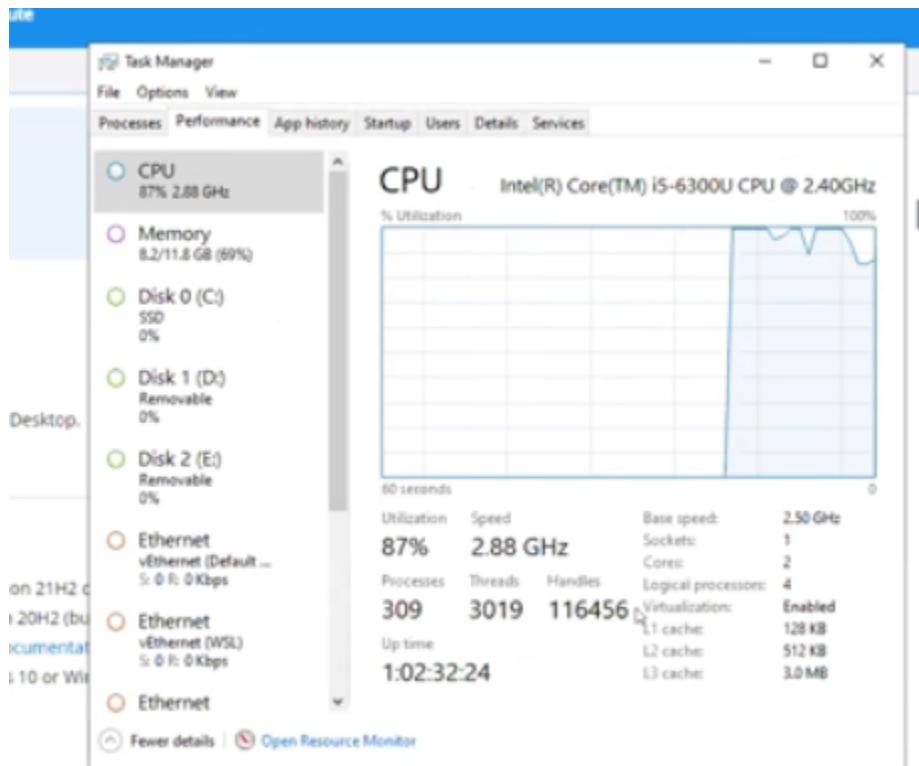


Download docker desktop.

pre -requisite -

Check virtualisation is enabled.



It means on your machine you can create multiple virtual machines.

To see all commands in docker-

Open cmd.

Type docker

hit enter and you can know list of commands.

How to use docker and pull images and run on local-

Go to docker hub.

Search for the software.

Click docker original image on left side.

The screenshot shows a web browser with multiple tabs open. The active tab is 'https://hub.docker.com/search?q=jenkins+latest&badges=official'. The Docker Hub search interface is displayed, showing a single result for 'jenkins latest'. The result is from the 'Docker Official Image' repository. It includes a thumbnail of the Jenkins logo, the name 'jenkins', a 'Docker Official Images' badge, a note that it is 'DEPRECATED; use "jenkins/jenkins:its" instead', and metrics: Pulls (100M+), Stars (5705), and Last Updated (almost 7 years). The sidebar on the left shows filtering options for 'Products' (Images, Extensions, Plugins), 'Trusted content' (Docker Official Image checked, Verified Publisher, Sponsored OSS), and 'Categories'.

When you scroll down you can see run command.  
Docker run pulls image and runs or if image exist then just runs.

```
docker run -p 8080:8080 -p 50000:50000 jenkins:2.60.2
```

The terminal window shows the command 'docker run -p 8080:8080 -p 50000:50000 jenkins:2.60.2' being run. The output indicates that Docker is unable to find the image locally and is pulling it from the Docker Hub library/jenkins. The progress bar shows the download of the image, which is 170.9MB/183.7MB. The text 'karan19' is visible in the bottom right corner of the terminal window.

```
See "docker run --help".  
C:\Users\l1>docker run -p 8080:8080 -p 50000:50000 jenkins:2.60.2  
Unable to find image 'jenkins:2.60.2' locally  
2.60.2: Pulling from library/jenkins  
219d2e45b4af: Pull complete  
a482fbcef407: Pull complete  
988edaaff53b: Pull complete  
f44dc7c129fe: Pull complete  
f34a55a78629: Pull complete  
a0608417860f: Pull complete  
74ffbb683f1a: Pull complete  
11ab86bd63bc: Downloading [=====] 170.9MB/183.7MB  
a0e8d28e95f8: Download complete  
b5d5ad9f8a74: Download complete  
3caeaa7ba6c3e: Download complete  
09cfdf5fe294a: Download complete  
d1bd1db0a004: Download complete  
f0efffb33601d: Download complete  
304196f84fee: Download complete  
8ddc6878c5f0: Download complete  
5f2982263a84: Download complete  
2cbac31ea617: Download complete  
49f4de9f38ff: Download complete  
82aa91ec8318: Download complete  
-
```

Git hub actions-

Go to the required git hub repo.

Click actions.

Click set up workflow

The screenshot shows a GitHub repository page for 'karanAtreya1986/MO\_CypressLearningWithSSHPart2'. The 'Actions' tab is selected. A prominent banner at the top says 'Get started with GitHub Actions' with the subtext 'Build, test, and deploy your code. Make code reviews, branch management, and issue triage'. Below the banner is a link 'Skip this and set up a workflow yourself →'.

## Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and

Skip this and [set up a workflow yourself →](#)

If you want to run on multiple events, use

**On: [push, pull, merge]**

How to make additional containers and run in parallel-

**Containers: [1,2,3]**

To get the os name - search for github self hosted runner and then in that page search for how to add runner. There are some changes.

To create own runner this is the area:

Project – settings – actions – runners.

The screenshot shows the GitHub Actions Runner settings page for a repository named 'MO\_CypressLearningWithSSHPart2'. The left sidebar has a 'Runners' tab selected. The main area is titled 'Runners / Add new self-hosted runner · karanAtreya1986/MO\_CypressLearningWithSSHPart2'. It includes instructions for adding a self-hosted runner, selecting a 'Runner image' (macOS, Linux, Windows), 'Architecture' (x64), and a 'Download' section with shell commands for creating a folder and downloading the latest runner package.

Older mocha reports dont work properly or do not generate in github actions--  
Json report is generated but not html.

## ✖ Why the HTML report was blank

This typically happens due to one of the following:

### 1. Generated with `marge` CLI instead of `cypress-mochawesome-reporter`

If you're using `mochawesome` as the reporter, you must **not** use `marge` directly anymore. Use the full plugin:

 [cypress-mochawesome-reporter](#)

Using `marge` manually can cause HTML rendering issues due to missing assets or format mismatches.

### Recommended Fix — Use `cypress-mochawesome-reporter`

Update your workflow like this:

#### Install correct dependencies:

npm install --save-dev cypress-mochawesome-reporter

#### In `cypress.config.js`:

```
const { defineConfig } = require('cypress');

module.exports = defineConfig({
  reporter: 'cypress-mochawesome-reporter',
  reporterOptions: {
    reportDir: 'cypress/results',
    overwrite: false,
    html: true,
    json: true,
  },
  e2e: {
    setupNodeEvents(on, config) {
      require('cypress-mochawesome-reporter/plugin')(on);
      return config;
    }
  }
});
```

 In your `e2e` support file ([cypress/support/e2e.js](#)):  
import 'cypress-mochawesome-reporter/register';

 Then update your GitHub Actions step:

```
- name: Run Cypress tests
  uses: cypress-io/github-action@v6
  with:
    command: npx cypress run
    continue-on-error: true
```

You can then remove the `mochawesome-merge` and `merge` steps entirely if you're using the plugin — it generates clean HTML automatically per test run.

 Optional — Upload Reports

```
- name: Upload HTML report
  if: always()
  uses: actions/upload-artifact@v4
  with:
    name: cypress-mochawesome-report
    path: cypress/results
```

## Summary

### Issue

### Fix

|                   |  |
|-------------------|--|
| Blank HTML report | Likely due to using <code>marge</code> instead of correct plugin       |
| Correct JSON      | Yes – you have full test results                                       |
| Best Practice     | Use <code>cypress-mochawesome-reporter</code> and register it properly |

Absolutely! Here's a clear explanation of **why you can remove `mochawesome-merge` and `marge`** when using the [`cypress-mochawesome-reporter`](#) plugin — and how the plugin simplifies everything.

## Understanding the Difference

### The Old Manual Way (you're doing now)

If you use the plain `mochawesome` reporter:

1. Cypress creates **multiple JSON files** in `cypress/results/` — one per spec file.
2. You have to:
  -  **merge** them manually using `mochawesome-merge`
  -  **generate HTML** using `marge` from the merged JSON
3. These tools are **separate**, and not built into Cypress.  
It's a manual, multi-step process.

### The Plugin Way: `cypress-mochawesome-reporter`

When you use the `cypress-mochawesome-reporter` plugin, it:

- Hooks directly into Cypress test lifecycle
- Automatically collects all results across spec files
- Generates:
  - A single merged **JSON**

- And a single, beautiful **HTML report**
- **No need to merge manually**
- **No need to run `marge`**
- Everything is done *automatically at the end of the test run*

## 📁 What You'll See in `cypress/results/`

When using the plugin, after `npx cypress run`, you'll find:

```
cypress/results/
  └── mochawesome-report/
      ├── mochawesome.html   ←  Full test report
      ├── mochawesome.json
      └── assets/...          ← CSS/JS for the report
```

You can open `mochawesome.html` and get a full dashboard: test list, errors, screenshots, etc.

## ✓ What You Can Remove from Your Workflow

If you switch to `cypress-mochawesome-reporter`, **you can delete these**:

```
- name: Merge Mochawesome JSON reports
  run: npx mochawesome-merge "cypress/results/*.json" > mochawesome.json

- name: Generate HTML report from merged JSON
  run: npx marge mochawesome.json --reportDir=cypress/results
  --reportFilename=mochawesome-report.html
```

And instead, **just upload the whole report folder**:

```
- name: Upload test report
  if: always()
  uses: actions/upload-artifact@v4
  with:
    name: mochawesome-report
    path: cypress/results/mochawesome-report/
```

## Result

This gives you:

- Cleaner workflow
- No need for manual merging or extra dependencies
- Fully styled interactive HTML report

## Summary

| Step                        | Old Way<br>( <code>mochawesome</code> )  | Plugin<br>( <code>cypress-mochawesome-reporter</code> )                                       |
|-----------------------------|--|---|
| Merge JSON manually         |  Required                         |  Not needed  |
| Generate HTML manually      |  Required ( <code>merge</code> ) |  Not needed |
| Auto HTML + JSON report     |  No                             |  Yes       |
| Simpler GitHub Actions flow |  No                             |  Yes       |

Exit codes -

### 1. `exit code 14` — Cypress couldn't run or crashed

`npx cypress run` returning `exit code 14` usually means:

Cypress failed to `launch`, often due to missing plugins, misconfiguration, or runtime errors in `cypress.config.js`.

 This happens frequently when `cypress-mochawesome-reporter` is **installed but not registered correctly** in the config.

## 2. Report folder is missing `(cypress/results/mochawesome-report/)`

The warning:

No files were found with the provided path: cypress/results/mochawesome-report/

means the report **was never generated** — likely because Cypress never completed its run, so the reporter plugin never ran either.

## Fix Both Issues

Here's how to fix the root problem: **correct installation and registration of the reporter plugin.**

### Step 1: Install the plugin

In your project:

```
npm install --save-dev cypress-mochawesome-reporter
```

### Step 2: Update `cypress.config.js`

Make sure this file exists and looks like this:

```
const { defineConfig } = require('cypress');

module.exports = defineConfig({
  reporter: 'cypress-mochawesome-reporter',
  reporterOptions: {
    reportDir: 'cypress/results/mochawesome-report',
    overwrite: false,
    html: true,
    json: true
  },
  e2e: {
    setupNodeEvents(on, config) {
      require('cypress-mochawesome-reporter/plugin')(on);
    }
  }
});
```

```
        return config;
    }
}
});
```

## Step 3: Add this to your support file

At the top of your test support file (`cypress/support/e2e.js`):

```
import 'cypress-mochawesome-reporter/register';
```

## Step 4: Update GitHub Actions YAML

Here's your fixed and simplified `yml` using the plugin:

```
name: Cypress Tests

on: push

jobs:
  cypress-run:
    runs-on: ubuntu-24.04
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Install dependencies
        run: npm ci

      - name: Run Cypress tests
        id: cypress
        continue-on-error: true
        uses: cypress-io/github-action@v6
        with:
          command: npx cypress run

      - name: Upload Mochawesome report
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: mochawesome-report
```

```
path: cypress/results/mochawesome-report
```

- name: Fail if Cypress tests failed  
if: steps.cypress.outcome == 'failure'  
run: exit 1

## Summary

| Problem                   | Fix  |
|---------------------------|--|
| exit code 14              | Register <code>cypress-mochawesome-reporter</code> correctly |
| HTML report not generated | Because Cypress never finished / plugin not initialized      |
| Artifact upload warning   | Report folder didn't exist due to the failure above          |

Great question!

## Where to install `cypress-mochawesome-reporter`

You should run **Step 1** — the install command — in the **root directory of your Cypress project** (same place as your `package.json`).

### Here's how to do it:

#### Example structure of your project:

```
my-project/
├── cypress/
├── cypress.config.js
├── package.json
└── ...
```

#### In your terminal:

```
cd my-project
npm install --save-dev cypress-mochawesome-reporter
```

## Why it goes in your project

This ensures:

- The reporter is listed in your `package.json` under `devDependencies`
- It's available during your GitHub Actions run (`npm ci` will install it)

## After installing:

You should see this in `package.json`:

```
"devDependencies": {  
  "cypress-mochawesome-reporter": "^3.4.0", // or latest  
  ...  
}
```

This [config.js](#) file –

```
e2e: { setupNodeEvents(on, config) { require('cypress-mochawesome-reporter/plugin')(on);  
return config; } is return mandatory?
```

Great question!

## Is `return config` mandatory in `setupNodeEvents`?

Yes — in most cases, it's best practice to include `return config`, especially if:

- You're using plugins that **modify or rely on** the `config` object (like environment variables, base URLs, etc.)
- You're chaining or extending the Cypress config with other settings

## What happens if you don't return `config`?

If you omit `return config`, Cypress may not apply any modifications made to the configuration during plugin setup. Some plugins (like `cypress-mochawesome-reporter`) **may still work**, but others that change config dynamically **won't**.

 Example with proper return:

```
e2e: {  
  setupNodeEvents(on, config) {  
    require('cypress-mochawesome-reporter/plugin')(on);  
    return config; //  ensures the updated config is passed back to Cypress  
  }  
}
```

}

## ✓ Bottom Line

- ✅ Returning `config` is **safe and recommended**
- ❌ Not returning it might break plugin behavior in certain cases
- ✓ For `cypress-mochawesome-reporter`, it may still work — but better to **always return it**

## ✓ Why it worked without `npm ci`

If you're running Cypress **locally**, or even in CI (like GitHub Actions), and you **didn't run `npm ci`**, here's why it may still work:

### 🧠 1. You may have already installed dependencies using `npm install`

If you've previously run:

```
npm install
```

Then your `node_modules/` directory already contains everything needed — including `cypress-mochawesome-reporter`.

→ So running Cypress tests afterward just works.

### 🧠 2. In GitHub Actions, dependencies might be cached or auto-installed

If your GitHub Actions workflow **does not run `npm ci`**, but the action still works, then:

- You might be committing your `node_modules/` (not ideal).
- You might be using a base Docker image or action that installs Cypress + reporter already.
- GitHub Actions may still run `npm install` **implicitly** if you don't override it.

### 3. Cypress GitHub Action can install dependencies for you

If you're using:

- uses: cypress-io/github-action@v6

That action can **automatically install dependencies** unless you explicitly tell it not to.

 From [official docs](#):

If your project does not have a `cypress` binary already installed, the action will automatically install it using `npm install`.

### Best Practice: Still Add `npm ci`

Even if it "works", best practice in CI is:

- run: `npm ci`

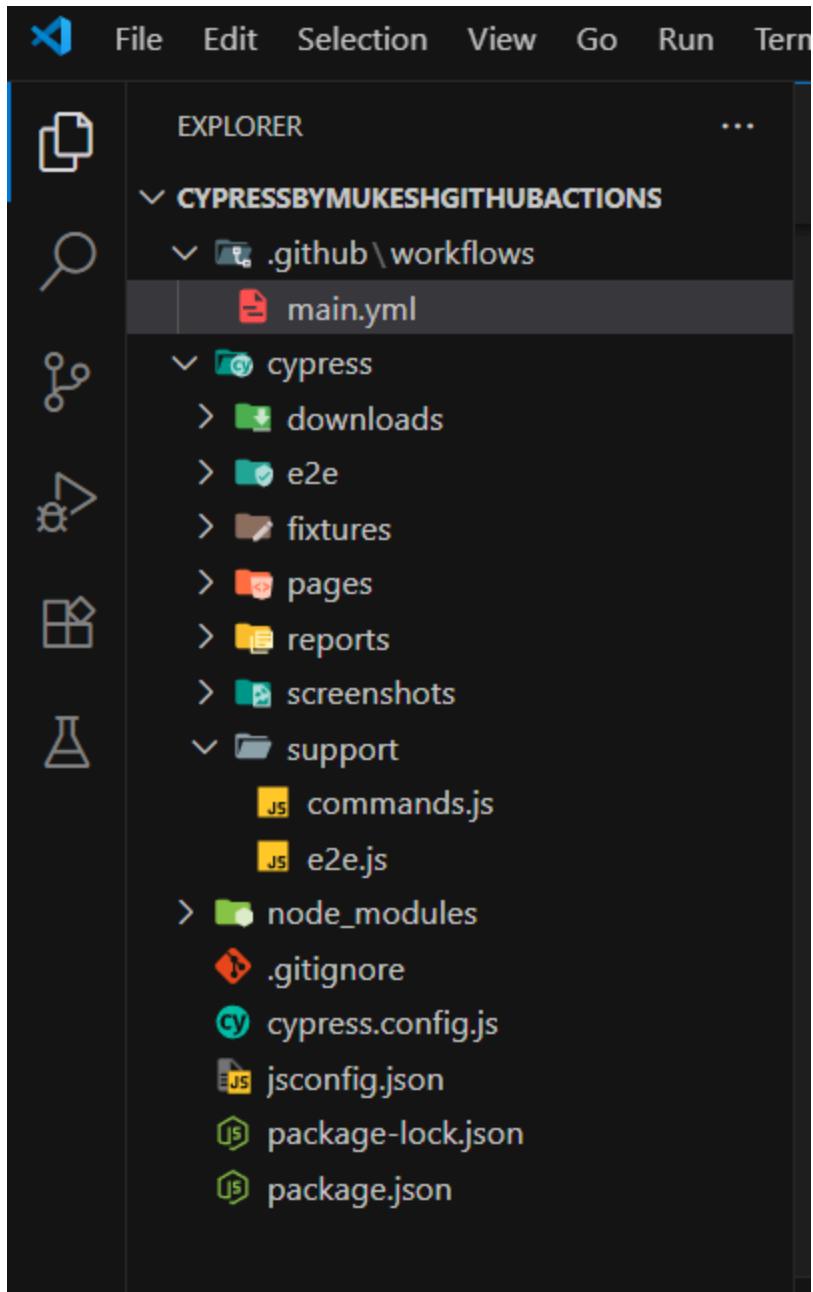
Because:

- It ensures clean, reproducible installs from `package-lock.json`
- It's faster and more predictable than `npm install`
- It avoids dependency drift

### Summary

| Reason it worked                     | Explanation  |
|--------------------------------------|--|
| Already ran <code>npm install</code> | Dependencies were in <code>node_modules</code>     |
| Cypress GitHub Action helped         | Automatically installs packages unless told not to |
| Local dev environment                | Had all the right packages already installed       |

Yaml file –



```
1 name: Cypress Tests
2
3 on: push
4
5 jobs:
6   cypress-run:
7     runs-on: ubuntu-24.04
8     steps:
9       - name: Checkout
10      uses: actions/checkout@v4
11
12     - name: Cypress run
13       id: cypress
14       uses: cypress-io/github-action@v6
15       with:
16         #run all the files
17         command: npx cypress run
18         #continue even if scripts fail
19         continue-on-error: true
20
21
22
23     - name: Upload Mochawesome reports
24       #always upload the reports irrespective of pass fail.
25       if: always()
26       uses: actions/upload-artifact@v4
27       ##report name and path where to upload.
28       with:
29         name: mochawesome-reports
30         path: cypress/results/mochawesome-report
31
```

codesnap.dev

Git ignore file—



1 node\_modules /  
2

[codesnap.dev](https://codesnap.dev)

## Cypress [config.js](#) -

```
1 const { defineConfig } = require("cypress");
2
3 module.exports = defineConfig({
4
5   reporter: 'cypress-mochawesome-reporter',
6   reporterOptions: {
7     reportDir: 'cypress/results/mochawesome-report',
8     overwrite:false,
9     html: true,
10    json: true,
11  },
12 },
13 e2e: {
14   setupNodeEvents(on, config) {
15     // implement node event listeners here
16     require('cypress-mochawesome-reporter/plugin')(on);
17     return config; // ✅ ensures the updated config plugins is passed back to Cypress
18   }
19 }
20
21 // watchForFileChanges:true,
22 // retries:{
23 //   runMode:2,
24 //   openMode:3
25 // }
26 // });
27 });
28
```

codesnap.dev

## Package.json –

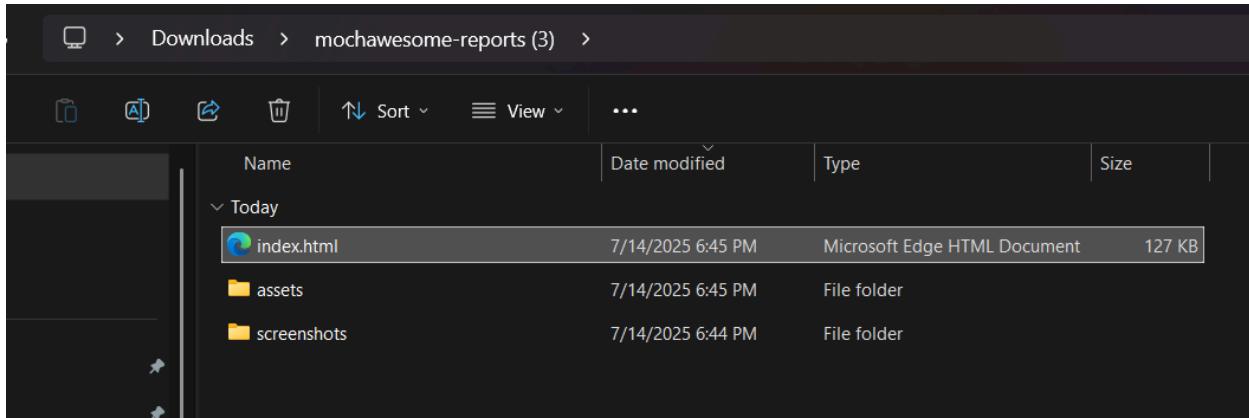
```
1  {
2    "name": "cypressbymukesh",
3    "version": "1.0.0",
4    "main": "index.js",
5    "scripts": {
6      "test": "echo \"Error: no test specified\" && exit 1"
7    },
8    "author": "",
9    "license": "ISC",
10   "devDependencies": {
11     "cypress": "^14.5.1",
12     "cypress-file-upload": "^5.0.8",
13     "cypress-iframe": "^1.0.1",
14     "cypress-mochawesome-reporter": "^3.8.4",
15     "cypress-xpath": "^2.0.1",
16     "mochawesome": "^7.1.3",
17     "mochawesome-merge": "^4.4.1",
18     "mochawesome-report-generator": "^6.2.0"
19   },
20   "dependencies": {
21     "cypress-real-events": "^1.14.0"
22   },
23   "keywords": [],
24   "description": ""
25 }
26
```

codesnap.dev

Download the artifact, then unzip it then check index.html.

Opening index.html directly from the zipped file wont open the report.

Unzip and open-



### 🧱 Code Block:

```
- name: Fail if Cypress tests failed
  if: steps.cypress.outcome == 'failure'
  run: exit 1
```

### ⌚ What does it do?

This step **explicitly fails the GitHub Actions job** if the Cypress tests failed — even if you previously allowed them to "continue on error."

It's a way to:

- Let **report steps run no matter what**
- Still make the workflow **fail overall** if Cypress failed

### 🔧 Why is it needed?

Earlier in your workflow, you likely used:

```
- name: Run Cypress
  id: cypress
  continue-on-error: true
  ...
```

This is good because:

- It allows the pipeline to **continue to reporting**, even if tests fail
- But GitHub Actions will **not fail the job** unless you manually do it later

So we fix that by checking:

```
if: steps.cypress.outcome == 'failure'
```

...and force the workflow to fail with:

```
exit 1
```



### Think of it like this:

- `continue-on-error: true` lets other steps run (like generating and uploading test reports)
- `exit 1` ensures your CI still **fails if tests failed**, so you don't accidentally merge broken code



### When should you use it?

Use this \*\*any time you want to:

1. Run Cypress,
2. Always generate reports/screenshots/videos,
3. Still fail the job if tests didn't pass.

Run single file-

[Edit file](#)

[Preview changes](#)

```
1 name: CypressOnChrome
2
3 on: push
4
5 jobs:
6   chrome:
7     runs-on: ubuntu-20.04
8     steps:
9       - name: Checkout
10      uses: actions/checkout@v2
11      - name: RunOnChrome
12      uses: cypress-io/github-action@v4
13      with:
14        spec: cypress/e2e/firstAPICall.cy.js
15      - name: Upload artifacts
16      uses: actions/upload-artifact@v2
17      if: always()
18      with:
19        name: cypress-execution-report
20        path: cypress/reports/html
21        retention-days: 10
22
23
```

Clearer picture –

```
1 name: Cypress Tests
2
3 on: push
4
5 jobs:
6   cypress-run:
7     runs-on: ubuntu-24.04
8     steps:
9       - name: Checkout
10      uses: actions/checkout@v4
11
12     - name: Cypress run
13       id: cypress
14       uses: cypress-io/github-action@v6
15       #run specific cy file.
16       with:
17         spec: cypress/e2e/first.cy.js
18       continue-on-error: true
19
20
21
22     - name: Upload Mochawesome reports
23       if: always()
24       uses: actions/upload-artifact@v4
25       with:
26         name: mochawesome-reports
27         path: cypress/results/mochawesome-report
28         #how many days report to be saved in artifacts.
29         retention-days: 10
30
```

codesnap.dev

Another way to run specific spec file - npm run <scriptname>

How to add more containers for parallel runs-

When we dont use the cypress dashboard and we dont separate at job level we get this error -  
the test runs on five containers but reports wont be generated.

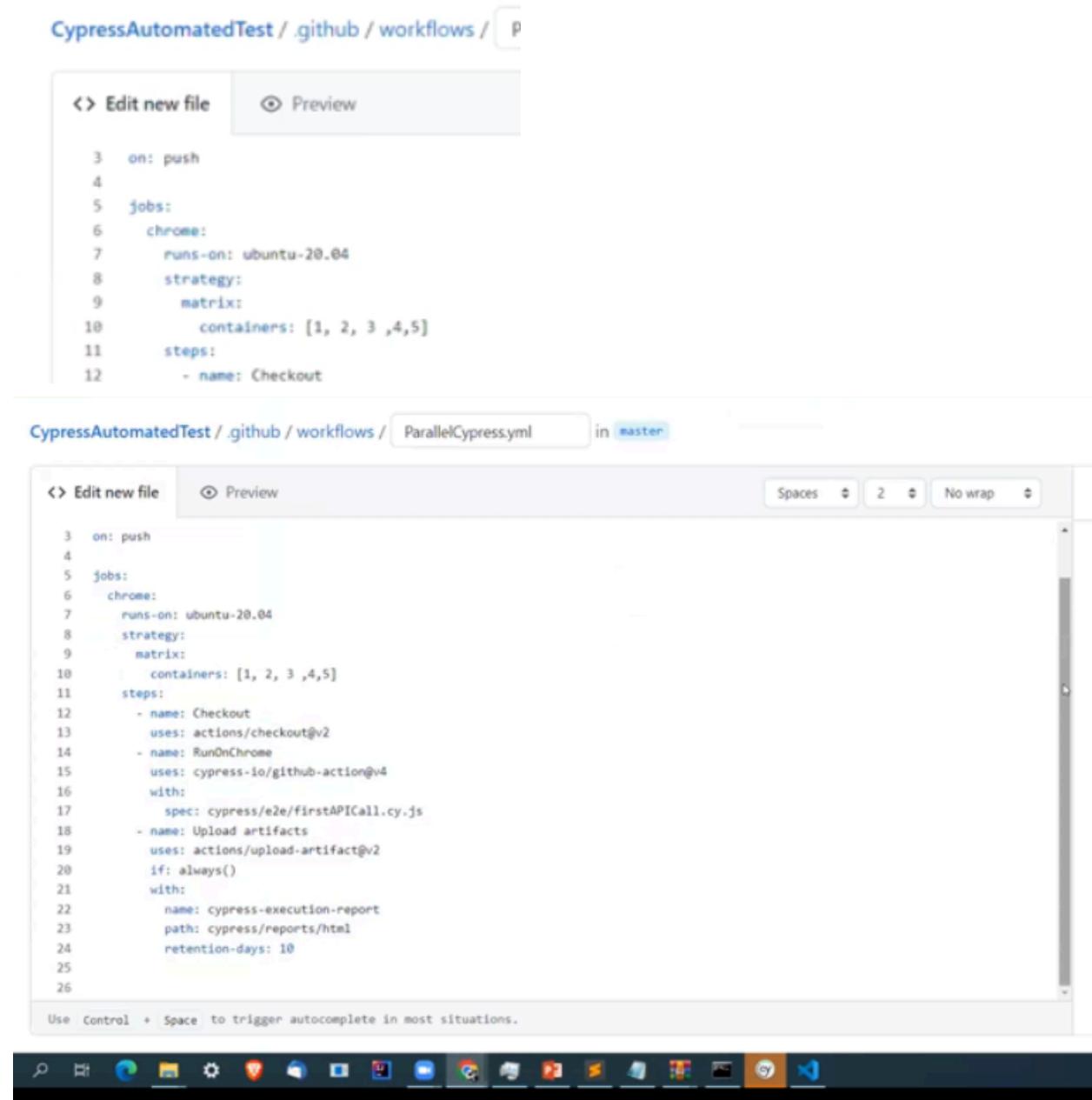
```
1 name: Cypress Tests
2
3 on: push
4
5 jobs:
6   cypress-run:
7     runs-on: ubuntu-24.04
8     #how many jobs to run in parallel
9     #if we use containers we need cypress dashboard to know
10    #which jobs run where.
11    strategy:
12      matrix:
13        containers: [1,2,3,4,5]
14
15 steps:
16   - name: Checkout
17     uses: actions/checkout@v4
18
19   - name: Cypress run
20     id: cypress
21     uses: cypress-io/github-action@v6
22     with:
23       spec: cypress/e2e/first.cy.js
24     continue-on-error: true
25
26
27
28   - name: Upload Mochawesome reports
29     if: always()
30     uses: actions/upload-artifact@v4
31     with:
32       name: mochawesome-reports
33       path: cypress/results/mochawesome-report
34       retention-days: 10
35
```

codesnap.dev

We get this error message in the upload mocha reports-

Error: Failed to CreateArtifact: Received non-retryable error: Failed request: (409)  
Conflict: an artifact with this name already exists on the workflow run

Create new file parallelcypress.yaml.  
Created above.

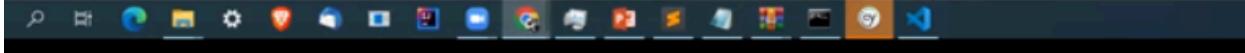


```
3  on: push
4
5  jobs:
6    chrome:
7      runs-on: ubuntu-20.04
8      strategy:
9        matrix:
10       containers: [1, 2, 3 ,4,5]
11     steps:
12       - name: Checkout
```

CypressAutomatedTest/.github/workflows / ParallelCypress.yml in master

```
3  on: push
4
5  jobs:
6    chrome:
7      runs-on: ubuntu-20.04
8      strategy:
9        matrix:
10       containers: [1, 2, 3 ,4,5]
11     steps:
12       - name: Checkout
13         uses: actions/checkout@v2
14       - name: RunOnChrome
15         uses: cypress-io/github-action@v4
16         with:
17           spec: cypress/e2e/firstAPICall.cy.js
18       - name: Upload artifacts
19         uses: actions/upload-artifact@v2
20         if: always()
21         with:
22           name: cypress-execution-report
23           path: cypress/reports/html
24           retention-days: 10
25
26
```

Use Control + Space to trigger autocomplete in most situations.



In cypress for parallel runs, we need to use github actions, no internal feature given by cypress.

Cypress dashboard-  
Where to run which test on which container.  
To run in parallel.  
Open cypress.

Npx cypress open.

```
C:\Users\karan\Desktop\cypressByMukeshGitHubActions>npx cypress open

DevTools listening on ws://127.0.0.1:59195/devtools/browser/b4aab81e-813a-401c-a9c0-1cec4561c538
```

Logout and login to cypress dashboard.



Create organization.

## Let's create an organization

This organization will be the workspace for your company or team

Organization name

It's best to use your company or team name

Organization website (optional)

This helps us validate your organization

[Create organization](#)

Looking for a different organization? Contact your Cypress admin for an invitation or [try another log in](#).

One organisation can have one or more projects.

Create new project.

# Create a new project

Projects act as a unique workspace to group together the runs from your repository.

You can also create projects from the Cypress app →

**Project Name**  
The name of the project, product, or workspace that will be associated to each run.

cypress practice 1

**Project Access**

**Private**  
Only invited users can view recorded test results.

**Public**  
Anyone on the internet will have view-only access.

**Create project**

Below is not a clear picture.

**Project Name**  
The name of the project, product, or workspace that will be associated to each run.

CypressEveningBatch

**CI Provider**  
Enter the CI provider you're using, which we'll use to guide the experience in the following step.

GitHub Actions X ▼

**Project Access**

**Private** — Only invited users can view recorded test results.

**Public** — Anyone can view recorded test results.

**Create project** **Back**

Create project.

You will get this project id -

# Project setup

Update your Cypress config file.

The screenshot shows a step in the Cypress Project Setup wizard titled "Add your project ID". It includes a link to "Check the docs." Below is a code editor window showing a snippet of `cypress.config.js`:

```
1 module.exports = {
2   projectId: "arexjt",
3   // ...rest of the Cypress project config
4 }
```

At the bottom right of the code editor is a blue "Next" button. To its left is a checkbox labeled "Ok, I added my project ID".

Go to cypress [config.js](#) file-  
Add here.

Add this project id.

The screenshot shows a code editor with a tab for `cypress.config.js`. The code is as follows:

```
1 const { defineConfig } = require("cypress");
2
3 module.exports = defineConfig([
4   projectId: "arexjt",
5 ])
```

Select ci provider-

# Project setup

Cypress Cloud is made for continuous integration.

Choose your CI Provider  
We'll customize the setup instructions.

The screenshot shows a user interface for choosing a Continuous Integration provider. At the top, a title says "Choose your CI Provider" with a subtitle "We'll customize the setup instructions." Below this, there is a grid of seven boxes, each representing a different CI provider. The first box, "GitHub Actions", has a green border around it, indicating it is the selected or recommended provider. The other providers shown are "GitLab CI", "CircleCI", "Azure", "Bitbucket", "AWS CodeBuild", and "Manual setup". At the bottom right of the grid, there are two buttons: "Back" and "Next".

Not clear picture below-

```
n Terminal Help
...
  cypress.config.js •
  cypress.config.js > [●] <unknown> > ↵
    1 const { defineConfig } = require('cypress');
    2
    3 module.exports = defineConfig({
    4   projectId: "n6k1eq",
    5
    6
  
```

A screenshot of a terminal window displaying the code of a file named "cypress.config.js". The code is a single-line JSON object definition using the Cypress configuration API. It includes a "projectId" key set to the value "n6k1eq". The terminal interface shows standard navigation keys like "n", "Terminal", and "Help" at the top, and a scroll bar on the left.

Add cypress record key changes to github actions--  
There are some changes and multiple steps.

# Project setup

Trigger a run with GitHub Actions.

## Try it first

Record a run in your terminal.

```
$ npx cypress run --record --key 8f946559-7c18-4216-ba0b-87f633a78703
```



## GitHub Actions Setup

Run Cypress tests every time you push.

- 1 Set `CYPRESS_RECORD_KEY` in your GitHub repo → Settings → Secrets → Actions:

```
CYPRESS_RECORD_KEY 8f946559-7c18-4216-ba0b-87f633a78703
```



- 2 Then add this file to your repository:

```
.github/workflows/cypress.yml
```



```
1 name: Cypress Tests
2 on: [push]
3 jobs:
4   cypress-run:
5     runs-on: ubuntu-latest
6     # Runs tests in parallel with matrix strategy https://docs.cypress.io/guides/gui
7     # https://docs.github.com/en/actions/using-jobs/using-a-matrix-for-your-jobs
8     # Also see warning here https://github.com/cypress-io/github-action#parallel
9     strategy:
10       fail-fast: false # https://github.com/cypress-io/github-action/issues/48
11       matrix:
12         containers: [1, 2] # Uses 2 parallel instances
13       steps:
14         - name: Checkout
15           uses: actions/checkout@v4
16         - name: Cypress run
17           # Uses the official Cypress GitHub action https://github.com/cypress-io/gith
18           uses: cypress-io/github-action@v6
```

```
18   uses: cypress-io/github-action@v6
19   with:
20     # Starts web server for E2E tests - replace with your own server invocation
21     # https://docs.cypress.io/guides/continuous-integration/introduction#Bootstrap
22     start: npm start
23     wait-on: 'http://localhost:3000' # Waits for above
24     # Records to Cypress Cloud
25     # https://docs.cypress.io/guides/cloud/projects#Set-up-a-project-to-record
26     record: true
27     parallel: true # Runs test in parallel using settings above
28   env:
29     # For recording and parallelization to work you must set your CYPRESS_RECORD_KEY
30     # in GitHub repo → Settings → Secrets → Actions
31     CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}
32     # Creating a token https://docs.github.com/en/authentication/keeping-your-
33     GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

 [Read our guide to GitHub Actions](#)

[Back](#)

 Waiting for run...

#### NOTE-

Containers: [windows, mac, linux] can also come.

Containers: [chrome, firefox] can also come.

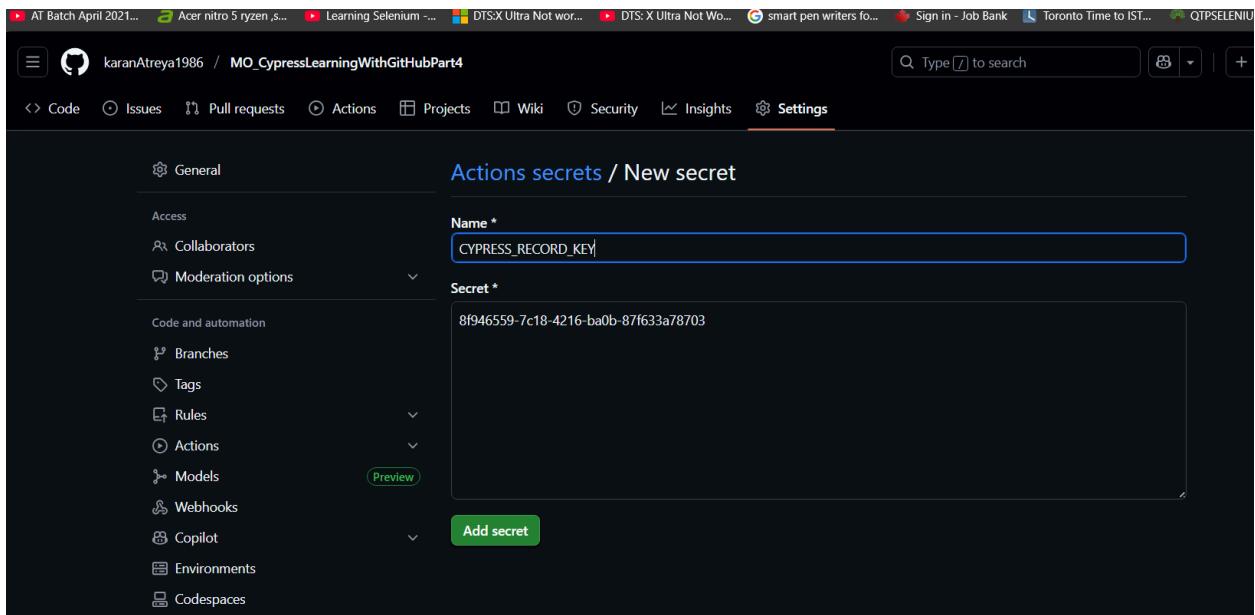
Go to the corresponding repo.

Click settings.

Click actions /secrets in left side.

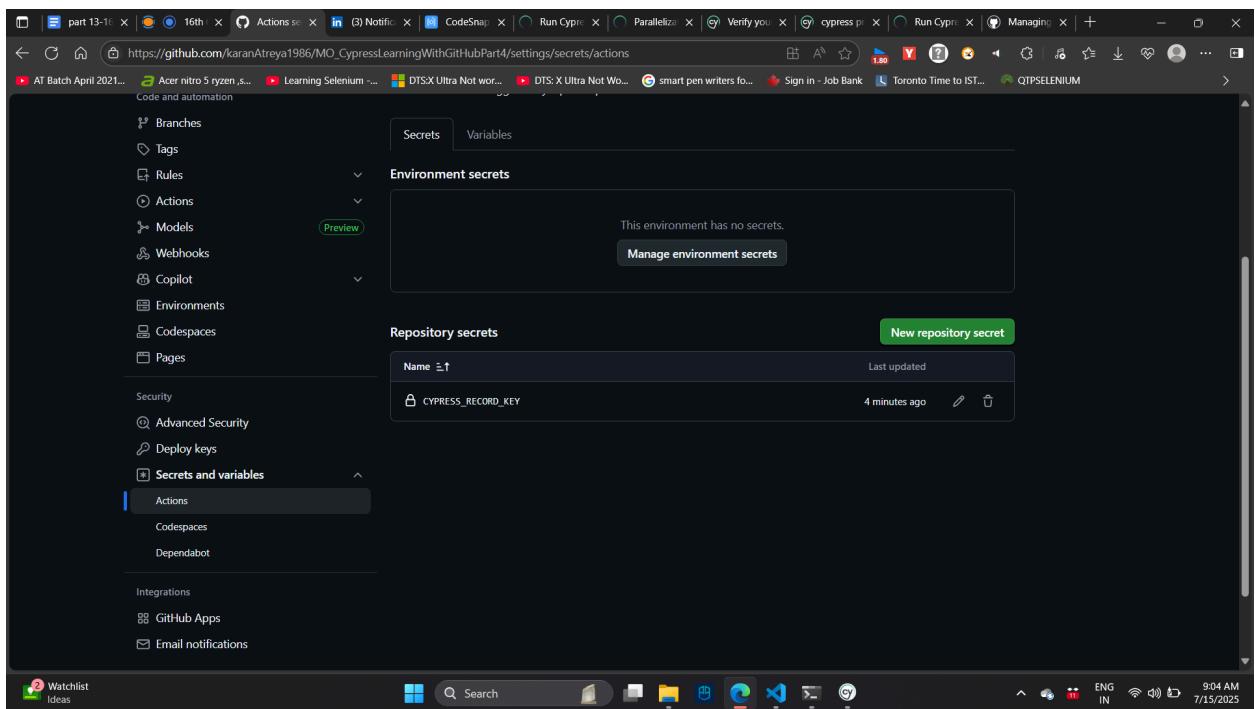
Enter new secret.

Store cypress secret here. Click add secret.



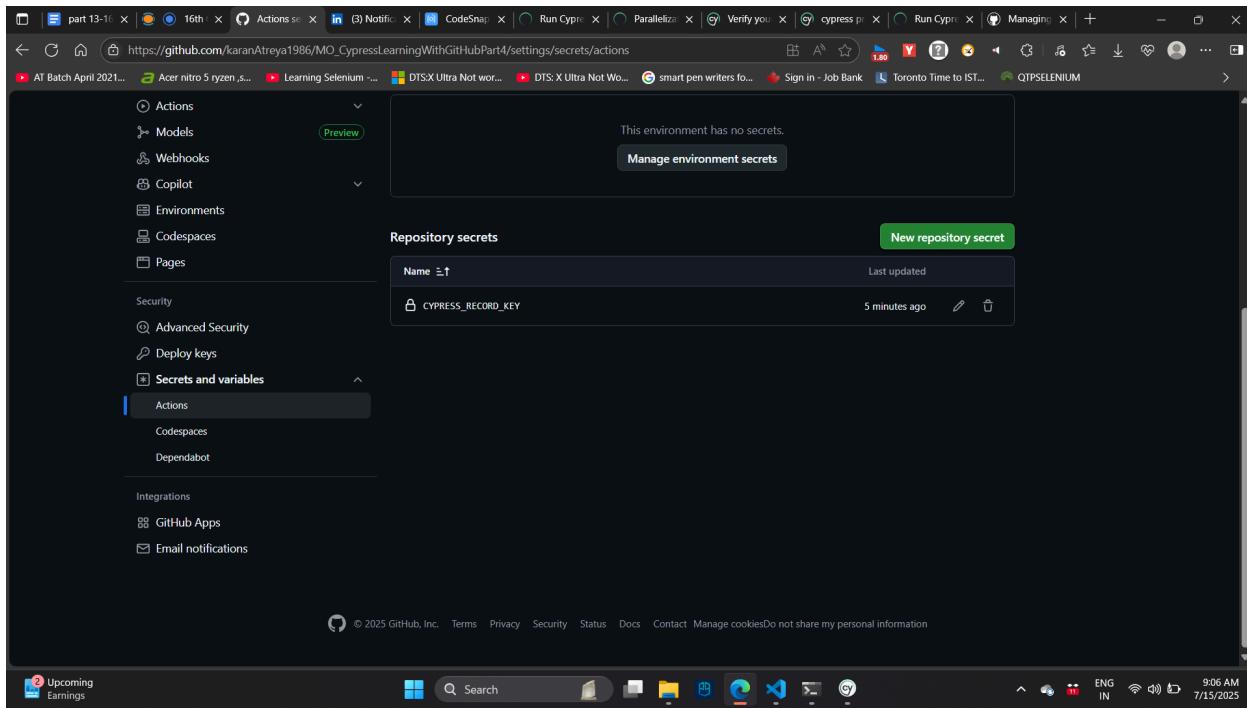
The screenshot shows the GitHub Actions secrets creation interface. On the left, a sidebar lists various GitHub features like Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main area is titled "Actions secrets / New secret". It has two fields: "Name \*" containing "CYPRESS\_RECORD\_KEY" and "Secret \*" containing the value "8f946559-7c18-4216-ba0b-87f633a78703". A green "Add secret" button is at the bottom.

ocd=

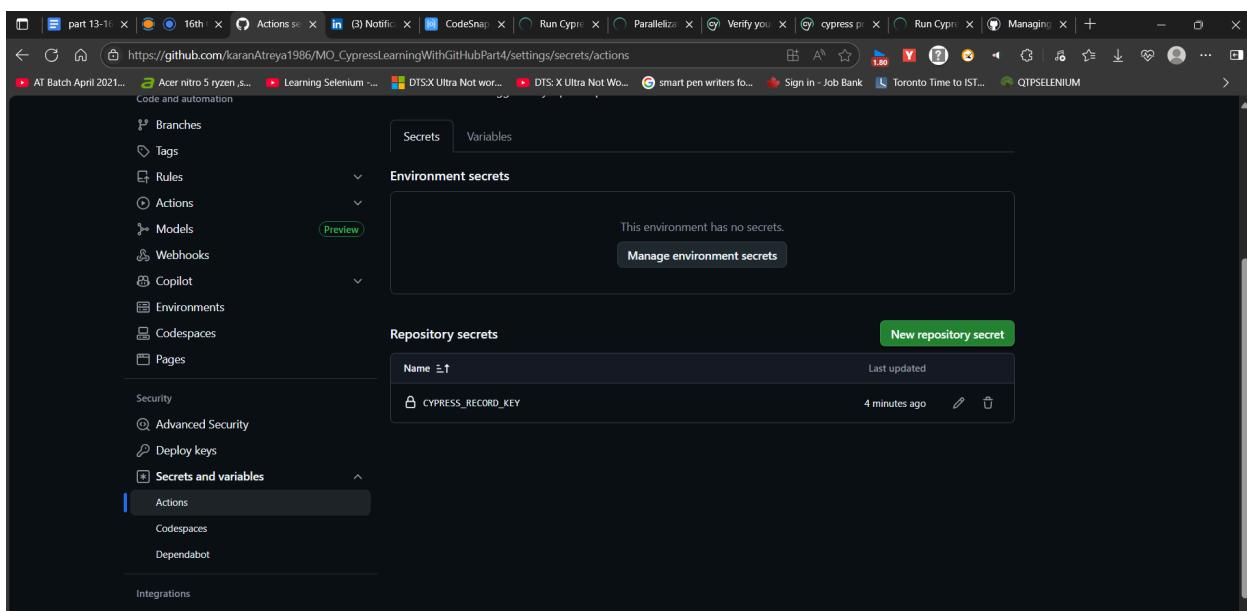


The screenshot shows the GitHub repository settings for "karanAtreya1986/MO\_CypressLearningWithGitHubPart4". The left sidebar shows "Actions" selected under "Secrets and variables". The main area displays "Environment secrets" (empty) and "Repository secrets" with one entry: "CYPRESS\_RECORD\_KEY" last updated 4 minutes ago. A green "New repository secret" button is visible.

Settings – secrets – actions –



ocd=



Set up github personal access token –

Github settings – developer settings - personal access tokens – tokens classic –

The screenshot shows the GitHub Developer Settings page for managing personal access tokens. The URL is https://github.com/settings/tokens. The left sidebar has options for GitHub Apps, OAuth Apps, Personal access tokens (which is expanded), and Fine-grained tokens. The main area is titled "Personal access tokens (classic)". It displays a message: "No personal access token created". Below this, it says "Need an API token for scripts or testing? Generate a personal access token for quick access to the GitHub API." A "Generate new token" button is present, along with a link to the GitHub API documentation.

This screenshot is identical to the one above, showing the GitHub Developer Settings page for managing personal access tokens. The URL is https://github.com/settings/tokens. The left sidebar has options for GitHub Apps, OAuth Apps, Personal access tokens (which is expanded), and Fine-grained tokens. The main area is titled "Personal access tokens (classic)". It displays a message: "No personal access token created". Below this, it says "Need an API token for scripts or testing? Generate a personal access token for quick access to the GitHub API." A "Generate new token" button is present, along with a link to the GitHub API documentation.

## New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

### Note

GITHUB\_TOKEN

What's this token for?

### Expiration

30 days (Aug 14, 2025) ▾

The token will expire on the selected date

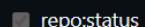
### Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)



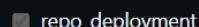
repo

Full control of private repositories



repo:status

Access commit status



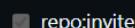
repo\_deployment

Access deployment status



public\_repo

Access public repositories



repo:invite

Access repository invitations

Generated token—

her scopes. Only the minimum set of necessary scopes has been saved.

## Personal access tokens (classic)

Generate new token ▾

Tokens you have generated that can be used to access the [GitHub API](#).

GITHUB\_TOKEN — repo

Never used

Delete

Expires on **Thu, Aug 14 2025**.

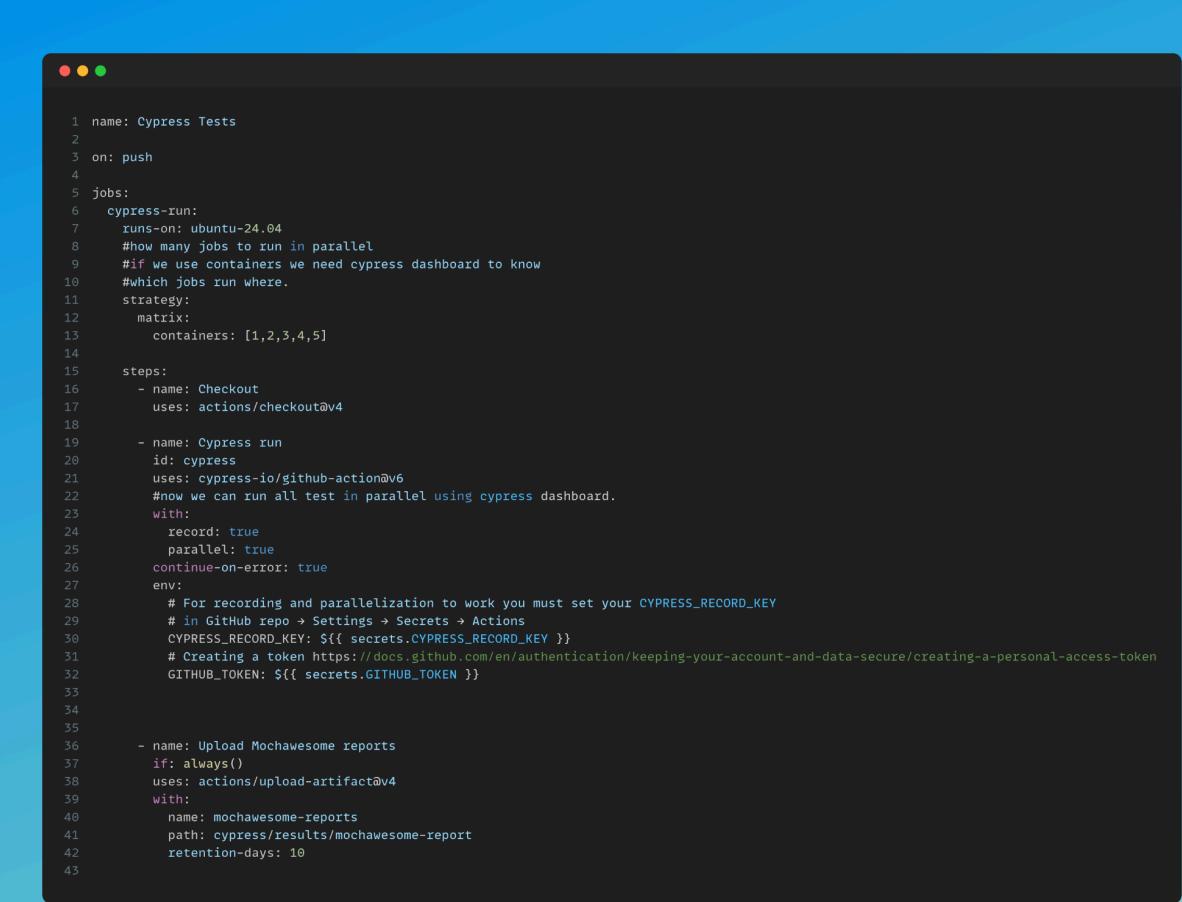
Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Below one is unclear screenshot.

<> Edit new file    Preview

```
7   runs-on: ubuntu-20.04
8   strategy:
9     matrix:
10    containers: [1, 2, 3 ,4,5]
11  steps:
12    - name: Checkout
13      uses: actions/checkout@v2
14    - name: RunOnChrome
15      uses: cypress-io/github-action@v4
16      with:
17        record: true
18        parallel: true
19        group: 'Actions example'
20      env:
21        CYPRESS_RECORD_KEY: 909af02f-29b9-4d4d-af90-815a9b55e30d
22    - name: Upload artifacts
```

Clear screenshot-



The screenshot shows a GitHub Actions workflow configuration in a dark-themed code editor. The workflow is named 'Cypress Tests' and triggers on push events. It contains a single job named 'cypress-run' which runs on an Ubuntu 24.04 container. The job uses the 'actions/checkout@v4' action to checkout the code. It then runs a Cypress test using the 'cypress-io/github-action@v6' action. The configuration includes parallel execution with 5 containers and recording tests. Environment variables set include CYPRESS\_RECORD\_KEY and GITHUB\_TOKEN. Finally, it uploads Mochawesome reports using the 'actions/upload-artifact@v4' action.

```
1 name: Cypress Tests
2
3 on: push
4
5 jobs:
6   cypress-run:
7     runs-on: ubuntu-24.04
8     #how many jobs to run in parallel
9     #if we use containers we need cypress dashboard to know
10    #which jobs run where.
11    strategy:
12      matrix:
13        containers: [1,2,3,4,5]
14
15  steps:
16    - name: Checkout
17      uses: actions/checkout@v4
18
19    - name: Cypress run
20      id: cypress
21      uses: cypress-io/github-action@v6
22      #now we can run all test in parallel using cypress dashboard.
23      with:
24        record: true
25        parallel: true
26        continue-on-error: true
27      env:
28        # For recording and parallelization to work you must set your CYPRESS_RECORD_KEY
29        # in GitHub repo → Settings → Secrets → Actions
30        CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}
31        # Creating a token https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token
32        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
33
34
35
36    - name: Upload Mochawesome reports
37      if: always()
38      uses: actions/upload-artifact@v4
39      with:
40        name: mochawesome-reports
41        path: cypress/results/mochawesome-report
42        retention-days: 10
43
```

codesnap.dev

No difference between web code snap and vs code snap-

```
● ● ●
1 name: Cypress Tests
2
3 on: push
4
5 jobs:
6   cypress-run:
7     runs-on: ubuntu-24.04
8     #how many jobs to run in parallel
9     #if we use containers we need cypress dashboard to know
10    #which jobs run where.
11    strategy:
12      matrix:
13        containers: [1,2,3,4,5]
14
15 steps:
16   - name: Checkout
17     uses: actions/checkout@v4
18
19   - name: Cypress run
20     id: cypress
21     uses: cypress-io/github-action@v6
22     #now we can run all test in parallel using cypress dashboard.
23     with:
24       record: true
25       parallel: true
26       continue-on-error: true
27     env:
28       # For recording and parallelization to work you must set your CYPRESS_RECORD_KEY
29       # in GitHub repo → Settings → Secrets → Actions
30       CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}
31       # Creating a token https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token
32       GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
33
34
35
36   - name: Upload Mochawesome reports
37     if: always()
38     uses: actions/upload-artifact@v4
39     with:
40       name: mochawesome-reports
41       path: cypress/results/mochawesome-report
42       retention-days: 10
43
```

Now cypress dashboard can connect with github.

After run check in the dashboard–

The screenshot shows a GitHub Actions test run for a job named "Update cypress.config.js". The run has completed 33 tests and 18 specs. A message indicates that all specs are complete. Below this, there is an "Actions example" section with a note to add 7 machines to CL for faster execution. The main area displays five machines (Machine 1 to Machine 5) with their status: Machine 1 failed, Machine 2 succeeded, Machine 3 succeeded, Machine 4 succeeded, and Machine 5 failed. A vertical bar on the left is labeled "Failed".

To run on multiple browsers-

Under same job create different step for chrome, ff etc.

To run parallelly on different browsers-

Create different jobs one for each browser in the same yaml file.

To run on different environment like qa, staging etc-

Create three steps in one job for each environment and run the script for the specific environment. OR

Three different jobs for each environment.

OR

Three yaml's for running on each environment.

Just change highlighted line with browser name in each steps:

At 1 contributor

23 lines (19 sloc) | 501 Bytes

```
1 name: CypressOnChrome
2
3 on: push
4
5 jobs:
6   chrome:
7     runs-on: ubuntu-20.04
8     steps:
9       - name: Checkout
10      uses: actions/checkout@v2
11      - name: RunOnChrome
12        uses: cypress-io/github-action@v4
13        with:
14          spec: cypress/e2e/firstAPICall.cy.js
15        - name: Upload artifacts
16          uses: actions/upload-artifact@v2
17          if: always()
18          with:
19            name: cypress-execution-report
20            path: cypress/reports/html
21            retention-days: 10
22
--
```

## Mukesh notes-

16 Oct Notes

1- Basic Workflow which will run all cypress test using single container

```
name: Chrome

on: push

jobs:
  chrome:
    runs-on: ubuntu-18.04
    steps:
      - name: Checkout
        uses: actions/checkout@v1
      - name: Chrome
        uses: cypress-io/github-action@v4
        timeout-minutes: 10
```

```

- name: Upload artifacts
  uses: actions/upload-artifact@v2
  if: always()
  with:
    name: cypress-execution-report
    path: cypress/reports/html
    retention-days: 10

```

2- Basic Workflow which will run specific spec file test using single container

```

name: Chrome

on: push

jobs:
  chrome:
    runs-on: ubuntu-18.04
    steps:
      - name: Checkout
        uses: actions/checkout@v1
      - name: Chrome
        uses: cypress-io/github-action@v4
        timeout-minutes: 10
        with:
          spec: cypress/e2e/firstAPICall.cy.js
      - name: Upload artifacts
        uses: actions/upload-artifact@v2
        if: always()
        with:
          name: cypress-execution-report
          path: cypress/reports/html

```

3- Workflow which will run all cypress test /spec file in parallel mode

Precondition before running test in parallel

- 1- Open cypress
  - 2- Login to cypress dashboard using github
  - 3- Create org
  - 4- Create project
  - 5- Copy project id and record key
  - 6- Add project id in cypress.config.js
- Sample
- ```

projectId: "szztj7"

```
- 7 - Use record key in workflow

```

name: Chrome

on: push

jobs:
  chrome:
    runs-on: ubuntu-18.04
    strategy:
      matrix:
        containers: [1, 2, 3, 4, 5]
    steps:
      - name: Checkout
        uses: actions/checkout@v1
      - name: Chrome
        uses: cypress-io/github-action@v4
        timeout-minutes: 10
        with:
          record: true
          parallel: true
          group: 'Actions example'
      env:
        CYPRESS_RECORD_KEY: please use recordkey
      - name: Upload artifacts
        uses: actions/upload-artifact@v2
        if: always()
        with:
          name: cypress-execution-report
          path: cypress/reports/html
          retention-days: 10

```

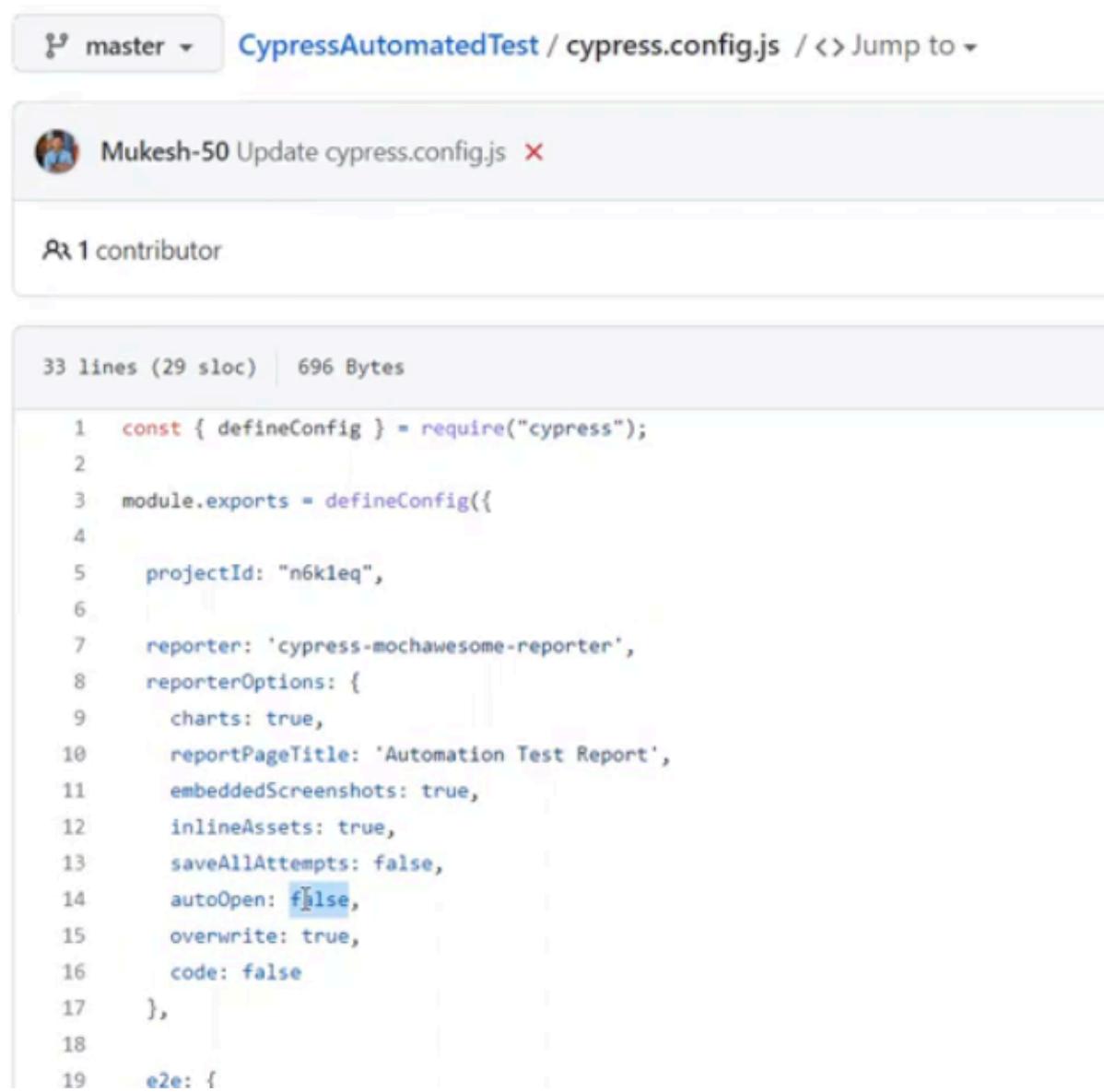
Some link to refer

- 1- Cypress Github Actions-
   
<https://docs.cypress.io/guides/continuous-integration/github-actions>
- 2- Parallel-
   
<https://docs.cypress.io/guides/guides/parallelization>
- 3- Cypress Github Actions -
   
<https://github.com/cypress-io/github-action#artifacts>
- 4- Github Hosted Runner -
   
<https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners>
- 5- All workflow -
   
<https://github.com/Mukesh-50/CypressAutomatedTest/tree/master/.github/workflows>

6- Docker Step By Step -

<https://www.youtube.com/watch?v=pBTH9FsgQuc&list=PL6f1ErFppaj1quyeiIF8Rz7n1BPOZTk6d>

MUKESH SIMPLE SOLUTION TO CREATE HTML STUCK PROBLEM IN GITHUB-



The screenshot shows a GitHub pull request interface. At the top, there's a header with a dropdown for 'master' and a link to 'CypressAutomatedTest / cypress.config.js'. Below the header, a commit card is displayed for 'Mukesh-50 Update cypress.config.js'. The commit has one contributor listed. The code editor shows the contents of the cypress.config.js file:

```
33 lines (29 sloc) | 696 Bytes
```

```
1 const { defineConfig } = require("cypress");
2
3 module.exports = defineConfig({
4
5   projectId: "n6k1eq",
6
7   reporter: 'cypress-mochawesome-reporter',
8   reporterOptions: {
9     charts: true,
10    reportpageTitle: 'Automation Test Report',
11    embeddedScreenshots: true,
12    inlineAssets: true,
13    saveAllAttempts: false,
14    autoOpen: false, // This line is highlighted in blue
15    overwrite: true,
16    code: false
17 },
18
19   e2e: {
```

Make auto open false in cypress [config.js](#) file.

Every test runs and opens report so subsequent test are not able to add to the reports.