



Middlebury

**CSCI 201: Data Structures**

**Fall 2024**

---

**Lecture 12R: Software Development II (Testing)**

# Goals for today:

**Motivation:** as codes get big, how do we make sure all the components still work?



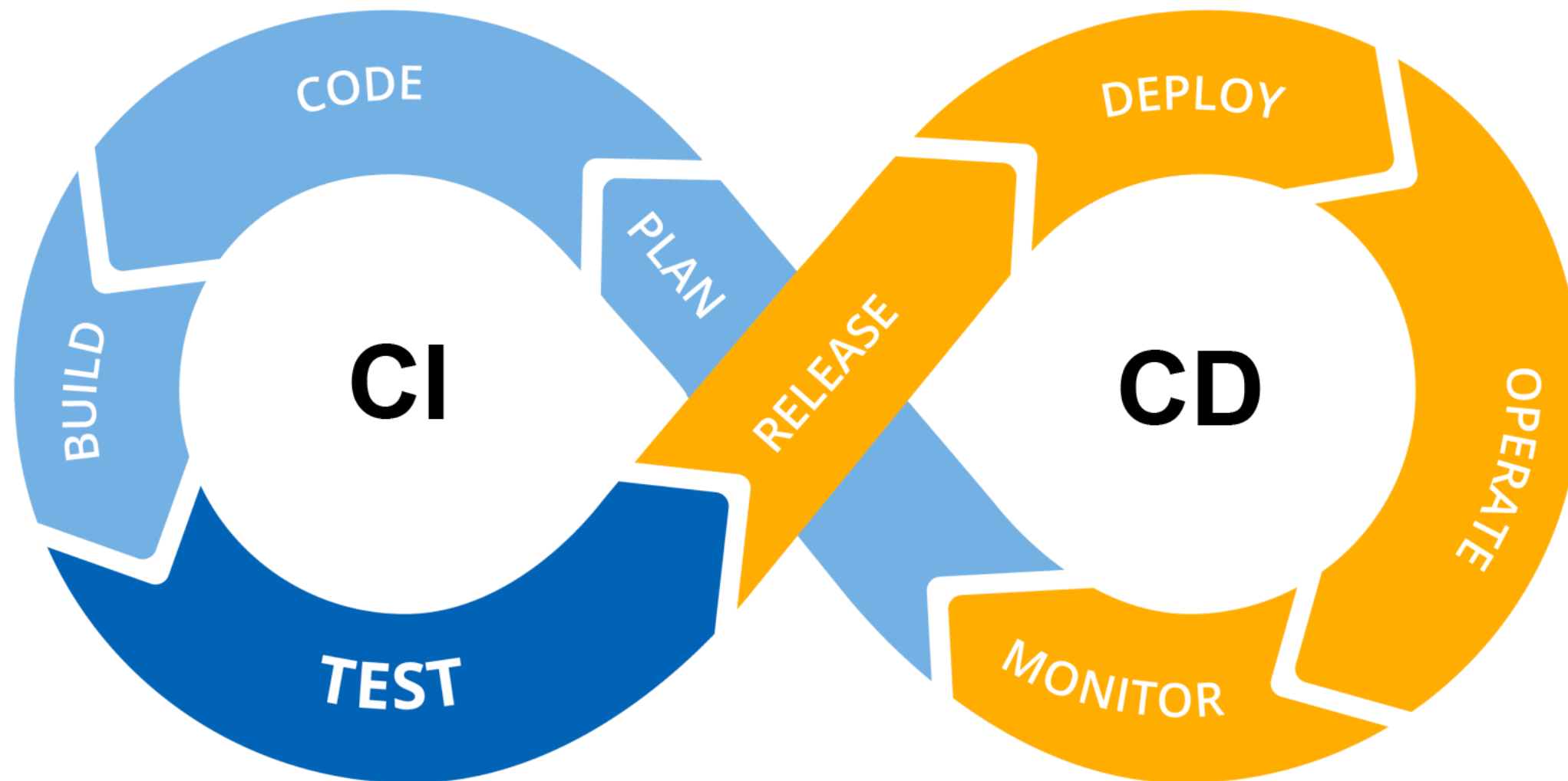
- Use **assertions** in **JUnit** to **test** the output of methods you write.
- **clone** a repository locally to your computer.
- Continue practicing with the **add** (stage), **commit** and **push** commands.
- Monitor the success of your tests using a **GitHub Action**.
- Publish a website with the results of your code using a **GitHub Action**.

# How do we test our codes right now?

```
1 public static void main(String[] args) {
2     Graph<Character> graph = new Graph<>();
3
4     graph.addEdge('e', 'g');
5     graph.addEdge('e', 'f');
6     graph.addEdge('c', 'g');
7     graph.addEdge('b', 'c');
8     graph.addEdge('a', 'd');
9     graph.addEdge('d', 'e');
10    graph.addEdge('c', 'd');
11    graph.addEdge('a', 'f');
12    graph.addEdge('b', 'd');
13    graph.addEdge('a', 'b');
14
15    System.out.println(graph.adj);
16
17    System.out.println("DFS:" + graph.dfs('b'));
18    System.out.println("BFS:" + graph.bfs('b'));
19 }
```

- Write a **PSVM** and call a few methods.
- Print and visually inspect the result.
- We can use assertions (**assert**), but these might cause the program to completely terminate.

It would be nice if we can make sure our product always has functioning code (that passes some tests).



**Continuously integrate** (CI) code that builds (compiles) and works (tests pass), which can then be **continuously deployed** (CD) into a customer-facing product.

# Getting the code for today: click on the **code** link in row for Lecture 12R (and accept the assignment).

csci201f24 / lecture23-philip-middlebury

Type / to search

<> Code

Issues

Pull requests 1

Actions

Projects

Wiki

Security

Insights

Settings

lecture23-philip-middlebury

Private

forked from [csci201f24/csci201f24-classroom-lecture23-lecture23-template](#)

Watch 0

Fork 0

Star 0

Your main branch isn't protected

Protect this branch

Protect this branch from force pushing or deletion, or require status checks before merging. [View documentation.](#)

main

2 Branches

0 Tags

Go to file

Add file

<> Code

This branch is 3 commits ahead of main .

#1

philip-middlebury

added some graph tests

b1d01a4 · 35 minutes ago

4 Commits

.devcontainer	Initial commit	42 minutes ago
.github	GitHub Classroom Feedback	42 minutes ago
.gitignore	Initial commit	42 minutes ago
Graph.java	Initial commit	42 minutes ago

About

csci201f24-classroom-lecture23-lecture23-template created by GitHub Classroom

Activity

Custom properties

0 stars

0 watching

0 forks

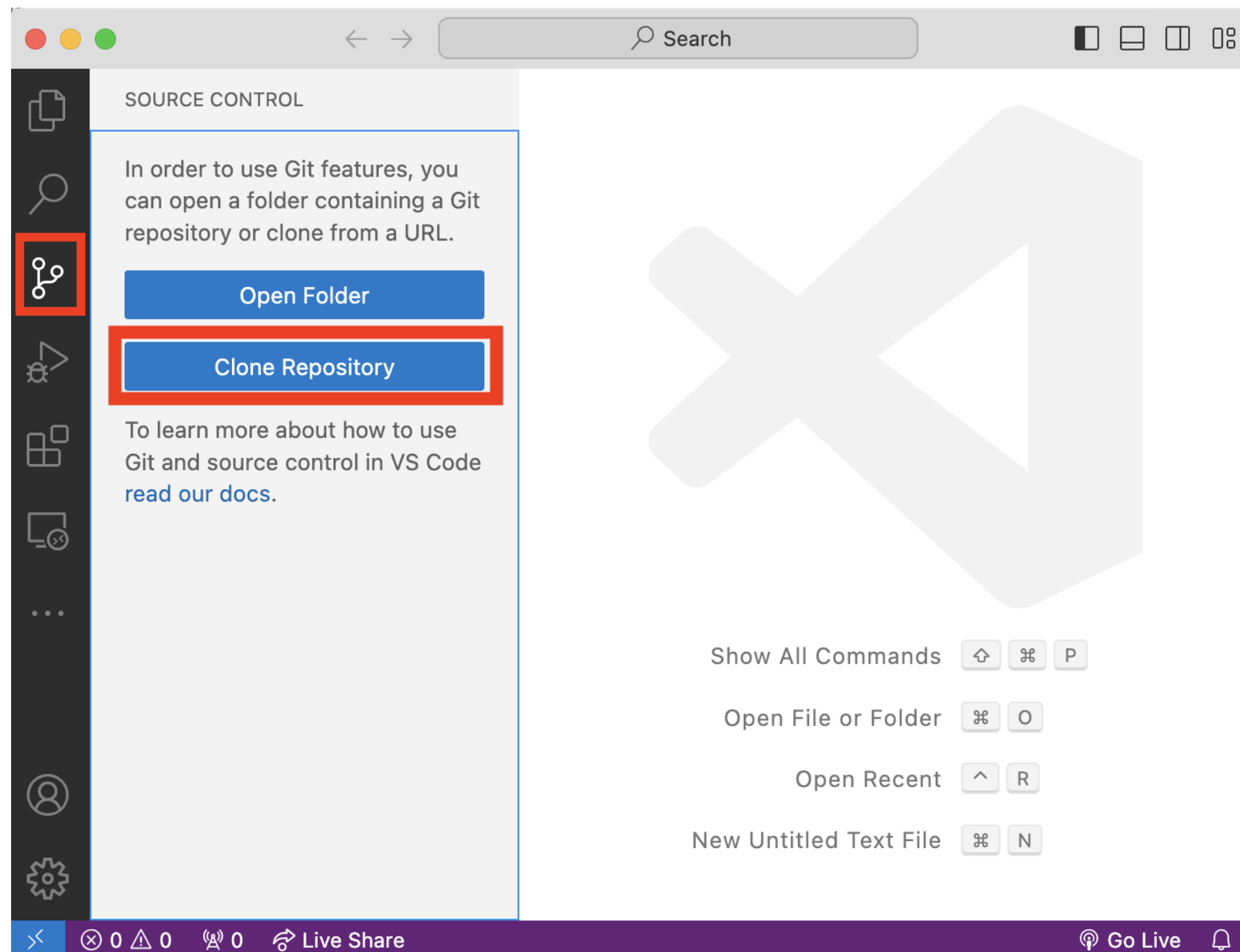
Releases

No releases published

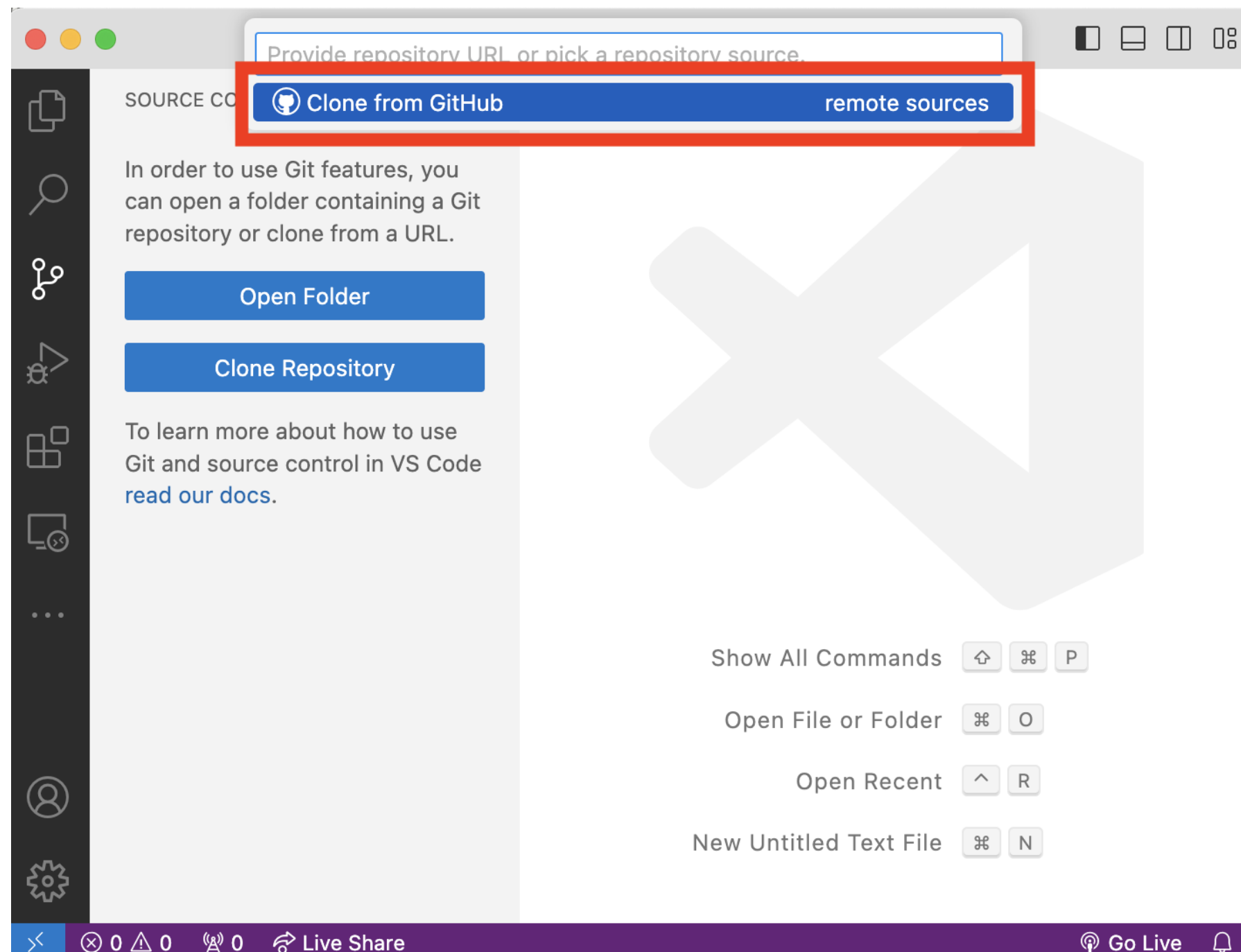
[Create a new release](#)

Packages

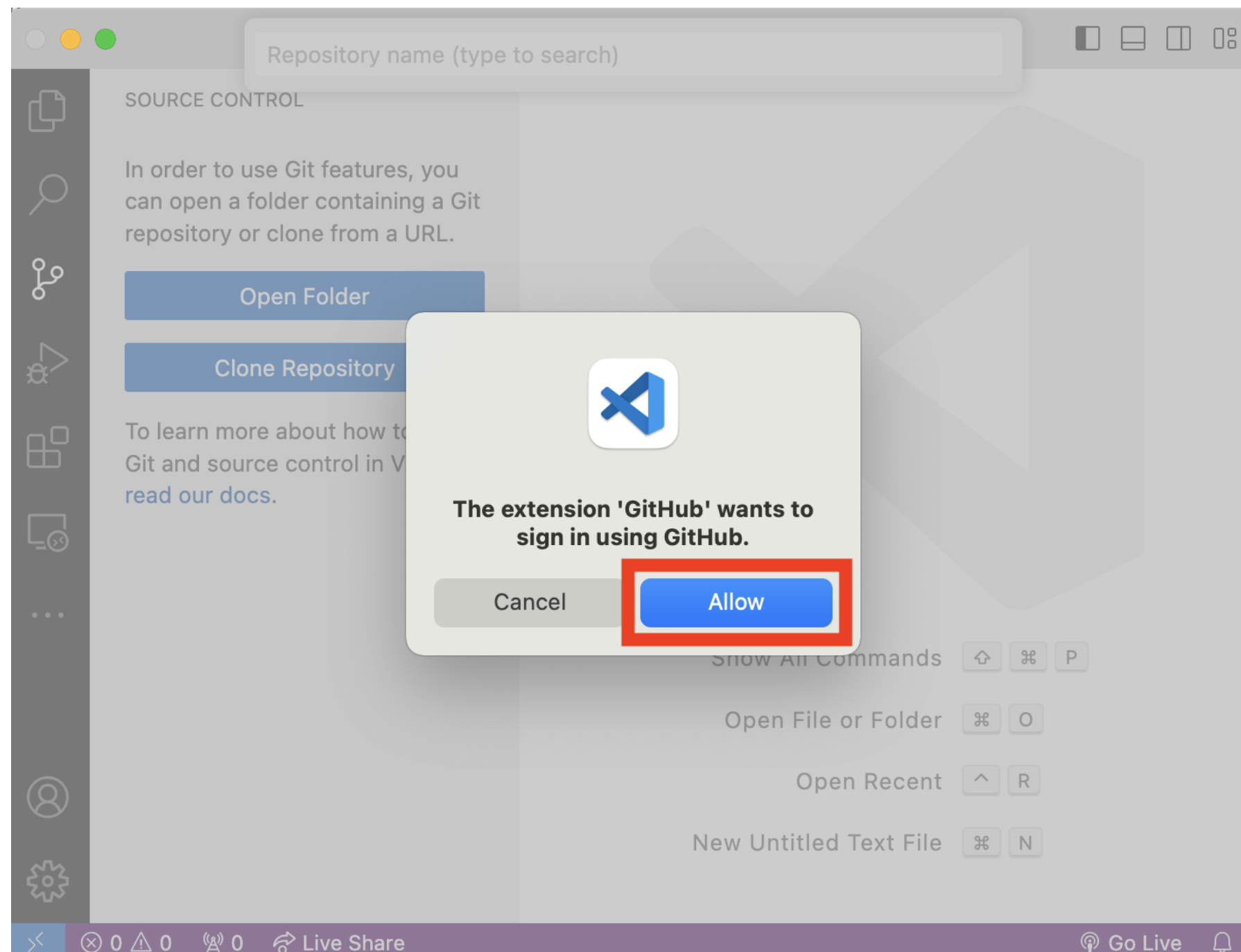
Open a new VS Code Window (**File -> New Window**). Then click on the **Source Control** icon and click **Clone Repository**.



Click on **Clone from GitHub**.




Then sign into **GitHub**.






# Continue signing into GitHub.



Select user to authorize  
**Visual Studio Code**

 Signed in as  
philipclaude

Continue

Use a different account

[Terms](#) [Privacy](#) [Docs](#) [Contact GitHub Support](#) [Manage cookies](#) [Do not share my personal information](#)

If your browser asks you to allow this application to open links, click "Open Link" to continue.

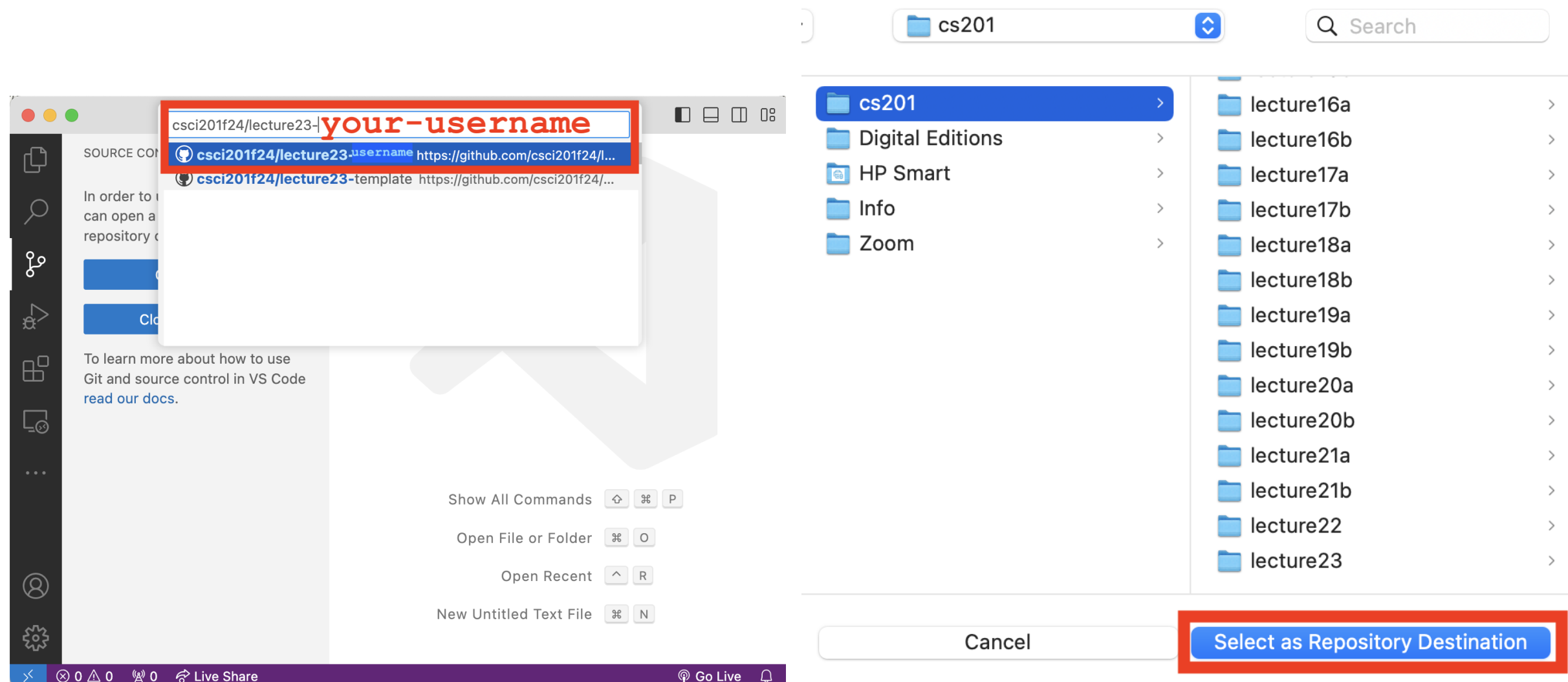
Allow `https://vscode.dev` to open the vscode link with Code?

[Choose a different application.](#)

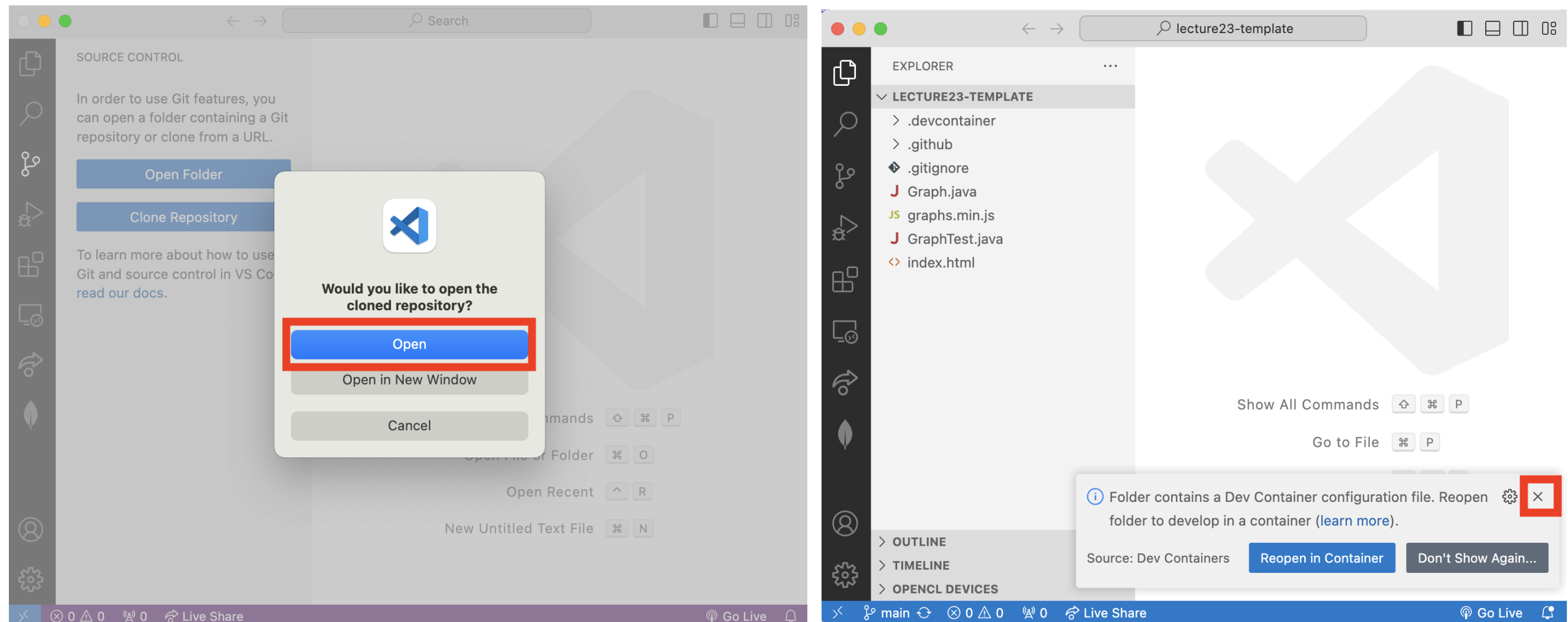
☐ Always allow `https://vscode.dev` to open vscode links

Cancel Open Link

Back in VS Code, start typing **csci201f24/lecture23-** and then add your **username**. Select where to save the repository.



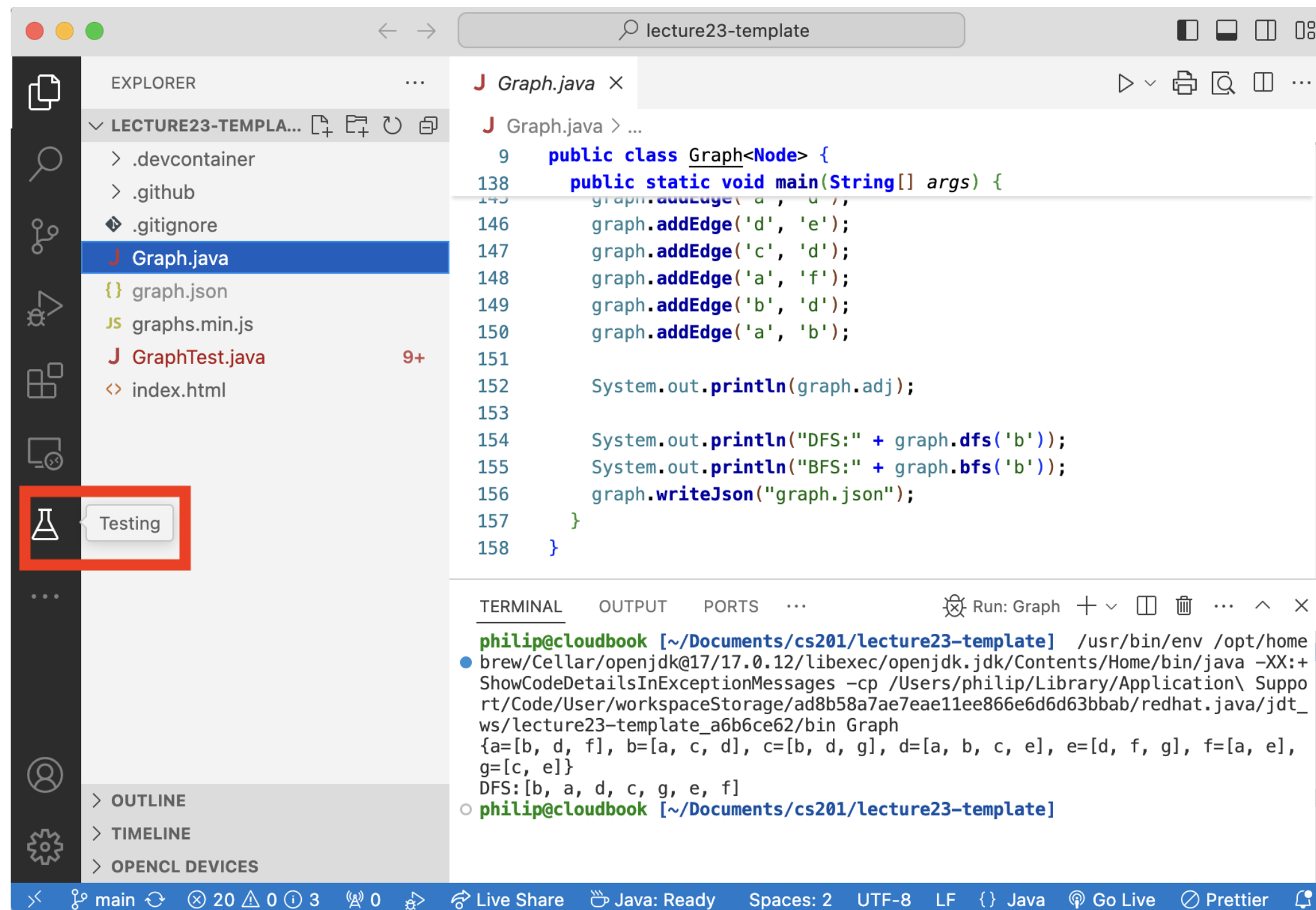
# Open the repository, and then you should see the code for today.



Ignore the **devcontainer** prompt (needed for a Codespace if you decide to work there instead).

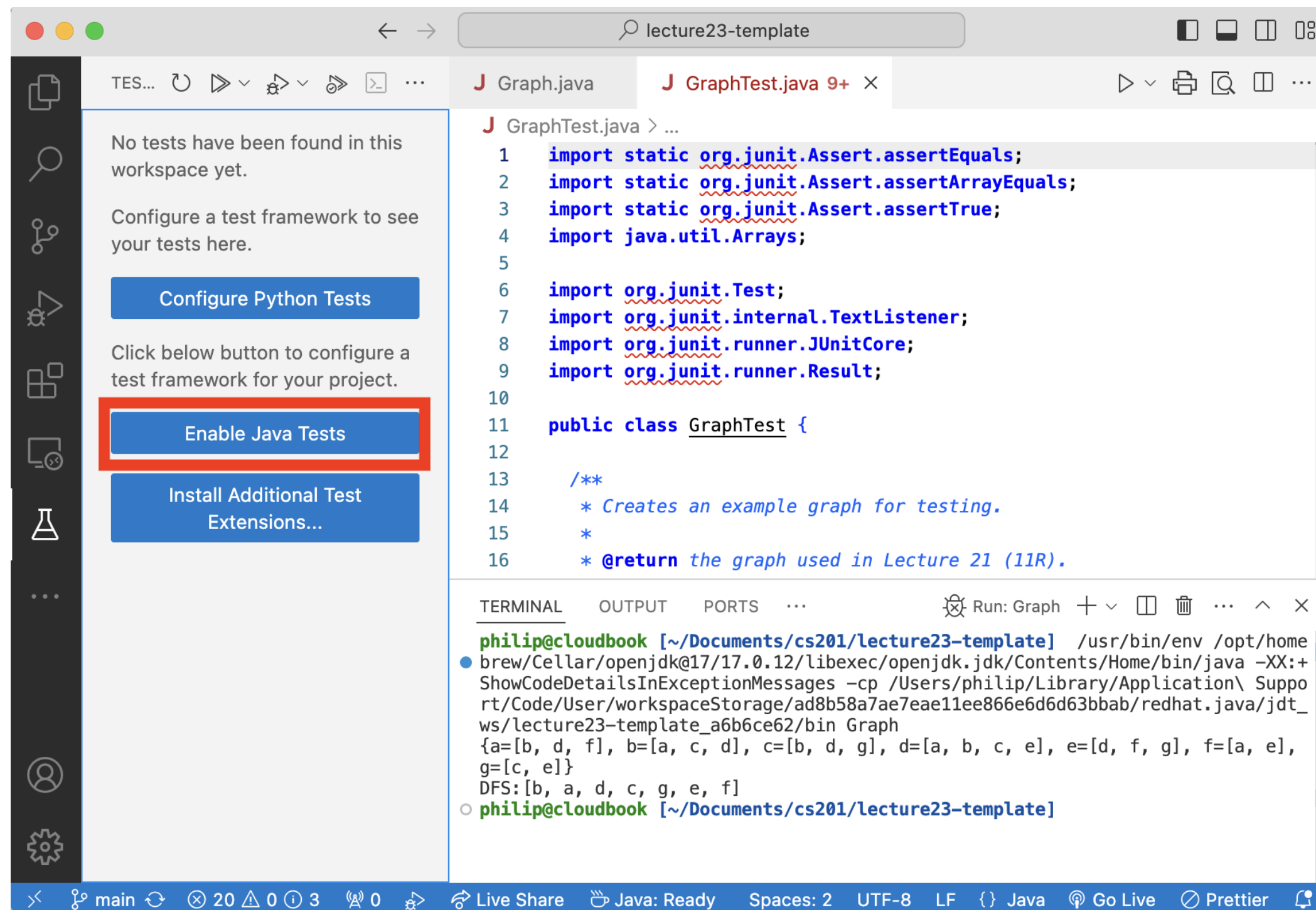
Now Compile and Run **Graph.java** (as usual). This is the same graph from Lecture 21 on BFS and DFS.

Click on the **Testing** Icon (which looks like a flask).



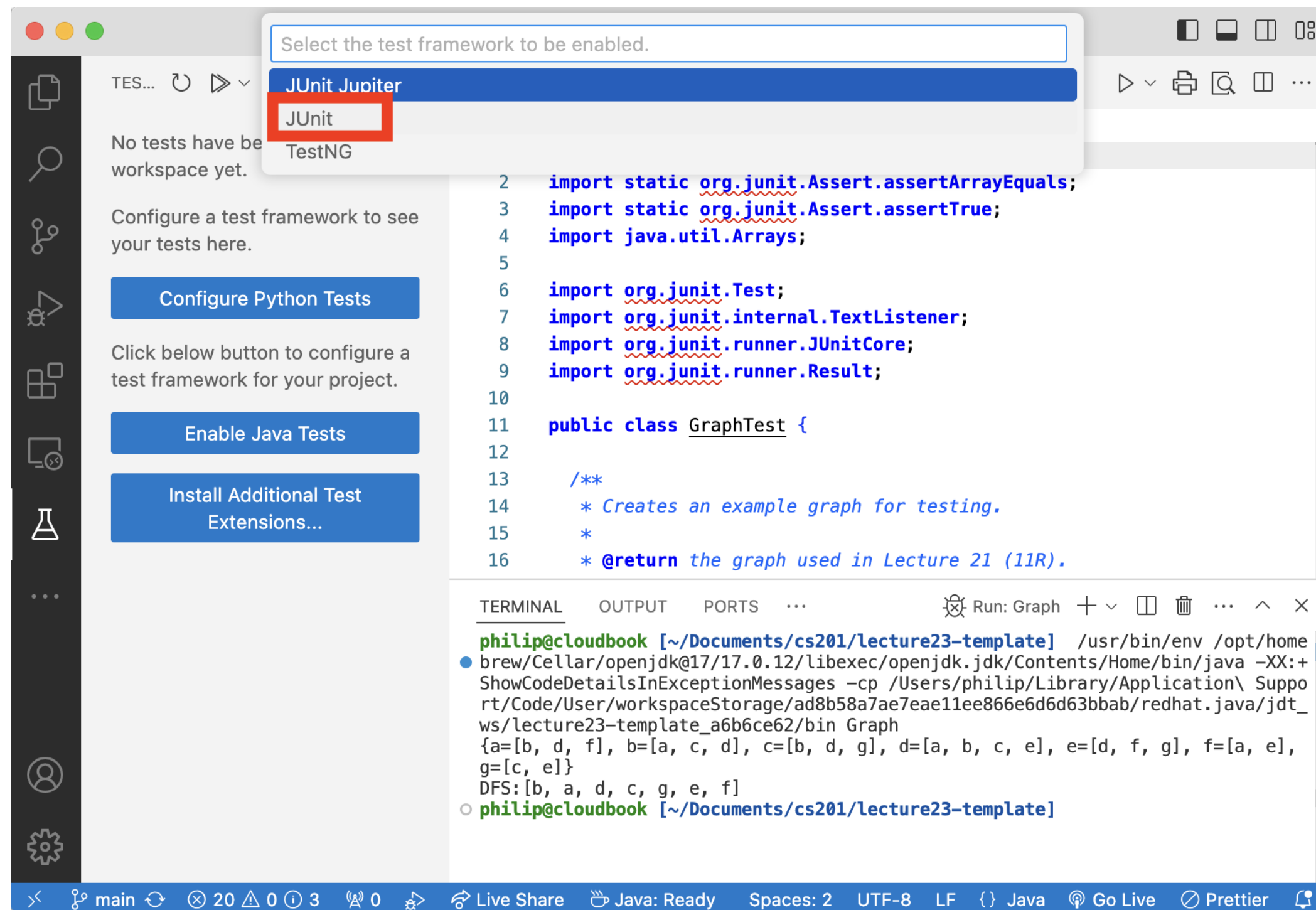
Notice that `GraphTest.java` has errors. We need to enable our testing framework.

Click on **Enable Java Tests**.



**GraphTest.java** still has errors. We need to enable our testing framework.

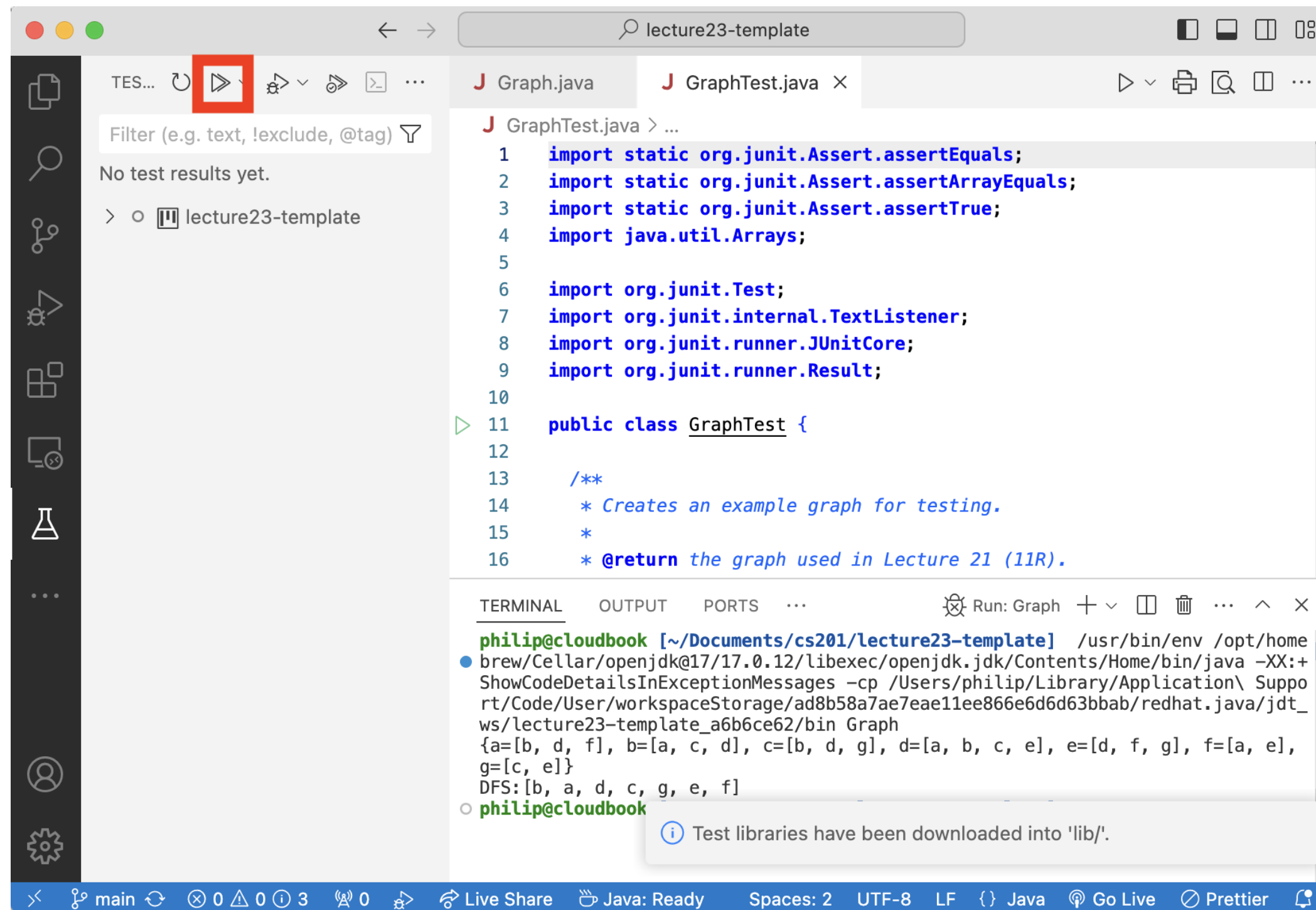
# Select JUnit.



The errors in `GraphTest.java` should go away.



Click on this button to run the tests.



Also, notice there is an additional directory with some **.jar** files (we need these to run the tests).

# JUnit is a testing framework for Java. We write testing methods and annotate them with `@Test`.

Within these methods, we will use assertions (e.g. `assertTrue`, `assertEquals`, `assertArrayEquals`) to test the output of our methods defined in some other file.

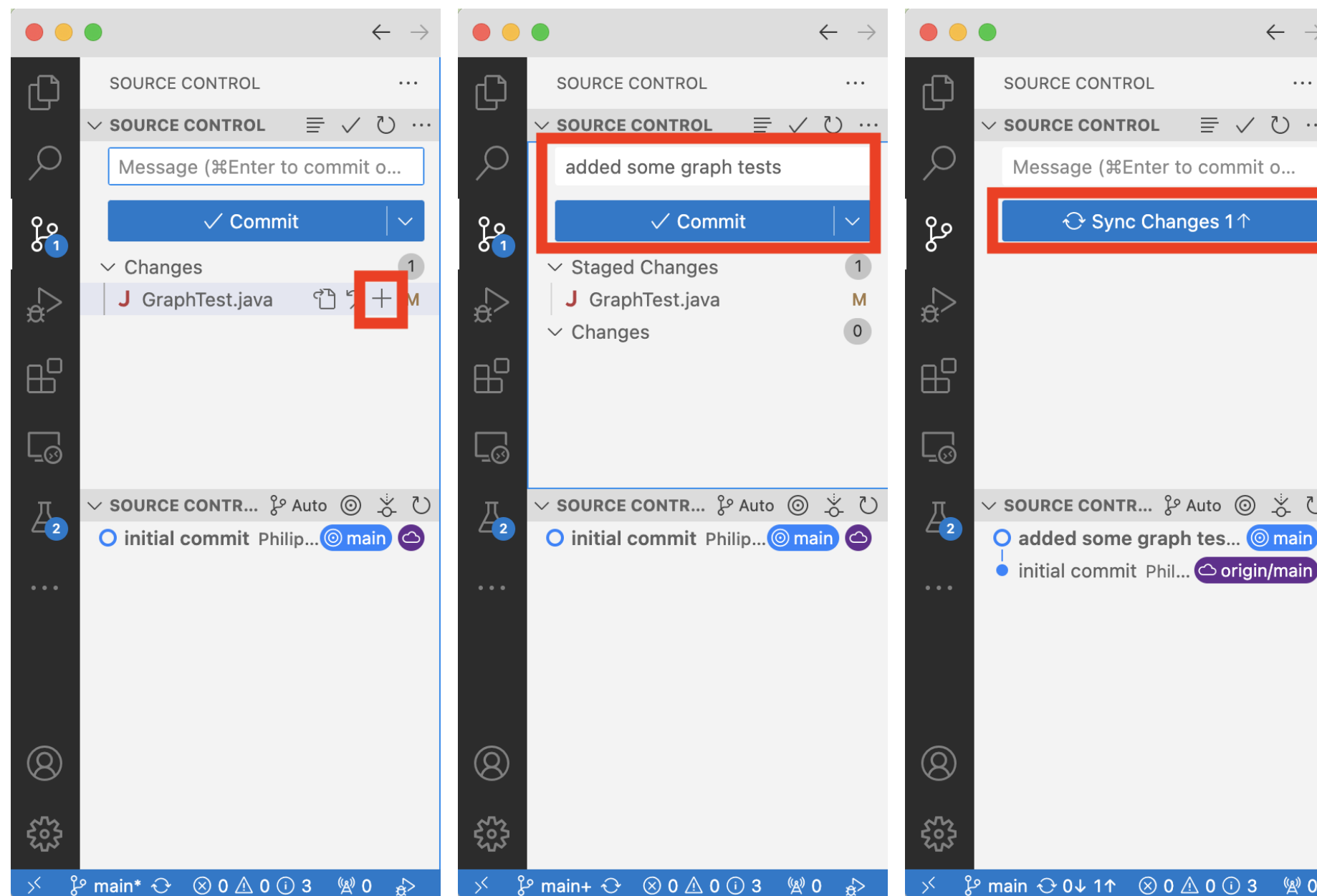
```
1  @Test
2  public void testNodes() {
3      Graph<Character> graph = makeExampleGraph();
4      assertEquals(graph.adj.size(), 7);
5
6      // retrieve the nodes and sort them to make sure the order is consistent
7      Character[] nodes = graph.adj.keySet().toArray(new Character[0]);
8      Arrays.sort(nodes);
9
10     Character[] expected = {'a', 'b', 'c', 'd', 'e', 'f', 'g'};
11     assertEquals(expected, nodes);
12 }
13
14 @Test
15 public void testGraphContainsEdges() {
16     Graph<Character> graph = makeExampleGraph();
17
18     // TODO check if the graph contains a few edges
19     // for example:
20     assertTrue(graph.hasEdge('d', 'a'));
21     // check a few more edges here
22 }
```



## Exercise: add some assertions in `testGraphDFS` and `testGraphBFS`.

*Hint:* use the output of running the `PSVM` in `Graph.java`.

Then **add** (click the **+** button for `GraphTest.java`), **commit** (adding a message), then sync (**push**).



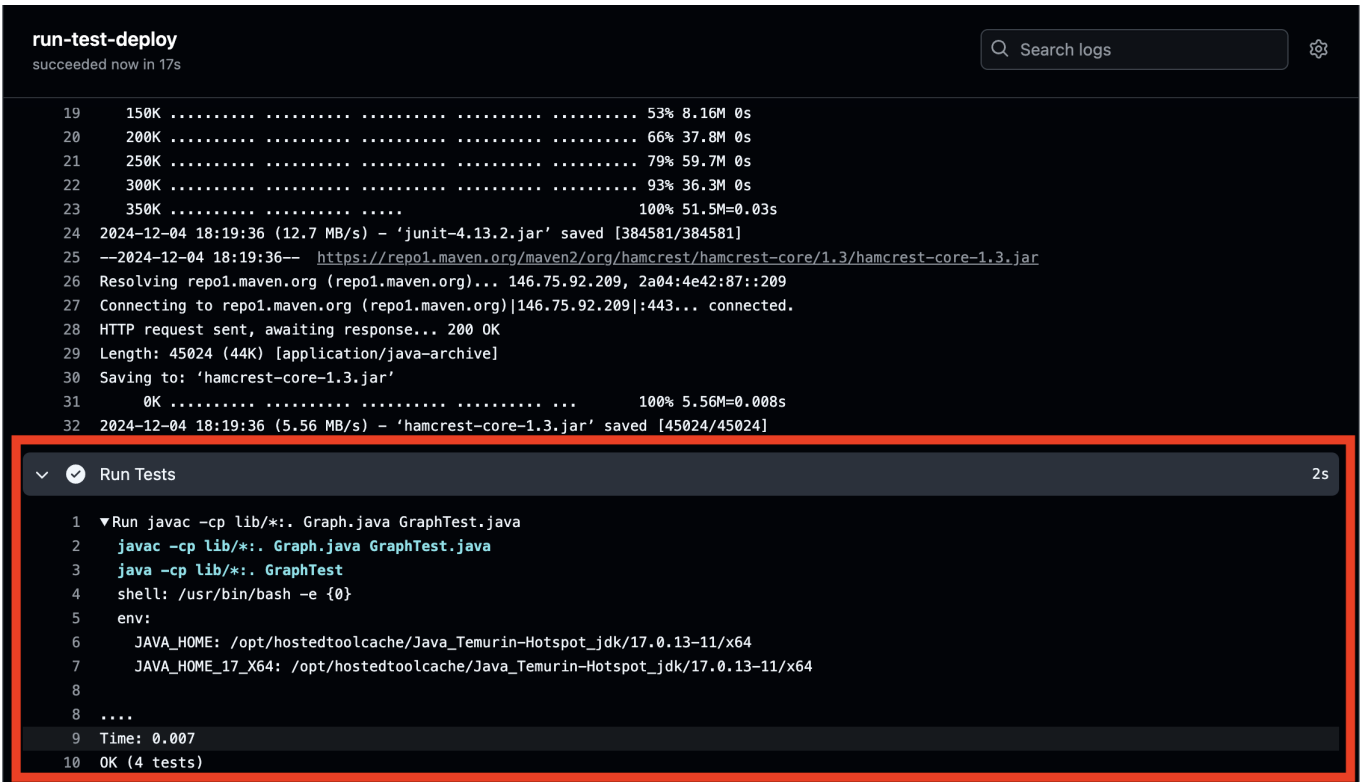
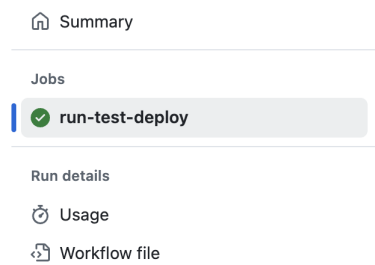
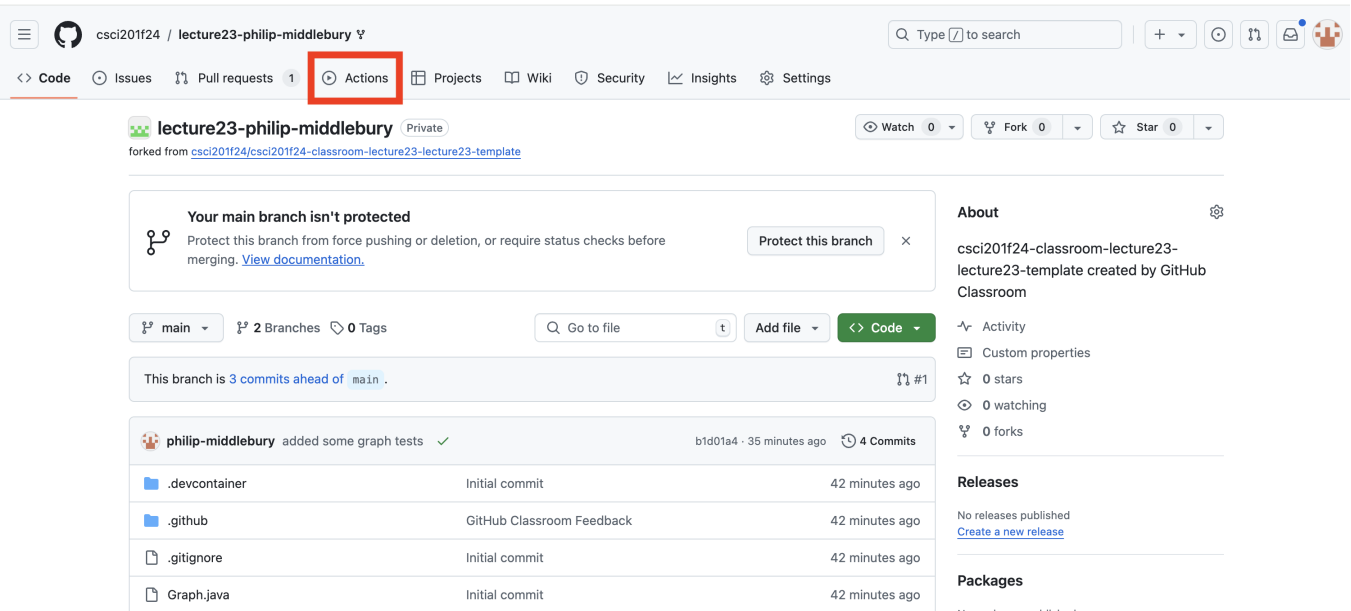
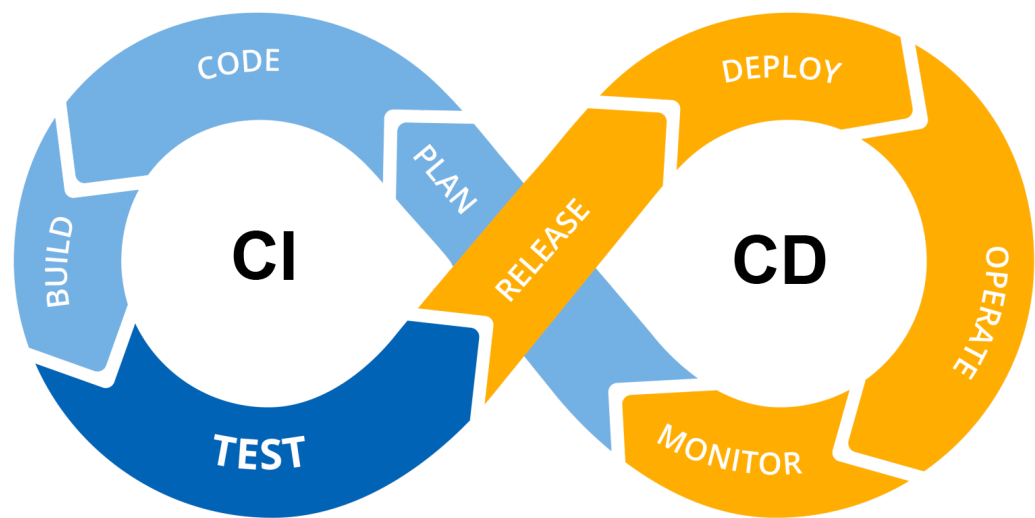
If there is an error about `git` credentials, enter the following in the **Terminal** (with your name/email):

```
git config --global user.email "youremail"  
git config --global user.name "yourname"
```

# Possible tests.

```
1  @Test
2  public void testGraphDFS() {
3      // run DFS starting at node 'b', saving the order
4      Graph<Character> graph = makeExampleGraph();
5      Character[] order = graph.dfs('b').toArray(new Character[0]);
6
7      Character[] expected = {'b', 'a', 'd', 'c', 'g', 'e', 'f'};
8      assertEquals(expected, order);
9  }
10
11 @Test
12 public void testGraphBFS() {
13     // run BFS starting at node 'b', saving the order
14     Graph<Character> graph = makeExampleGraph();
15     Character[] order = graph.bfs('b').toArray(new Character[0]);
16
17     Character[] expected = {'b', 'a', 'c', 'd', 'f', 'g', 'e'};
18     assertEquals(expected, order);
19 }
```

Let's revisit the idea of Continuous Integration and Continuous Deployment. Go back to your repository on **GitHub** and find the **Actions** tab.



On **GitHub**, this is controlled by **YAML** workflow files. See **`.github/workflows/run-test-deploy.yml`**

The screenshot shows the GitHub Actions interface for a workflow run named 'run-test-deploy'. On the left, a sidebar contains links for 'Summary', 'Jobs', 'Run details', 'Usage', and 'Workflow file'. The 'Jobs' section is active, showing a single job 'run-test-deploy' with a green checkmark. The main area displays the job's log, which includes progress bars for downloading 'junit-4.13.2.jar' and 'hamcrest-core-1.3.jar'. Below the log, a section titled 'Run Tests' is expanded, showing the execution of Java code: 'Run javac -cp lib/\*:. Graph.java GraphTest.java', 'javac -cp lib/\*:. Graph.java GraphTest.java', and 'java -cp lib/\*:. GraphTest'. The tests passed, resulting in 'OK (4 tests)'.

Other things happening here:

- Installing dependencies (build).
- Compiling and running the code (build).
- Running the unit tests (test).
- Notice that the **PSVM** in **Graph.java** produces a new file called **graph.json**.
- A few files are then copied to a "public" folder, which is then published (deploy).

We need to adjust a setting before being able to "deploy" our graph visualizer.

Click on these buttons in the following order.

The screenshot shows the GitHub repository settings page for 'csci201f24 / lecture23-philip-middlebury'. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings (marked with a red '1'). The left sidebar shows the 'Settings' tab selected, with the 'Pages' option highlighted (marked with a red '2'). The main content area displays the 'GitHub Pages' settings. The 'Build and deployment' section is expanded, showing the 'Source' dropdown set to 'Deploy from a branch' (marked with a red '3'). The 'GitHub Actions' option is highlighted with a red box, indicating it should be selected. The 'Visibility' section is also visible, showing options for restricting access to the repository.

Then re-run the workflow.

csci201f24 / lecture23-philip-middlebury

<> Code

Issues

Pull requests 1

Actions

Projects

Wiki

Security

Insights

Settings

Q Type to search

+ ▾

🕒

🔗

📧

🏠

Actions

New workflow

All workflows

run-test-deploy

Management

Caches

Deployments

Attestations

Runners

Usage metrics

Performance metrics

All workflows

Showing runs from all workflows

Filter workflow runs

3 workflow runs

Event ▾ Status ▾ Branch ▾ Actor ▾

✖ added some graph tests

run-test-deploy #3: Commit b1d01a4 pushed by philip-middlebury

main

📅 1 minute ago

🕒 25s

⋮

✖ Setting up GitHub Classroom Feedback

run-test-deploy #2: Commit 4d8ffd8 pushed by github-classroom bot

main

📅 7 minutes ago

🕒 19s

⋮

ⓘ GitHub Classroom Feedback

run-test-deploy #1: Commit 3877e26 pushed by github-classroom bot

main

📅 7 minutes ago

🕒 2s

⋮

csci201f24 / lecture23-philip-middlebury

<> Code

Issues

Pull requests 1

Actions

Projects

Wiki

Security

Insights

Settings

Q Type to search

+ ▾

🕒

🔗

📧

🏠

← run-test-deploy

✖ added some graph tests #3

Re-run jobs ⌵ ⋮

Summary

Jobs

run-test-deploy

Run details

Usage

Workflow file

Triggered via push 1 minute ago

philip-middlebury pushed → b1d01a4 main

Status

Failure

Total duration

25s

Artifacts

1

run-test-deploy.yml

on: push

✖ run-test-deploy

9s

🔍 - +

# After the workflow finishes, check out your own personal graph visualizer!

The top screenshot shows a GitHub Actions workflow run for the repository `csci201f24 / lecture23-philip-middlebury`. The workflow is `run-test-deploy` and it has successfully completed. The status is **Success**, the total duration is **39s**, and there is **1** artifact. The artifact is named `run-test-deploy.yml` and it was triggered by a push to the `main` branch. A red box highlights a link in the artifact's details: `https://csci201f24.github.io/lecture23-...`. An arrow points to this link with the text **click the link**.

The bottom screenshot shows a web browser at the URL `csci201f24.github.io/lecture23-philip-middlebury/`. The page title is **CSCI 201 Graph Visualizer**. There is a `30` spacing input field. The main content area displays a graph with 7 nodes labeled `a`, `b`, `c`, `d`, `e`, `f`, and `g`. The nodes are arranged in a circular pattern with edges connecting them in a complex structure.

You can change the graph in the **PSVM** in **Graph.java**.  
Any push will automatically update the graph that is used in the visualizer.

# Additional notes:

- **Homework 10** due Tuesday 12/10.
- Friday's lab (**Lab 10**) will have two components:
  - We will complete the **Course Response Form** in the lab period.
  - You will submit your reflection for your **Participation** grade.
- No class/lab on Monday (but I will be in my office during lab time for office hours).
- **Final Exam:** (released Tuesday 12/10 and will be due Friday 12/13)
  - **Two components:** Part 1 on Canvas and Part 2 (programming) submitted to Gradescope.
  - **Study Guide** will be posted by the end of the week.
- Please vote on the proposed changes to the Honor Code: <https://go.middlebury.edu/middpresence>.