# Learning Objectives

**Learners will be able to…**

- **Define unit tests**
- **Work with commit statuses**
- **Automate tests on different in different scenarios**

info

## Make Sure You Know

Learners should be familiar with `nodejs` and its basic use.

## Limitations

Assignment will require an active GitHub repository for activities.

# Introduction

**Unit tests** are designed to test each individual component, or **unit**, of code and confirm that it performs as expected. This ensures that all parts of the application function properly, before they are integrated into the whole system.

To explore unit testing, we'll use `jest` to run a few autotests. Using jest for unit testing is an effective way to automate the testing process and quickly write and run tests on our code.

Unit tests typically consist of one or more **assertions** that evaluate a value, such as verifying if a certain output is expected from a given input. The assert function determines whether an expression is true or false, and throws an error if it evaluates to false.

## Checkpoint

## Define Unit Tests

In order to explore using `jest`, we first need to make sure we install all dependencies needed for our app.

**Click the button below to install necessary dependencies for our project.**

Let’s look at an example of how we might use `jest` to define a simple unit test:

```
  test('adds 1 + 2 to equal 3', () => {

    expect(sum(1, 2)).toBe(3);

  });
```

Here we are writing a simple test to make sure that a function, `sum`, is performing as expected. We pass in the values 1 and 2 and then use jest’s `expect` command to say that we expect the result of the function `toBe` 3. If this assertion turns out to be true, the test passes.

Now, let’s take a look at some tests we’ve defined for our practice application in the `calc.test.js` file to the left.

```
const num1 = 16;
const num2 = 4;

test('Add numbers', () => {
  expect(calc(num1, num2, '+')).toBe(20);
});

test('Subtract numbers', () => {
  expect(calc(num1, num2, '-')).toBe(12);
});
...
```

**Here, we’ve defined:**

- Two tests, `Add numbers` and `Subtract numbers`.
- Two integer variables, `num1` and `num2` and an operator to pass into the

`calc` function.
- A result that we expect the function `toBe`.

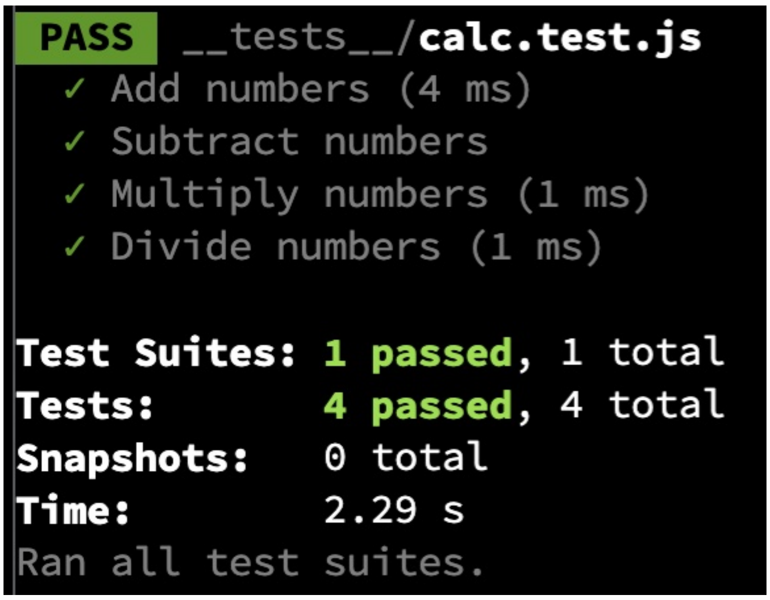
In other words, if we pass 16, 4, and + into our `calc` function, we expect the result `toBe` 20.

If this is not the case, we suspect that something is wrong with either our test or our function.

We can use the `jest` command in the terminal to run the tests in our `calc.test.js` file and test our application function, `calc`.

**Click the button below to run the `jest` command and trigger our unit tests.**

Jest will generate a report detailing the tests it ran and the status of each test.



```
 PASS  __tests__/calc.test.js
  ✓ Add numbers (4 ms)
  ✓ Subtract numbers
  ✓ Multiply numbers (1 ms)
  ✓ Divide numbers (1 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        2.29 s
Ran all test suites.
```

Terminal output for successful jest command

When all of our tests pass successfully, the `jest` command will return a 0 success code.

## Checkpoint

# Define Tests Task

In this lesson, we will take a closer look at how to define a test task inside of a `package.json` file using `jest`.

Make sure we're located inside our `ci-cd-sample-app` directory.

```
cd ci-cd-sample-app
```

We'll make sure we have all the dependencies necessary for our project.

**Let's add the following code snippet to the `scripts` section of our `package.json` file.**

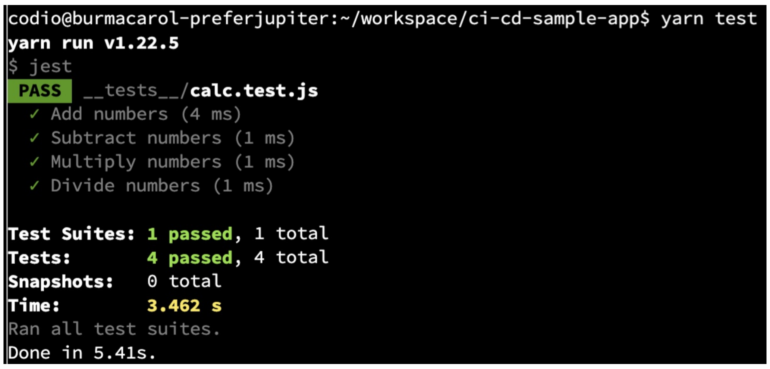
```
    "test": "jest"
```

This defines the command `yarn test`, which can then be used to run the `jest` command in our project.

**Click the button below to run `yarn test` in the terminal.**

When Jest is run, it searches for any tests that are declared inside of test files. Test files should use the following naming convention: `filename.test.js` (e.g., `user-interface.test.js`).

We should see that `jest` runs all of our defined tests



```
codio@burmacarol-preferjupiter:~/workspace/ci-cd-sample-app$ yarn test
yarn run v1.22.5
$ jest
 PASS  __tests__/calc.test.js
  ✓ Add numbers (4 ms)
  ✓ Subtract numbers (1 ms)
  ✓ Multiply numbers (1 ms)
  ✓ Divide numbers (1 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        3.462 s
Ran all test suites.
Done in 5.41s.
```

Terminal output for successful jest command

Using this framework, we can easily automate the testing stage of our CI/CD process.

To learn more about how to write individual tests with Jest, check out the official **documentation**.

## Checkpoint

# Add Workflow For Github Actions

With GitHub Actions we can define a unit test task workflow inside our workflow file to run these tests automatically each time code is pushed to the repository.

For example, we can define our Unit Tests task workflow by adding the following code to our `workflow` file:

```
name: Unit Tests


on:
  # the 1st condition
  workflow_run:                             # triggers when
        "build" runs and completes
    workflows: ["CI-CD"]
    types:
      - completed

jobs:                                       # create job called
        "test" to perform Unit Tests
  test:
    runs-on: ubuntu-latest.                 # this job will run
        on the latest version of Ubuntu
    steps:                                  # list of steps that
        need to be taken when Unit Tests is triggered
    - uses: actions/checkout@v3             # check out code
        from repository
    - name: Install dependencies            # install all
        dependencies needed for Unit Tests (ex. Jest)
      run: npx yarn
    - uses: actions/download-artifact@v3    # download compiled
        code from build and make it available for Unit Tests
      with:
        path: dist
    - name: Run Tests                       # run Unit Tests
        using Jest
      run: npx yarn test
```

## Let’s install the `get tests` action by running the command below in the terminal.

```
npx yarn
```

This adds all dependencies, including `jest`, to our project.

To perform this test, we need the compiled code from the `build` workflow to be available, so we need to change the `on` condition from `build` to `push`.

Once this is done, any changes pushed to the repository should automatically trigger tests.

In case of a failure, an email will be sent out with more details and instructions.

## Checkpoint

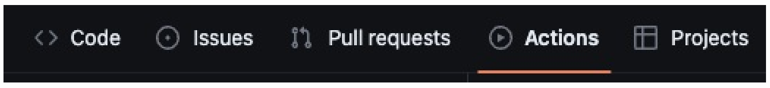
# Add Commit Badge

Adding a **commit status badge** to your GitHub project is an easy way to visualize the current state of commits in real-time. A **badge** indicates whether a workflow is passing or failing, so you don't have to manually check it each time.

This badge can be placed in your project's `README` file, making it easy for visitors and contributors to check the current status of your project.

### Let's add a badge for tests!

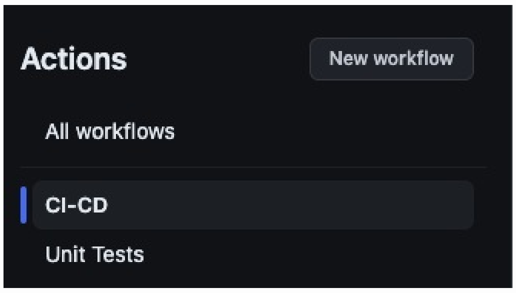
- **Navigate to your GitHub repository and Click on the `Actions` tab.**



GitHub Actions Tab

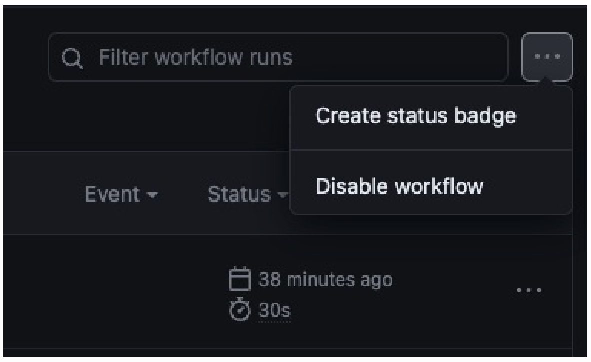
- This will open up your **workflows** folder.

- **In this folder, find the `.yml` file for the workflow you'd like to create a badge for.**



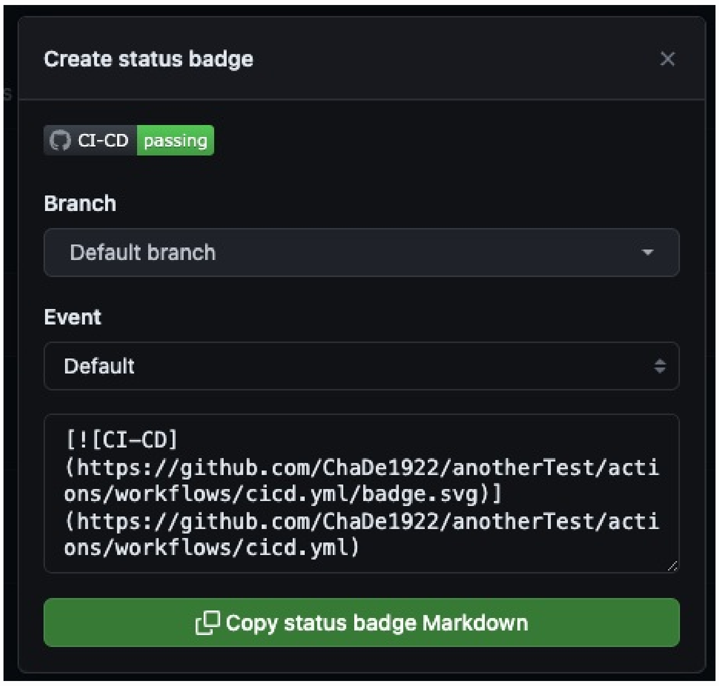
GitHub Workflows Folder with CI-CD file selected

- **In the upper-left corner, near the search bar, click on the triple-dot . . . symbol to find the `Create status badge` option.**



.guides/img/createBadge

- **Now, copy the badge URL from GitHub's badge generator. It should look something like this.**



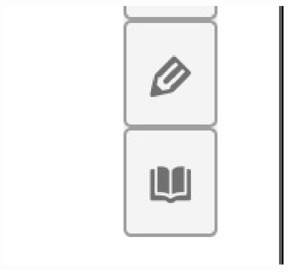
Branch: Default branch

Event: Default

```
[![CI-CD]
(https://github.com/ChaDe1922/anotherTest/acti
ons/workflows/cicd.yml/badge.svg)]
(https://github.com/ChaDe1922/anotherTest/acti
ons/workflows/cicd.yml)
```

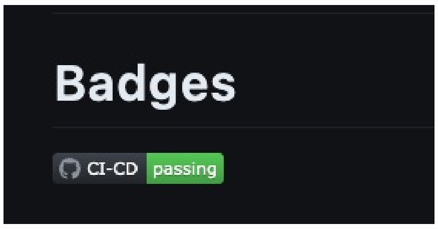
GitHub badge generator copy link

- **Paste the badge URL into your workflow editor, in place of where you'd like it to display in your `README` file by clicking the pencil in the file to the left.**



Pencil icon to edit README file

- **Save the changes and commit them to GitHub. You should now see a status badge on your repository page!**



Passing GitHub Actions status badge display

## Checkpoint