Cypress Commands Cheat Sheet

Author: Anshita Bhasin
https://www.linkedin.com/in/anshita-bhasin/

Cypress is a testing tool for web applications. It is based on JavaScript and uses a unique and modern approach to testing that makes it easy to use and powerful at the same time.

Cypress has built-in support for continuous integration and deployment, making it a popular choice among developers and testers needing to move quickly and efficiently.

Here are some examples of useful Cypress commands:

- cy.visit(): This command is used to visit a specific URL. For example, if you wanted to visit any website, you could use the following code:

Plain text

```
cy.visit('https://www.amazon.ae/')
```

In this example, we are using the cy.visit() command to visit the URL https://www.amazon.ae.This can be useful for testing the behavior of the page at that URL, or for testing the behavior of the page when it is redirected to a specific URL.

- cy.get(): This command is used to select an element on the page.Here is an example of how to use the cy.get() command:

Plain text

```
cy.get('button').click();
```

In this example, we are using the cy.get() command to find a button element on the page, and then we are using the click() command to simulate a click on that element. This allows us to test the behavior of the page when a user clicks on the button.

- .contains(): This command is used to search for a specific string of text on the page. For example, if you wanted to search for the word "testing" on the page, you could use the following code (Ref Line 1)

Plain text

```
1  cy.contains('testing')
2  cy.get('#footer').contains('Terms and conditions')
```

- .click(): This command is used to click on an element on the page. You can click on an element by providing the co-ordinates, positions and can also combine keys with the click(). Some of the most used click commands are as below:

Plain text

```
cy.get('Sign Up').click()
cy.get("[alt='iPhone']").as("iphone");
    cy.get("@iphone").click()
cy.get('li:first').click({
  shiftKey: true,
})
cy.get('Sign Up').click({force: true}})
```

- .type(): This command is used to type text into an input field on the page. For example, if you wanted to type the text "cypress" into a search field, you could use the 1st code from below. You can also pass the keys like enter. Below are some of the commonly used commands:

Plain text

```
cy.type('cypress')
cy.get('input[name=email]').type('johndoe@example.com')
cy.get("[placeholder='Search']").type("{shift}{alt}macboo
cy.get('input[type=text]').type('Test', { force: true })
cy.get('#input-firstname').type("Iphone {enter}")
cy.get('#input-firstname').type("Iphone").type('{enter}')
cy.get("#input-lastname").type("AB", { delay: 1000 });
```

- .clear(): This command is used to clear the text from an input field. For example, if you wanted to clear the text from a search field, you could use .clear(). You can also type and clear it or get the specific element if an element appears more than once on the web page.

Plain text

```
cy.get('#search-field').clear()
cy.get('#search-field').type("hello").clear()
cy.get('#button-cart').first().clear()
```

- .check(): This command is used to check a checkbox or radio button. For example, if you wanted to check a form with multiple values at the same time, it can be done by passing the values in an array format.(Refer example 4th)

Plain text

```
cy.get('[type="checkbox"]').check()
cy.get('[type="radio"]').check()
cy.get('[type="checkbox"]').check('Female')
cy.get('[type="checkbox"]').check(['US','India'])
cy.get('[type="checkbox"]').check({ force: true })
```

- .uncheck(): This command is used to uncheck a checkbox or radio button. For example, if you wanted to uncheck the multiple values at the same time in a form, it can be done by passing the values in an array format.(*Refer example 1)*

Plain text

```
cy.get('.action-check [type="checkbox"]')
      .check(['checkbox1', 'checkbox3'])
      .uncheck(['checkbox1', 'checkbox3']).should('not.be.ch

cy.get('[type="checkbox"]')
      .not('[disabled]')
      .uncheck()
```

- .select(): This command is used to select an option from a dropdown menu. For example, if you wanted to select the option "Avatar" from a dropdown with the ID "animals", you could use the first command from the below code and if you want to select multiple values at the same time, you can the use 2nd command. You can also select the hidden elements on the page by using the command 3 from the below code

Plain text

```
cy.get("select#animals").select("Avatar");
cy.get("select#second").select(["Donut", "Bonda"]);
cy.get("select#first").select("Google", { force: true });
```

- cy.wait(): This command is used to pause the execution of your test for a specified amount of time. For example, if you wanted to pause your test for 2 seconds, you could use the following code:

Plain text

```
cy.wait(2000)
```

- cy.intercept(): This command is used to intercept and mock network requests made by the page being tested. This command accepts a request method and a URL pattern as arguments, and it returns an object that can be used to configure the mock response for the intercepted request.

Here is an example of how to use the cy.intercept() command:

Plain text

```
cy.intercept('GET', 'https://example.com/api/users').as('get
// Perform some actions on the page that make a GET request
cy.get('button').click();// Configure the mock response for
cy.get('@getUsers').then(interception => {
  interception.reply(200, [{ id: 1, name: 'Tom' }, { id: 2,
});
```

In this example, we are using the cy.intercept() command to intercept and mock a GET request to the URL https://example.com/api/users. Then, we perform some actions on the page that make this request, and finally we use the cy.get() and then() commands to configure the mock response for the intercepted request. This allows us to test the behavior of the page when it receives a specific response from the server.

- cy.request(): This command is used to make a request to a specific URL. For example, if you wanted to make a GET request to the URL "https://api.coinbase.com/v2/currencies" and log the response, you can use the following code:

Plain text

```
cy.request("GET", "https://api.coinbase.com/v2/currencies").
    (response) => {
      console.log(response.status)})
```

- cy.task(): This command is used to run custom Node.js code within your test. For example, if you wanted to print a message on the console, you could use the following code:

Plain text

```
cy.task("log", "hello World");
```

You need to update setupNodeEvents in cypress.config.js for using task command

- cy.scrollTo(): This command is used to scroll to a specific element on the page. You can scroll on a page by passing the coordinates or the position like bottom,topLeft.

Here is an example of how to use the cy.scrollTo() command:

Plain text

```
cy.scrollTo(0, 500);
cy.scrollTo("topRight");
cy.scrollTo("bottom");
cy.scrollTo(0, 100);
```

- cy.getCookie(): This command is used to retrieve a cookie with a specific name from the current page. This command accepts the name of the cookie as an argument and returns an object containing the cookie's name, value, and other information.

  Below is an example of how to use the cy.getCookie() command:

```
cy.getCookie('myCookie').then(cookie => {
  console.log(cookie.name); // logs the cookie name
  console.log(cookie.value); // logs the cookie value
});
```

Keep in mind that this only works if the cookie has been set in the current page. If you are trying to access a cookie from a different domain, you will need to use a different method.

- cy.clearCookies() : This command is used to delete all the cookies on the current page. This command does not accept any arguments, and it does not return anything.

Here is an example of how to use the cy.clearCookies() command:

Plain text •••

```
cy.clearCookies();
```

This command will delete all the cookies in the current page, so be careful when using it. If you only want to delete a specific cookie, you can use the cy.clearCookie() => *cy.getCookie('token')* command instead. This command accepts the name of the cookie as an argument, and only deletes the cookie with that name.

- cy.url() : This command is used to get the current URL of the page being tested. This command does not accept any arguments, and it returns a string containing the current URL.

Below is an example of how to use the cy.url() command

Plain text •••

```
cy.url().then(url => {
  console.log(url); // logs the current URL
});
```

This command is commonly used to verify that the page being tested has the expected URL. For example, you could use this command to verify that a user has been redirected to the correct page after logging in or submitting a form.

- cy.location(): This command is used to get the current location of the page being tested. This command does not accept any arguments, and it returns an object containing information

about the current location, such as the URL, hostname, pathname, and search parameters.

Here is an example of how to use the cy.location() command:

Plain text

```
cy.location().then(location => {
  console.log(location.href); // logs the current URL
  console.log(location.hostname); // logs the current hostna
  console.log(location.pathname); // logs the current pathna
  console.log(location.search); // logs the current search p
});
```

This command is commonly used to verify that the page being tested has the expected location. For example, you could use this command to verify that a user has been redirected to the correct page after logging in or submitting a form.

- cy.reload(): This command is used to reload the current page. This command does not accept any arguments, and it does not return anything.

Here is an example of how to use the cy.reload() command:

Plain text

```
cy.reload();
```

This command will cause the current page to be reloaded, which can be useful in a number of situations. For example, you could use this command to ensure that the page being tested is in a consistent state before running a test or to test the behavior of the page when it is reloaded.

- cy.clock() : This command is used to control the clock for a test. This command can be used to set the initial time for the clock or to manually advance the clock by a specified amount of time. This can be useful when testing time-dependent features, such as timeouts or scheduled events.

Here is an example of how to use the cy.clock() command:

Plain text

```
const myFunction = () => {
  // do something
```

```
};cy.clock(1000); // set the initial time to 1 second// Call
myFunction();cy.tick(1000); // advance the clock by another
```

In this example, we are using the cy.clock() command to set the initial time for the clock to 1 second. Then, we use the cy.tick() command to advance the clock by another second. This allows us to verify that myFunction was called after 1 second, as expected.

- cy.viewport() : This command is used to change the size of the viewport for a test. This can be useful when testing the responsiveness of a web application, or when simulating different devices and screen sizes.

Here is an example of how to use the cy.viewport() command:

Plain text

```
cy.viewport(320, 480); // set the viewport size to 320x480 p
```

In this example, we are using the cy.viewport() command to set the viewport size to 320x480 pixels. To check, in detail about testing responsiveness using cypress, you can refer to this blog.

- cy.window() : This command is used to get the global window object for the page being tested. This command does not accept any arguments, and it returns the window object as a Cypress subject, which can be used to interact with the page in the same way a user would.

Here is an example of how to use the cy.window() command:

Plain text

```
cy.window().then(win => {
  // Use the window object to interact with the page
  win.scrollTo(0, 100);
  win.alert('Hello, world!');
});
```

In this example, we are using the cy.window() command to get the window object, and then we are using it to scroll the page and display an alert. This allows us to test the behavior of the page in the same way a user would.

- cy.title() : This command is used to get the title of the page being tested. This command does not accept any arguments, and it returns a string containing the page's title.

Here is an example of how to use the cy.title() command:

Plain text

```
cy.title().then(title => {
  console.log(title); // logs the page's title
});
```

This command is commonly used to verify that the page being tested has the expected title. For example, you could use this command to verify that a user has been redirected to the correct page after logging in or submitting a form.

- cy.dblclick() : This command is used to simulate a double-click on an element in the page being tested. This command accepts a selector as an argument, and it returns the element that was double-clicked as a Cypress subject.

Here is an example of how to use the cy.dblclick() command :

Plain text

```
cy.get('button').dblclick();
```

In this example, we are using the cy.get() command to find a button element on the page, and then we are using the cy.dblclick() command to simulate a double-click on that element. This can be useful for testing the behavior of the page when a user double-clicks on an element.

- cy.rightclick(): This command is used to simulate a right-click on an element in the page being tested. This command accepts a selector as an argument, and it returns the element that was right-clicked as a Cypress subject.

Here is an example of how to use the cy.rightclick() command:

Plain text

```
cy.get('button').rightclick();
```

In this example, we are using the cy.get() command to find a button element on the page, and then we are using the cy.rightclick() command to simulate a right-click on that element. This can be useful for testing the behavior of the page when a user right-clicks on an element.

- cy.children(), cy.parent(), cy.closest(): Cypress does not have a built-in “traversal” command. However, Cypress provides a number of powerful commands that can be used to traverse the page and interact with elements in the page.

For example, the cy.get() command can be used to find elements in the page based on their CSS selector. The cy.children() and cy.parent() commands can be used to find the child and parent

elements of a given element, respectively. The cy.closest() command can be used to find the closest ancestor of a given element that matches a given selector.

These commands can be combined to traverse the page and interact with elements in a variety of ways. For example, you could use the cy.get() command to find a button on the page, and then use the cy.closest() command to find the closest ancestor of that button that has a specific class. You could then use the cy.parent() command to find the parent element of that ancestor, and so on.

- cy.spy(): This command is used to create a "spy" on a specific function. A spy is a special type of test double that records information about how the function is called and what arguments it is called with. This information can be used to verify that the function is being called as expected in the test.

Here is an example of how to use the cy.spy() command:

Plain text

```
const myFunction = () => {
  // do something
};
cy.spy(myFunction);// Call myFunction in your test
myFunction();// Verify that myFunction was called
cy.get('@myFunction').should('have.been.calledOnce');
```

In this example, we are creating a spy on the myFunction function. Then, we call the function in our test, and finally, we use the cy.get() and should() commands to verify that the function was called as expected.

Conclusion:

The above-shared commands are the commonly used cypress commands. It can be helpful when you are using Cypress in your current project or if you are preparing for your interview.

Reference: https://docs.cypress.io/api/table-of-contents

Thanks for reading. Happy Learning!

~Anshita Bhasin