ts
```

Keep helper logic in **support**, test data in **fixtures**, and avoid cross-spec coupling (each spec should be runnable alone).

# 2. Cypress execution model

Cypress commands are enqueued and run asynchronously. Commands yield subjects that flow through the chain. Assertions automatically retry until they pass or time out.

## Golden rules

- Do **not** store command results in variables (anti-pattern). Use **.then()** or aliases.
- Prefer **cy.get(...).should(...)** over manual sleeps - Cypress retries queries and assertions for you.
- Use **within()** to scope queries and avoid brittle selectors.
- Keep each test independent: set up what you need, assert, and clean up.

## Anti-pattern vs. correct pattern

```
# Anti-pattern: variable assignment (won't work as expected)
const btn = cy.get('[data-cy=submit]')
btn.click()

# Correct: keep it in the chain
cy.get('[data-cy=submit]').click()

# Correct: extract a value with then()
cy.get('[data-cy=total]')
  .invoke('text')
  .then((txt) => {
    expect(Number(txt)).to.be.greaterThan(0)
  })
```

Tip: if you need to share a yielded value later, use **.as()** and retrieve with **cy.get('@alias')** (or use **cy.wrap(value).as('alias')**).

# 3. Command cheat sheet

Most-used commands grouped by purpose. Focus on chaining and explicit assertions.

## 3.1 Navigation & page state

| | |
|---|---|
| **cy.visit()**<br>Example | Load a page. Use baseUrl for cleaner calls.<br>`cy.visit('/login')` |
| **cy.url()**<br>Example | Read the current URL (yield string).<br>`cy.url().should('include', '/dashboard')` |
| **cy.location()**<br>Example | Read parts of the URL (pathname, hash, etc.).<br>`cy.location('pathname').should('eq', '/dashboard')` |
| **cy.go()**<br>Example | Navigate back/forward in history.<br>`cy.go('back')` |
| **cy.reload()**<br>Example | Reload the page.<br>`cy.reload()` |

## 3.2 Querying & scoping

| | |
|---|---|
| **cy.get()**<br>Example | Query DOM by CSS selector (retries until found).<br>`cy.get('[data-cy=login-email]')` |
| **cy.contains()**<br>Example | Find an element containing text (useful for buttons/labels).<br>`cy.contains('button', 'Sign in').click()` |
| **cy.find()**<br>Example | Find descendants of the current subject.<br>`cy.get('[data-cy=modal]').find('input').first().type('abc')` |
| **cy.within()**<br>Example | Scope all queries to an element subtree.<br>`cy.get('[data-cy=checkout]').within(() => {`<br>`  cy.get('input[name=zip]').type('10001')`<br>`})` |
| **cy.root()**<br>Example | Get the root element of a within() scope.<br>`cy.get('[data-cy=form]').within(() => {`<br>`  cy.root().should('have.class', 'ready')`<br>`})` |

## 3.3 Actions

| | |
|---|---|
| **click()**<br>Example | Click an element. Prefer visible, actionable elements.<br>`cy.get('[data-cy=submit]').click()` |
| **type()**<br>Example | Type into an input/textarea.<br>`cy.get('[data-cy=email]').type('user@example.com')` |
| **clear()**<br>Example | Clear an input.<br>`cy.get('[data-cy=search]').clear().type('cypress')` |
| **check()/uncheck()**<br>Example | Toggle a checkbox/radio.<br>`cy.get('[data-cy=tos]').check()` |
| **select()**<br>Example | Select an option in .<br>`cy.get('[data-cy=country]').select('United States')` |
| **trigger()**<br>Example | Fire an event (drag/drop, hover, etc.).<br>`cy.get('[data-cy=card]').trigger('mouseover')` |

## 3.4 Assertions

| | |
|---|---|
| **should()**<br>Example | Implicit retry until assertion passes or times out.<br>`cy.get('[data-cy=toast]').should('contain', 'Saved')` |
| **and()**<br>Example | Chain multiple assertions.<br>`cy.get('[data-cy=price]').should('be.visible').and('match', /\$/)` |
| **invoke()**<br>Example | Call a jQuery method and yield the result.<br>`cy.get('[data-cy=total]').invoke('text').should('match', /\d+/)` |
| **its()**<br>Example | Get a property of the current subject.<br>`cy.get('@resp').its('response.statusCode').should('eq', 200)` |
| **expect()**<br>Example | Use Chai assertions inside .then().<br>`cy.get('[data-cy=count]').then(($el) => {`<br>`  expect(Number($el.text())).to.be.gte(1)`<br>`})` |

## 3.5 Utilities

**cy.wrap()**
Example

Wrap a value and keep it in the command chain.

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

**cy.then()**
Example

Bridge async Cypress chain to synchronous JS logic.

```
cy.then(() => {
  // do pure JS work here
})
```

**cy.as()**
Example

Alias a command's subject for later reuse.

```
cy.get('[data-cy=nav]').as('nav')
cy.get('@nav').should('be.visible')
```

**cy.log()**
Example

Add custom messages to the command log.

```
cy.log('Creating customer test data')
```

**cy.request()**
Example

HTTP request without UI (seed data, health checks).

```
cy.request('POST', '/api/test/seed', { plan: 'pro' })
```

**cy.task()**
Example

Run Node-side code (DB, files, external systems).

```
cy.task('db:reset')
```

**cy.fixture()**
Example

Load test data from cypress/fixtures.

```
cy.fixture('users/admin.json').then((u) => {
  cy.wrap(u).as('admin')
})
```

# 4. Network testing with cy.intercept()

Use **cy.intercept()** to spy on requests, stub responses, and make tests more deterministic. You can alias intercepts and wait on them instead of using fixed delays.

## 4.1 Spy (no stubbing) + wait

```
cy.intercept('GET', '/api/users*').as('getUsers')
cy.visit('/users')
cy.wait('@getUsers').its('response.statusCode').should('eq', 200)
```

## 4.2 Stub a response (static)

```
cy.intercept('GET', '/api/users*', { fixture: 'users/list.json' }).as('getUsers')
cy.visit('/users')
cy.wait('@getUsers')
cy.get('[data-cy=user-row]').should('have.length', 3)
```

## 4.3 Stub dynamically (route handler)

```
cy.intercept('POST', '/api/orders', (req) => {
  expect(req.body).to.have.property('items')
  req.reply({ statusCode: 201, body: { id: 'ORDER-123' } })
}).as('createOrder')

cy.get('[data-cy=checkout]').click()
cy.wait('@createOrder').its('response.body.id').should('eq', 'ORDER-123')
```

## Best practices

- Register intercepts **before** the action that triggers the request (often before cy.visit()).
- Prefer waiting on **aliases** (cy.wait('@alias')) over waiting on time (cy.wait(2000)).
- Keep stubs realistic: match real shapes, status codes, and headers where relevant.
- For deterministic UI, stub unstable 3rd-party calls and keep the app under your control.

## 4.4 Route matcher patterns

```
# Method + wildcard URL
cy.intercept('GET', '/api/users/*').as('user')

# Matcher object (method, url, query, headers)
cy.intercept({
  method: 'GET',
  url: '/api/search*',
  headers: { 'x-tenant': 'acme' },
}).as('search')
```

# 5. Sessions & authentication with cy.session()

**cy.session()** caches browser session state so you can avoid repeating slow UI logins. The most stable approach is to wrap session creation in a reusable login helper or custom command, and optionally add a validate() function to confirm the session is still valid.

## 5.1 Example: login command using cy.session()

```
// cypress/support/commands.ts
Cypress.Commands.add('login', (user) => {
  cy.session(['login', user.email], () => {
    cy.request('POST', '/api/auth/login', {
      email: user.email,
      password: user.password,
    }).then((resp) => {
      window.localStorage.setItem('token', resp.body.token)
    })
  }, {
    validate() {
      cy.request({
        method: 'GET',
        url: '/api/auth/me',
        failOnStatusCode: false,
      }).its('status').should('eq', 200)
    },
  })
})
```

## 5.2 Use it in tests

```
beforeEach(() => {
  cy.fixture('users/admin.json').then((admin) => {
    cy.login(admin)
  })
})

it('opens the dashboard', () => {
  cy.visit('/dashboard')
  cy.get('[data-cy=welcome]').should('be.visible')
})
```

## Notes

- Use a **stable session key** (array is fine) so different users get different cached sessions.
- If your app uses cookies instead of localStorage tokens, set cookies in the session callback.
- Use validate() to avoid false positives when a cached session is expired or invalid.
- Use cacheAcrossSpecs only when you fully control global test isolation and understand the tradeoffs.

# 6. Selector strategy & test design best practices

Stable selectors and focused assertions reduce flake and maintenance. Design tests around user intent, not implementation details.

## 6.1 Selector hierarchy (recommended)

- **Best:** dedicated test attributes (e.g., data-cy, data-testid).
- **Good:** accessible roles/labels (often via Testing Library helpers) when stable.
- **Avoid:** long CSS chains, nth-child, volatile classes, and text that changes frequently.

## 6.2 Example: data-cy convention

```
<!-- app code -->
<button data-cy="submit-order">Place order</button>

<!-- test -->
cy.get('[data-cy=submit-order]').click()
```

## 6.3 Test design checklist

- **Arrange:** create/seed data via API (cy.request) or tasks; avoid UI-only setup when possible.
- **Act:** take the minimum UI steps required to exercise the behavior.
- **Assert:** verify outcomes users care about (visible state, navigation, key network calls).
- **Isolate:** reset state between tests (db reset, cleanup APIs, or new test data).
- **Control dependencies:** avoid external sites/services; stub unstable third-party calls.

## 6.4 When to use cy.request()

Use cy.request() to seed state, create users/orders, or perform auth without relying on UI steps. It is faster and tends to be more reliable than driving the UI for setup.

```
cy.request('POST', '/api/test/reset')
cy.request('POST', '/api/test/seed', { plan: 'pro' })
```

# 7. Custom commands & TypeScript tips

Custom commands keep specs readable and consistent. With TypeScript, add type declarations so commands are autocompleted and type-checked.

## 7.1 Example: selector helper

```
// cypress/support/commands.ts
Cypress.Commands.add('getBySel', (id, ...args) => {
  return cy.get(`[data-cy="${id}"]`, ...args)
})

// Usage
cy.getBySel('login-email').type('user@example.com')
```

## 7.2 Add TypeScript types for custom commands

```
// cypress/support/index.d.ts (or cypress/support/e2e.ts)
declare global {
  namespace Cypress {
    interface Chainable {
      getBySel(id: string, options?: Partial<Loggable & Timeoutable & Withinable & Shadow>):
      login(user: { email: string; password: string }): Chainable<void>
    }
  }
}
export {}
```

## 7.3 Patterns to keep support code healthy

- Keep commands **small** and composable; prefer returning the Cypress chain.
- Avoid hard-coded waits and arbitrary timeouts inside commands.
- Prefer API-level login + cy.session() for speed; keep a UI login fallback if needed.
- Put complex one-off logic in helper functions (pure TS) and call them from tests.

# 8. Debugging, flake reduction, and CI checklist

## 8.1 Debugging tools

- Use **cy.pause()** to stop execution in the runner and step through commands.
- Use **.debug()** to inspect a yielded subject in DevTools.
- Use **Cypress.config()** and **Cypress.env()** to verify runtime configuration.
- Save artifacts: screenshots/videos on failure (especially in CI).

```
cy.get('[data-cy=card]').debug()
cy.pause()
```

## 8.2 Flake reduction checklist

- Replace fixed waits with **cy.wait('@alias')** + assertions.
- Assert on **stable UI state** (visibility, enabled/disabled, text) before acting.
- Disable or reduce **animations** in test environments if they cause timing issues.
- Use deterministic test data (seed/reset via API or cy.task).
- Keep each test focused; avoid long, multi-step 'mega tests'.

## 8.3 CI checklist (recommended defaults)

- Run headless: **npx cypress run**.
- Use a consistent browser version and baseUrl per environment.
- Fail fast on uncaught exceptions only when appropriate (don't hide real issues).
- Use retries for known flaky environments (but fix the root cause).
- Split specs for speed; consider parallelization if available.

# 9. Common pitfalls & fixes

## 9.1 Detached DOM / stale elements

If the page re-renders, previously grabbed elements may detach. Prefer re-querying before acting, or assert stability first.

```
# Prefer: query right before the action
cy.contains('button', 'Save').should('be.enabled').click()
```

## 9.2 Overusing before()

Heavy setup in before() can create hidden coupling. Prefer beforeEach() with fast setup (API + session caching).

## 9.3 Fighting Cypress's retry-ability

Cypress retries commands and assertions automatically. If you pull values out too early, you can break retries.

```
# Better: keep assertions inside should()
cy.get('[data-cy=status]').should('have.text', 'Ready')
```

## 9.4 Mixing concerns in E2E tests

- E2E tests should validate end-to-end flows. Keep deep UI styling and edge-case logic in component/unit tests.
- Prefer API tests for contract validation, and E2E for high-value user journeys.

## 10. References

- **Cypress Best Practices**: https://docs.cypress.io/app/core-concepts/best-practices
- **cy.intercept() command**: https://docs.cypress.io/api/commands/intercept
- **Network Requests guide**: https://docs.cypress.io/app/guides/network-requests
- **cy.session() command**: https://docs.cypress.io/api/commands/session
- **Cypress App changelog**: https://docs.cypress.io/app/references/changelog

Document date: January 25, 2026. If you're on a different Cypress major version, cross-check command options against the official docs above.