

# CI/CD with GitHub Actions

Revision 1.2 – 11/09/24

Tech Skills Transformations LLC / Brent Laster

**Important Prerequisite:** You will need two separate GitHub accounts for this. (Free tier is fine.) To avoid confusion, we'll refer to your first one as your "primary" account and your second one as your "secondary" account. In the labs, the example primary account is "gwstudent" and the example secondary account is "gwstudent2".

## Lab 1 – Creating a simple example

**Purpose:** In this lab, we'll get a quick start learning about CI with GitHub Actions by creating a simple project that uses them. We'll also see what a first run of a workflow with actions looks like.

1. Log in to GitHub with your primary GitHub account.
2. Go to <https://github.com/skillrepos/greetings-ci> and fork that project into your own GitHub space. After this, you'll be on the project in your user space.

The screenshot shows the GitHub repository page for 'skillrepos/greetings-ci'. At the top, there's a search bar and navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the header, there's a 'Watch' button, a 'Fork' button (which is circled in red), and a 'Star' button. The main content area shows the repository's code structure with several commits listed. On the right side, there's an 'About' section describing it as a 'Simple starter repo for CI/CD with GitHub Actions', showing 0 stars, 1 watching, and 0 forks. There are also sections for 'Releases' (No releases published) and 'Packages'.

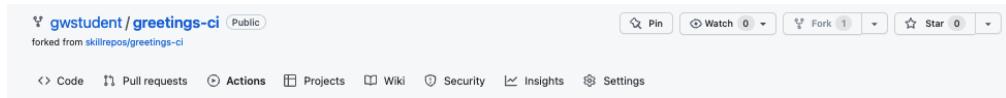
The screenshot shows the 'Create a new fork' dialog box. It starts with a heading 'Create a new fork' and a note explaining what a fork is. It has fields for 'Owner' (set to 'gwstudent') and 'Repository name' (set to 'greetings-ci'). Below these, there's a note about naming conventions and a 'Description (optional)' field containing 'Simple starter repo for CI/CD with GitHub Actions'. There are also checkboxes for 'Copy the main branch only' (which is checked) and a note about contributing back. At the bottom, there's a note about creating a fork in a personal account and a prominent green 'Create fork' button (which is circled in red).

3. We have a simple java source file named `echoMsg.java` in the subdirectory `src/main/java`, a Gradle build file in the root directory named `build.gradle`, and some other supporting files. We could clone this repository and build it manually via running Gradle locally. But let's set this to build with an automatic CI process specified via a text file. Click on the *Actions* button in the top menu under the repository name.

4. This will bring up a page with categories of starter actions that GitHub thinks might work based on the contents of the repository. We'll select a specific CI one. Scroll down to near the bottom of the page under "Browse all categories" and select "Continuous integration".

5. In the CI category page, let's search for one that will work with Gradle. Type "Gradle" in the search box and press Enter.

6. From the results, select the "Java with Gradle" one and click the "Configure" button to open a predefined workflow for this.



- This will bring up a page with a starter workflow for CI that we can edit as needed. There are two edits we want to make here. The first is to change the name. In the top section where the path is, notice that there is a text entry box around "gradle.yml". This is the current name of the workflow. Click in that box and edit the name to be "pipeline.yml". (You can just backspace over or delete the name and type the new name.)

TO

- The second edit is to remove the second job in this workflow since it currently has issues. To do this we will just highlight/select the code from line 50 on and hit delete. (If you have trouble just selecting that code, try starting at the bottom and selecting/highlighting from the bottom up.) The code to be deleted is highlighted in the next screenshot.

[Edit](#)[Preview](#)[Spaces](#)[2](#)[No wrap](#)

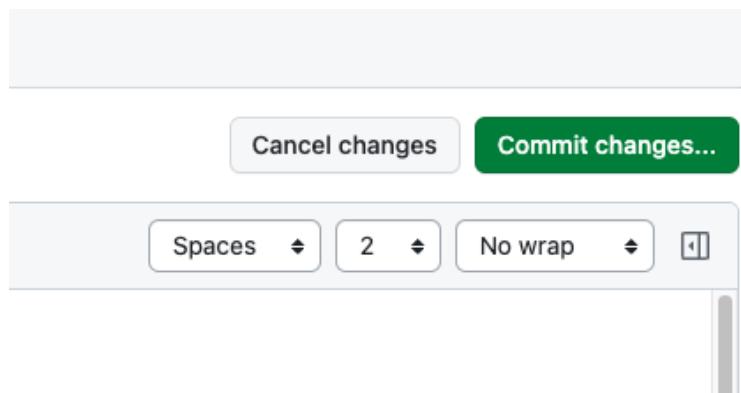
```

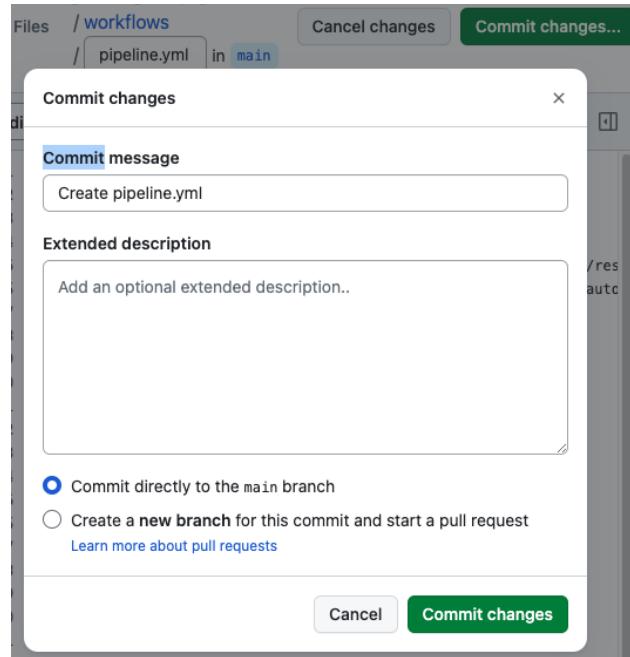
32     # See: https://github.com/gradle/actions/blob/main/setup-gradle/README.md
33   - name: Setup Gradle
34     uses: gradle/actions/setup-gradle@ec92e829475ac0c2315ea8f9eced72db85bb337a # v3.0.0
35
36   - name: Build with Gradle Wrapper
37     run: ./gradlew build
38
39   # NOTE: The Gradle Wrapper is the default and recommended way to run Gradle (https://docs.gradle.org/current/userguide/gradle_wrapper.html)
40   # If your project does not have the Gradle Wrapper configured, you can use the following configuration to run Gradle with a specified version
41   #
42   # - name: Setup Gradle
43   #   uses: gradle/actions/setup-gradle@ec92e829475ac0c2315ea8f9eced72db85bb337a # v3.0.0
44   #   with:
45   #     gradle-version: '8.5'
46   #
47   # - name: Build with Gradle 8.5
48   #   run: gradle build
49
50   dependency-submission:
51
52     runs-on: ubuntu-latest
53     permissions:
54       contents: write
55
56     steps:
57       - uses: actions/checkout@v4
58       - name: Set up JDK 17
59         uses: actions/setup-java@v4
60         with:
61           java-version: '17'
62           distribution: 'temurin'
63
64   # Generates and submits a dependency graph, enabling Dependabot Alerts for all project dependencies.
65   # See: https://github.com/gradle/actions/blob/main/dependency-submission/README.md
66   - name: Generate and submit dependency graph
67     uses: gradle/actions/dependency-submission@ec92e829475ac0c2315ea8f9eced72db85bb337a # v3.0.0
68

```

highlight/select and delete

- Now, we can go ahead and commit the new workflow via the “Commit changes...” button in the upper right. In the dialog that comes up, you can enter an optional comment if you want. Leave the “Commit directly...” selection checked and then click on the “Commit changes” button.





10. Since we've committed a new file and this workflow is now in place, the "on: push:" event is triggered and the CI automation kicks in. Click on the Actions menu again to see the automated processing happening.

11. After a few moments, the workflow should succeed. (You may need to refresh your browser.) After it is done, you can click on the commit message for the run to get to the details for that particular run.

12. From here, you can click on the build job in the graph or the “build” item in the list of jobs to get more details on what occurred on the runner system. You can expand any of the steps in the list to see more details.

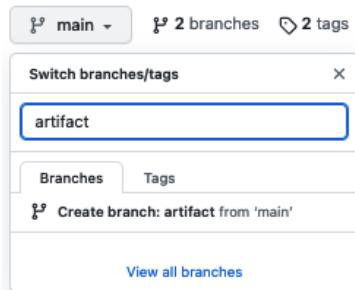
The screenshot shows the GitHub Actions pipeline details for a workflow named 'Create pipeline.yml #1'. The 'Actions' tab is selected. On the left, there's a sidebar with 'Summary', 'Jobs' (which has a 'build' item circled in red), 'Run details', 'Usage', and 'Workflow file'. The main area shows a tree view of the build steps. The 'Set up job' step is collapsed. The 'Run actions/checkout@v3' step is expanded, showing its sub-steps: 'Run actions/checkout@v3', 'Syncing repository: brentlaster/greetings-ci', 'Getting Git version info', and 'Temporarily overriding HOME=/home/runner/work/\_temp/77610fae-ac20-4aea-8803530bce6e before making global git config changes'. A 'Re-run all jobs' button is at the top right.

END OF LAB

## Lab 2 – Managing Artifacts

**Purpose:** In this lab, we'll look at how to do simple artifact management – an important part of Continuous Delivery.

- As a best practice for building out the pipeline as a larger project, let's create a separate branch to work in for managing the versioning and storage of the artifact. We'll call it “artifact”. In the “Code” tab, click on the branch dropdown that says “main”. Then in the text area that says “Find or create a branch...”, enter the text “artifact”. Then click on the “Create branch: artifact from ‘main’” link.



- Now you should be on the *artifact* branch. We're going to first add the code to persist the artifact that we built in our build step. We want to persist this for use with other jobs in our pipeline such as ones that might test it. Open the .github/workflows/pipeline.yaml file (click on the name) and edit it by clicking on the pencil icon.

A screenshot of the GitHub code editor interface. The left sidebar shows a tree view with 'artifact' selected under '.github/workflows'. The main area displays the contents of 'pipeline.yml'. A red circle highlights the 'Edit' icon in the toolbar above the code area.

```
greetings-ci/.github/workflows/pipeline.yml
```

```
gwstudent2 fix: update branch name a60e5af · last week History
```

```
38 lines (30 loc) · 1.03 KB
```

```
Code Blame Raw ⌂ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂
```

```
1 # This workflow uses actions that are not certified by GitHub.
2 # They are provided by a third-party and are governed by
3 # separate terms of service, privacy policy, and support
4 # documentation.
5 # This workflow will build a Java project with Gradle and cache/restore any dependencies to improve the workflow execution time
6 # For more information see: https://help.github.com/actions/language-and-framework-guides/building-and-testing-java-with-gradle
7
8 name: Java CI with Gradle
9
10 on:
11   push:
12     branches: [ "artifact" ]
13   pull_request:
14     branches: [ "artifact" ]
```

3. Change the references in the “on:” clause to be just the “artifact” branch so we don’t trigger action runs on the other branches while we are working on this one. Make sure you are on the *artifact* branch before you proceed.

A screenshot of the GitHub code editor interface, showing the same 'pipeline.yml' file. A large red oval highlights the 'push' and 'pull\_request' sections of the 'on:' clause. The 'push' section now contains the single branch 'artifact', and the 'pull\_request' section also contains it.

```
greetings-ci/.github/workflows/pipeline.yml in artifact
```

```
Edit file Preview changes Spaces 2 No wrap
```

```
1 # This workflow uses actions that are not certified by GitHub.
2 # They are provided by a third-party and are governed by
3 # separate terms of service, privacy policy, and support
4 # documentation.
5 # This workflow will build a Java project with Gradle and cache/restore any dependencies to improve the workflow execution time
6 # For more information see: https://help.github.com/actions/language-and-framework-guides/building-and-testing-java-with-gradle
7
8 name: Java CI with Gradle
9
10 on:
11   push:
12     branches: [ "artifact" ]
13   pull_request:
14     branches: [ "artifact" ]
```

4. Click on the square to the right to show the pane to search for actions (if search pane isn’t shown).

A screenshot of the GitHub code editor interface, showing the 'pipeline.yml' file. A red circle highlights the square icon in the top right corner of the code editor, which opens the Marketplace pane. The Marketplace pane is visible on the right side of the screen, displaying the 'Cache' action.

```
greetings-ci/.github/workflows/pipeline.yml in artifact
```

```
Cancel changes Commit changes...
```

```
Edit Preview Spaces 2 No wrap
```

```
Marketplace Documentation
```

```
Search Marketplace for Actions
```

```
Featured Actions
```

```
Cache By actions Cache artifacts like dependencies and build outputs to improve ⭐ 3.7k
```

5. To the right, you should see a pane with references to GitHub actions. We're going to add a job to our workflow to upload an artifact. Let's find actions related to uploading.

In the "Search Marketplace for Actions" box on the upper right, enter "Upload build" and see what's returned.

Find the one that is named "Upload a Build Artifact By actions" and click on it. Take a look at the page that comes up from that. Let's look at the full listing on the Actions Marketplace. Click on the "View full Marketplace listing".

The screenshot shows two side-by-side views of the GitHub Marketplace. On the left, a search result for 'Upload build' is shown with 10 results. Two specific actions are circled with red ovals: 'Upload a Build Artifact' by actions (2.7k stars) and 'Upload build to Autify for Mobile' by autifyhq (5 stars). On the right, the detailed view for 'Upload a Build Artifact' is displayed. It shows the action card with a play icon, the name 'Upload a Build Artifact', the creator 'By actions', the version 'v2.2.4', and the star count '1.1k'. Below the card is a description: 'Upload a build artifact that can be used by subsequent workflow steps'. A red oval highlights the link 'View full Marketplace listing'. Further down, there's an 'Installation' section with a code snippet for .yml files:

```

Version: v2.2.4 +
- name: Upload a Build Artifact
  uses: actions/upload-artifact@v2.2.4
  with:
    # Artifact name
    name: # optional, default is artifact
    # A file, directory or wildcard pattern that describes the artifact
    path:
      # The desired behavior if no files are found using the path
      warn: Output a warning but do not fail the action
      error: Fail the action with an error message
      ignore: Do not output any warnings or errors, the artifact will not be uploaded
  
```

6. This should open the full GitHub Actions Marketplace listing for this action. Notice the URL at the top - <https://github.com/marketplace/actions/upload-a-build-artifact>.

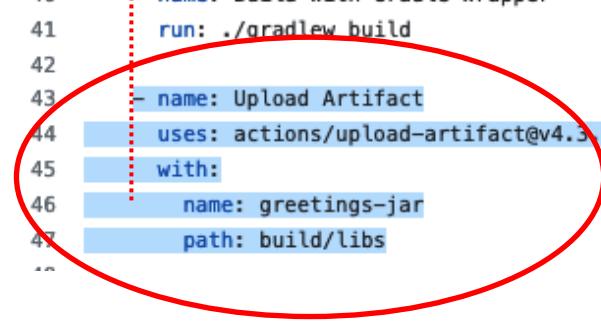
The screenshot shows the GitHub Marketplace listing for the 'Upload a Build Artifact' action. At the top, it displays the GitHub logo, the URL 'github.com/marketplace/actions/upload-a-build-artifact', and various navigation links like 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The main card for the action is shown with a blue play button icon, the name 'GitHub Action Upload a Build Artifact', the version 'v3.1.0 (Latest version)', and a 'Use latest version' button. To the right of the card, there are sections for 'Verified creator', 'Stars' (1.7k), 'Contributors' (a list of 10 contributors), 'Categories' (Utilities), and 'Links' (actions/upload-artifact). Below the card, there's a 'What's new' section with a list of changes, such as easier upload, multi-path upload, and support for excluding certain files.

7. Switch back to the browser tab where you are editing the workflow for greetings-actions. Update the build job to include a new step to use the "upload-artifact" action to upload the jar the build job creates. To do this, add the following lines inline with the build job steps. **Pay attention to the indenting.** See the screenshot (lines 40-44) for how this should look afterwards. (Your line numbers may be different.)

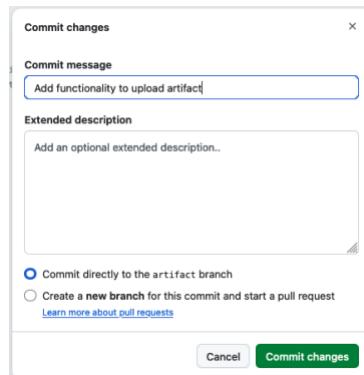
The code to add is immediately below. You can copy and paste but may need to adjust the indenting. Notice this should go after the *Build with Gradle Wrapper* step.

```
- name: Upload Artifact
  uses: actions/upload-artifact@v4.3.0
  with:
    name: greetings-jar
    path: build/libs
```

```
35   # Configure Gradle for optimal use in GitHub Actions, including caching
36   # See: https://github.com/gradle/actions/blob/main/setup-gradle/README
37   - name: Setup Gradle
38     uses: gradle/actions/setup-gradle@v3
39
40   - name: Build with Gradle Wrapper
41     run: ./gradlew build
42
43   - name: Upload Artifact
44     uses: actions/upload-artifact@v4.3.0
45     with:
46       name: greetings-jar
47       path: build/libs
```



8. Click on the green "Commit changes..." button in the upper right. In the dialog that comes up, add a commit message like "Add functionality to upload artifact", then click the green "Commit changes" button to make the commit.



9. Switch to the "Actions" tab in your repository to see the workflow run. After a few moments, you should see that the run was successful. Click on the title of that run "Add functionality to upload artifact". On the next screen, in addition to the graph, there will be a new section called "Artifacts" around the middle of the page. (You may have to scroll past several warnings that you can ignore.) You can download the artifact from there. Click on the name of the artifact to try this.

The screenshot shows the GitHub Actions interface. The top navigation bar includes Code, Pull requests, Actions (which is highlighted with a red border), Projects, Wiki, Security, Insights, and Settings. The main area is titled "All workflows" and shows "Showing runs from all workflows". There are two workflow runs listed:

- Add functionality to upload artifact**: Triggered via push 3 minutes ago by gwstudent2. Status: Success. Total duration: 32s. Artifacts: 1. This run is circled in red.
- Create pipeline.yml**: Triggered via push 8 hours ago by gwstudent2. Status: main. Total duration: 53s.

Below the workflow runs, the "Add functionality to upload artifact" run is expanded. It shows a "Summary" card with details: Triggered via push 3 minutes ago by gwstudent2 pushed d9f7ea8, Status: Success, Total duration: 32s, Artifacts: 1. The "pipeline.yml" step is shown with a green checkmark and a duration of 20s. A "build summary" table provides details about the build environment:

Gradle Root Project	Requested Tasks	Gradle Version	Build Outcome	Build Scan®
greetings-ci	build	4.10	✓	Not published

A note indicates that Caching for Gradle actions was read-only - expand for details. The "Artifacts" section shows a single artifact named "greetings-jar" with a size of 1010 Bytes. This artifact is also circled in red.

✓ Add functionality to upload artifact #2

[Re-run all jobs](#) [...](#)

The screenshot shows the GitHub Actions pipeline summary for a push event. It includes a summary card with basic stats (triggered via push 3 minutes ago, gwstudent2 pushed artifact, Status Success, Total duration 32s, Artifacts 1), a detailed view of the pipeline.yml file showing a single build step, and a build summary table for the greetings-ci project. The build summary table has columns for Gradle Root Project, Requested Tasks, Gradle Version, Build Outcome, and Build Scan®. The row for greetings-ci shows build as the task, 4.10 as the version, success as the outcome, and Not published as the scan status.

Gradle Root Project	Requested Tasks	Gradle Version	Build Outcome	Build Scan®
greetings-ci	build	4.10	✓	Not published

► Caching for Gradle actions was read-only - expand for details

Job summary generated at run-time

**Artifacts**  
Produced during runtime

Name	Size
greetings-jar	1010 Bytes

END OF LAB

### Lab 3 – Merging changes with a pull request

Purpose: In this lab, we'll see how to merge changes with a GitHub pull request

1. Now, let's open up a new pull request for the actual merging. To keep things simple, and avoid targeting the wrong repository, copy the link below, insert your GitHub userid in place of <github-userid> and then paste and go to the link. You can scroll to the bottom to see the actual file changes.

<https://github.com/<github-userid>/greetings-ci/compare/main...artifact?expand=1>

The screenshot shows the GitHub interface for creating a pull request. At the top, there's a header with navigation icons and the URL <https://github.com/gwstudent2/greetings-ci/compare/main...artifact?expand=1>. Below the header, the repository name 'gwstudent2 / greetings-ci' is displayed. A search bar with the placeholder 'Type to se...' is on the right.

The main area is titled 'Open a pull request'. It includes a note: 'Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about comparing branches'.

Below this, there are dropdown menus for 'base: main' and 'compare: artifact'. A green checkmark indicates 'Able to merge. These branches can be automatically merged.'

Form fields include 'Add a title' (with placeholder 'Add functionality to upload artifact') and 'Add a description' (with placeholder 'Add your description here...'). A rich text editor toolbar is above the description input.

At the bottom, there are buttons for 'Markdown is supported' and 'Paste, drop, or click to add files', followed by a large green 'Create pull request' button.

2. When done, click on the *Create pull request* button to actually create the pull request. On the next page, you'll see the pull request with all of the tabs for it. The main one is the *Conversation* tab. Notice the *All checks have passed* section in the middle. This where our GitHub Actions workflow ran because of the *on: pull\_request* trigger. Click on the *Show all checks* link to see the workflow run. Then click on the *Details* link to go to the run of the workflow. You can expand the steps of the workflow and look around if you want.

## Add functionality to upload artifact #3

The screenshot shows a GitHub pull request page with the 'Conversation' tab selected. A comment from 'gwstudent2' is visible, stating 'Closes #2'. Below the comment is a commit message: 'Add functionality to upload artifact'.

In the middle of the page, there's a section titled 'All checks have passed'. It contains a link 'Show all checks' which is circled in red.

The screenshot shows the GitHub Checks interface for a pull request. It displays a summary of checks: "All checks have passed" (1 successful check) and "This branch has no conflicts with the base branch". A specific check entry for "Java CI with Gradle / build (push)" is shown as "Successful in 20s". A red oval highlights the "Details" link next to this entry.

The screenshot shows the GitHub Actions pipeline details for a pull request. The "build" job is listed under "Jobs" and is marked as "Succeeded 2 hours ago in 20s". The steps of the job are detailed: Set up job, Run actions/checkout@v4, Set up JDK 17, Setup Gradle, Build with Gradle Wrapper, Upload Artifact, Post Setup Gradle, Post Set up JDK 17, Post Run actions/checkout@v4, and Complete job. A red oval highlights the "Search logs" input field.

- When done, go back to the pull request. (You can just click the "back arrow" in your browser.) Now, let's review the changes (as another user would if we added them as a reviewer). Click on the *Files changed* tab at the top to get to a color-code view of the changes.

The screenshot shows the GitHub Pull Request interface for a pull request titled "Add functionality to upload artifact #3". The "Files changed" tab is highlighted with a red oval. The list of changed files includes ".github/workflows/pipeline.yml", showing a diff between main and artifact branches. The changes include adding a new "branches: [ "artifact" ]" line and changing "pull\_request: branches: [ "main" ]" to "branches: [ "main", "artifact" ]". A red oval also highlights the "Review changes" button.

4. You may have noticed that there is a problem. The branches value in main would be overwritten with the value of artifact that we had in that branch. Let's make some review comments on this. Hover over one of the lines in green that says `branches: ["artifact"]`. You should see a + sign popup. Click on that and you'll get a pop-up to add a comment.

The screenshot shows a GitHub pull request interface. At the top, there are navigation links: "Changes from all commits", "File filter", "Conversations", "Jump to", and a gear icon. Below this is a code diff for a file named ".github/workflows/pipeline.yml". The diff shows the following changes:

```

@@ -9,9 +9,9 @@
@@ name: Java CI with Gradle

 9   9
10  10  on:
11  11  push:
12  -  branches: [ "main" ]
12 + branches: [ "artifact" ]
13  13  pull_request:
14  -  branches: [ "main" ]

```

The line `12 + branches: [ "artifact" ]` has a blue plus sign icon next to it, indicating it's a new addition. Below the diff, there's a comment input field with the placeholder "Leave a comment". A toolbar above the input field includes "Write" (selected), "Preview", and various rich text editing icons. At the bottom of the input field, there are status messages: "Markdown is supported", "Paste, drop, or click to add files", and a note "Please fill out this field.".

5. In the comment field, enter text like "*Need to change artifact to main before merge*". Then click on the *Start a review* button.

The screenshot shows the same GitHub pull request interface as the previous one, but now with a comment added. The comment text is "Need to change artifact to main before merge". The "Write" tab is still selected. At the bottom of the comment input field, there are three buttons: "Cancel", "Add single comment", and a green "Start a review" button.

6. This will have started a larger review with your comment pending. Repeat the same 2 steps above for the other line below that one with *branches*: [ "artifact" ].

Changes from all commits ▾ File filter ▾ Conversations ▾ Jump to ▾ ⚙

11 .github/workflows/pipeline.yml

```
@@ -9,9 +9,9 @@ name: Java CI with Gradle
 9   9
10  10   on:
11  11     push:
12  12   + branches: [ "main" ]
12  12   + branches: [ "artifact" ]
```

gwstudent2 Pending

Need to change artifact to main before merge

Reply...

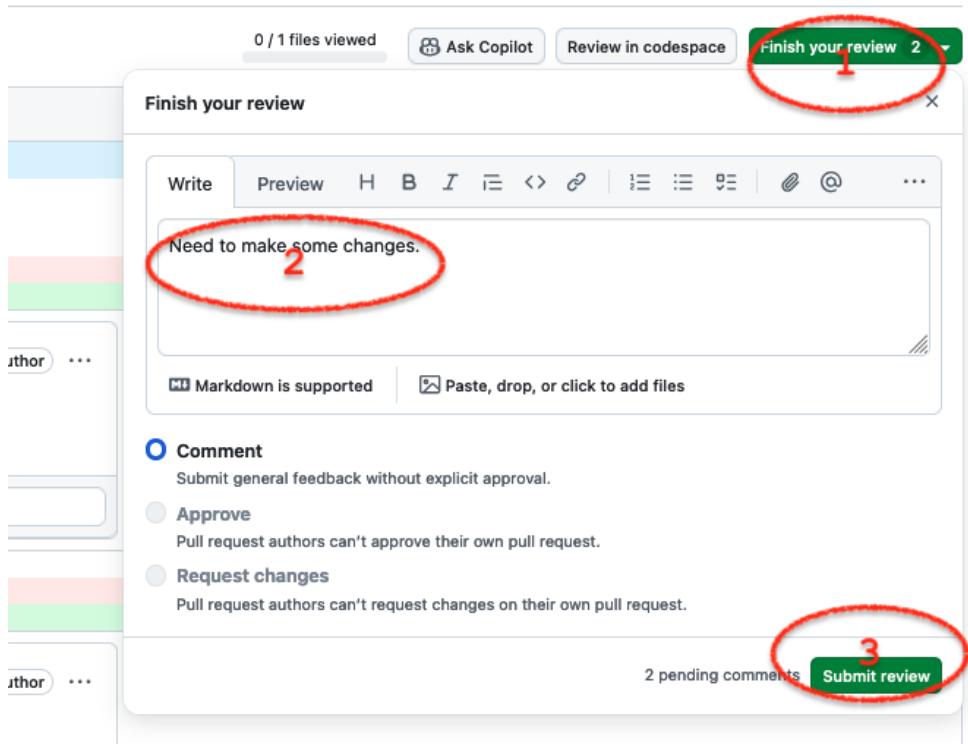
```
13 13   pull_request:
14 14   - branches: [ "main" ]
14 14   + branches: [ "artifact" ]
```

gwstudent2 Pending

Need to change artifact to main before merge

Reply...

7. Now that we've completed the individual comments, we can finish the review. Click on the *Finish your review (2)* button in the top right. In the main comment section, you can just add some text like "Need to make some changes". As the author of the pull request, your only option will be to leave an overall comment (*vs Approve* or *Request changes*). When ready, click on the green *Submit review* button.



8. After this, you'll see an indicator that your review was submitted successfully. Since we've realized we need to make some changes, let's edit the file in the browser. To do this, let's edit the `pipeline.yml` file in the `artifact` branch. To get there, you can just use the quick link below and substitute your GitHub userid for `<github-userid>`.

<https://github.com/<github-userid>/greetings-ci/edit/artifact/.github/workflows/pipeline.yml>

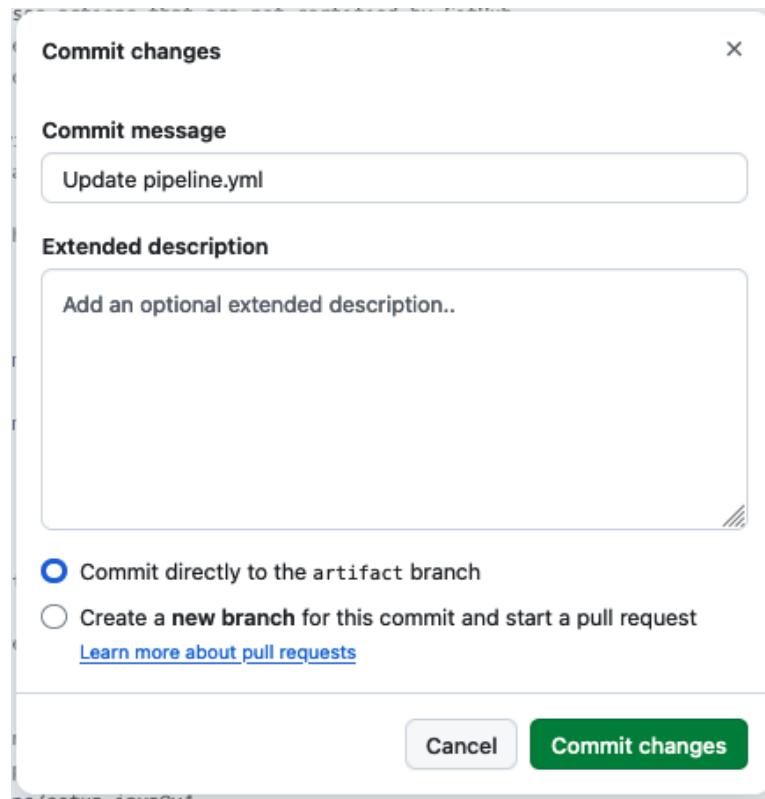
9. In the edit screen, change [ "artifact" ] to [ "main" ] in both lines.

```

1 # This workflow uses actions that are not certified by GitHub.
2 # They are provided by a third-party and are governed by
3 # separate terms of service, privacy policy, and support
4 # documentation.
5 # This workflow will build a Java project with Gradle and cache/restore any dependencies to improve t
6 # For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-a
7
8 name: Java CI with Gradle
9
10 on:
11   push:
12     branches: [ "main" ]
13     pull_request:
14       branches: [ "main" ]
15
16 jobs:
17   build:

```

10.. Click on the ***Commit changes...*** button in the upper right. In the dialog that comes up, you can add an extended description if you want. Leave the option set to "*Commit directly to the artifact branch*" and click "*Commit changes*" when ready.



11.. Now if you go back to the pull request, you can look at the *Commits* tab and see both of our commits as well as look at the *Files changed* tab and see that the only difference now is the addition of the step as intended.

Direct links are below (assuming you replace <github-userid> with your actual GitHub userid. (Also replace the "2" with a different number if your pull request has a different number.)

<https://github.com/<github-userid>/greetings-ci/pull/2/commits>

<https://github.com/<github-userid>/greetings-ci/pull/2/files>

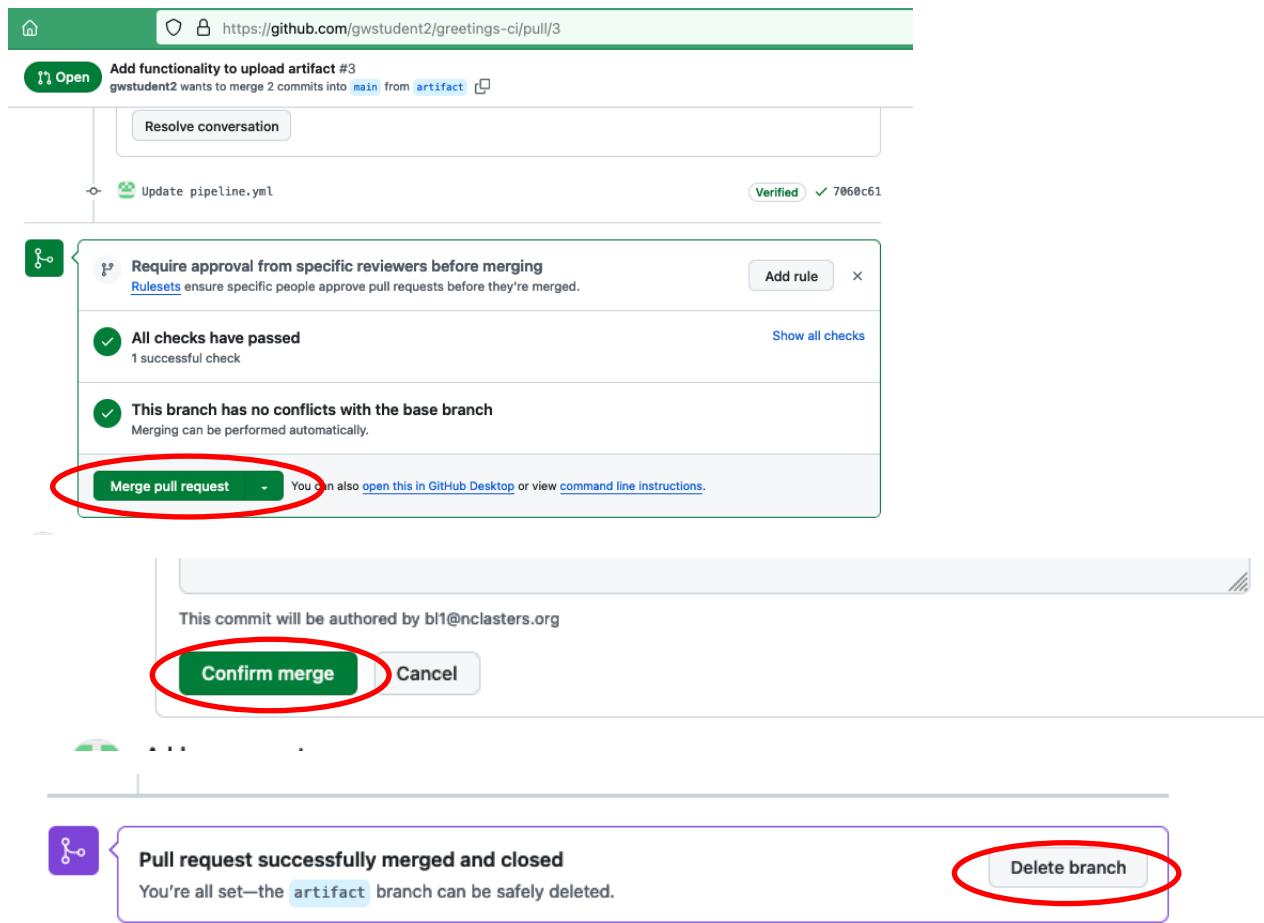
(Another workflow run will have been kicked off, but this should have also succeeded.)

The screenshot shows the GitHub Actions interface under the "All workflows" tab. On the left, there are filters for "Actions", "New workflow", and "All workflows" (which is selected). Below these are sections for "Management", "Caches", "Attestations", and "Runners". The main area displays a table of "3 workflow runs". The columns are "Event", "Status", "Branch", and "Actor". The rows show the following details:

Event	Status	Branch	Actor
Add functionality to upload artifact	SUCCEEDED	main	Java CI with Gradle #3: Pull request #3 synchronize by gwestudent2
Add functionality to upload artifact	SUCCEEDED	main	Java CI with Gradle #2: Commit d9f7eaf pushed by gwestudent2
Create pipeline.yml	SUCCEEDED	main	Java CI with Gradle #1: Commit f2b5c3d pushed by gwestudent2

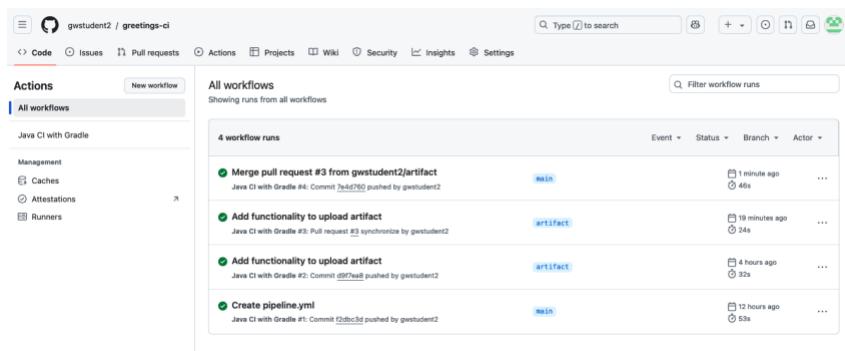
12. Now, you can go to the main page of the pull request (via the *Pull requests* tab at the top or link below) and then click on the *Merge pull request* button. Then click on the *Confirm merge* button. After this, you can click the Delete branch button if you want.

<https://github.com/<github-userid>/greetings-ci/pull/2>



The screenshot shows the GitHub pull request interface for a pull request from the 'artifact' branch to the 'main' branch. The pull request has been merged successfully. A red circle highlights the 'Merge pull request' button in the main pull request view. Another red circle highlights the 'Confirm merge' button in the confirmation dialog that appears when clicking the merge button. A third red circle highlights the 'Delete branch' button in the success message at the bottom of the page.

(After this, if you got to the *Actions* menu, you can see a final run of the workflow that was initiated when the merge was done.)



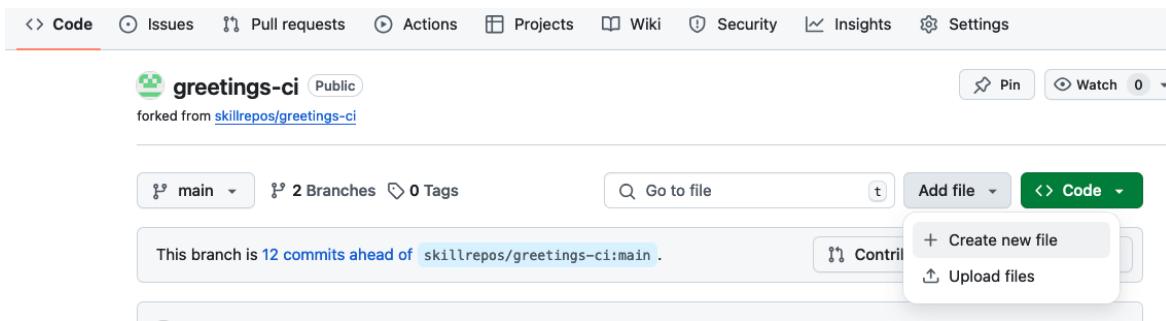
The screenshot shows the GitHub Actions menu for the 'gwestudent2 / greetings-ci' repository. It displays a list of workflow runs. A red circle highlights the 'Delete branch' button in the success message at the bottom of the page.

Workflow	Status	Time	Actor
Merge pull request #3 from gwstudent2/artifact	main	1 minute ago	...
Add functionality to upload artifact	artifact	19 minutes ago	...
Add functionality to upload artifact	artifact	4 hours ago	...
Create pipeline.yml	main	12 hours ago	...

## Lab 4 – Adding in a test case

Purpose: In this lab, we'll add a simple test case to download the artifact and verify it

- Now, let's create a new script to test our code. To create a new file via the browser, go back to the "Code" tab at the top. **To save time, we'll just do this on the *main* branch.** Click on the *Add file* button next to the green *Code* button. From the list that pops up, select *Create new file*.



- In the new editor that pops up, you'll be at the location to type in a name. You can name this "test-script.sh". Then copy and paste the following code into the new file. (A screenshot is shown after the code so you can see how things line up.) (This code is also available at <https://gist.github.com/brentlaster/f1c922ff4266882f0e5f2982da053cde>)

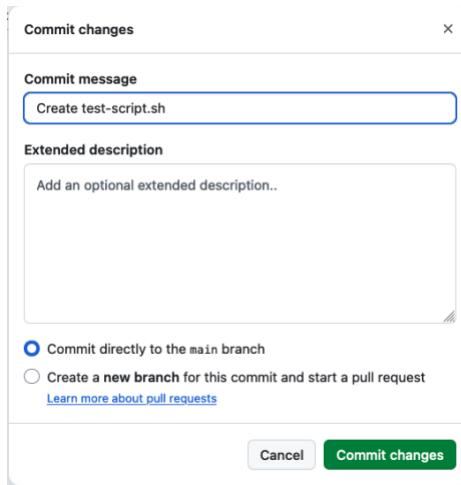
```
# Simple test script for greetings jar

set -e

java -jar build/libs/greetings-ci-$1.jar ${@:2} > output.bench
IFS=' ' read -ra ARR <<< "${@:2}"
for i in "${ARR[@]}"; do
    grep "^$i$" output.bench
done
```

A screenshot of the GitHub code editor for the 'test-script.sh' file. The code is identical to the one above. At the top right, there are 'Cancel changes' and 'Commit changes...' buttons. Below the code editor, there are 'Edit' and 'Preview' buttons, and settings for 'Spaces', '2', and 'No wrap'.

3. This script takes the version of the jar to run as its first parameter and the remaining values passed in as the rest of the parameters. Then it simply cycles through all but the first parameter checking to see if they print out on a line by themselves. Go ahead and click on the *Commit changes...* button to commit this file into the repository on the *test* branch. Just leave the dialog settings as-is or add a description if you want.



4. Let's add code to version our artifact. For simplicity, we'll just use the current run number as the "patch" component of the version number. And we'll just do a simple move (rename) of the file with the version number to get the artifact "versioned".

As you've done before, edit the *pipeline.yml* file (select it and click on the *pencil* icon).

Add the code below in the *build* job between the steps for "*Build with Gradle Wrapper*" and "*Upload Artifact*". The code to add is below and the screenshot shows where to put it. (Make sure to align it with the other steps in the *build* job.)

```

- name: Version Artifact
  run: >
    mv build/libs/greetings-ci.jar
    build/libs/greetings-ci-0.0.${{ github.run_number }}.jar

31   # Configure Gradle for optimal use in GitHub Actions, including caching of downloaded dependencies.
32   # See: https://github.com/gradle/actions/blob/main/setup-gradle/README.md
33   - name: Setup Gradle
34     uses: gradle/actions/setup-gradle@af1da67850ed9a4cedd57bfd976089dd991e2582 # v4.0.0
35
36   - name: Build with Gradle Wrapper
37     run: ./gradlew build
38
39   - name: Version Artifact
40     run: >
        mv build/libs/greetings-ci.jar
        build/libs/greetings-ci-0.0.${{ github.run_number }}.jar
41
42   - name: Upload Artifact
43     uses: actions/upload-artifact@v4.3.0
44     with:
45
46

```

5. Since each job executes on a separate runner system, we need to make sure our new test script is available on the runner that will be executing the tests. For simplicity, we can just add it to the list of items that are included in the uploading of artifacts. Modify the **path** section of the "Upload Artifact" step in the "build" job to look like below.

```
- name: Upload Artifact
  uses: actions/upload-artifact@v4.3.0
  with:
    name: greetings-jar
    path: |
      build/libs
      test-script.sh
```

The screenshot below shows where to make this change.

```
38
39      - name: Version Artifact
40        run: >
41          mv build/libs/greetings-ci.jar
42          build/libs/greetings-ci-0.0.${{ github.run_number }}.jar
43
44      - name: Upload Artifact
45        uses: actions/upload-artifact@v4.3.0
46        with:
47          name: greetings-jar
48          path: |
49            build/libs
50            test-script.sh
51
```

6. Now, we'll add the job definition for a job called "test-run" that runs on ubuntu-latest. What this code does is wait for the build job to complete (the *needs: build* part), then run two steps. The first step downloads the artifacts we uploaded before to have them there for the testing script. And the second step runs the separate testing script against the downloaded artifacts, making it executable first.

Since we want to test what we built, it will need to wait for the build job to be completed. That's what the "*needs: build*" part does in the code below.

The screenshot shows where it should go. Pay attention to indentation - *test-run:* should line up with *build: .* (If you see a wavy red line under part of the code, that probably means the indenting is not right.). Also, we will explain the *github.events.input.myTestArgs* in the next section.

```

test-run:

runs-on: ubuntu-latest
needs: build

steps:
- name: Download candidate artifacts
  uses: actions/download-artifact@v4
  with:
    name: greetings-jar

- name: Set up JDK 17
  uses: actions/setup-java@v4
  with:
    java-version: '17'
    distribution: 'temurin'

- name: Execute test
  shell: bash
  run: >
    chmod +x ./test-script.sh &&
    ./test-script.sh 0.0.${{ github.run_number }}
    ${{ github.event.inputs.myTestArgs || '1 2 3' }}

47
48   - name: Upload Artifact
49     uses: actions/upload-artifact@v4.3.0
50   with:
51     name: greetings-jar
52     path: |
53       build/libs
54       test-script.sh
55
56 test-run:
57
58 runs-on: ubuntu-latest
59 needs: build
60
61 steps:
62   - name: Download candidate artifacts
63     uses: actions/download-artifact@v4
64   with:
65     name: greetings-jar
66
67   - name: Set up JDK 17
68     uses: actions/setup-java@v4
69   with:
70     java-version: '17'
71     distribution: 'temurin'
72
73   - name: Execute test
74     shell: bash
75     run: >
76       chmod +x ./test-script.sh &&
77       ./test-script.sh 0.0.${{ github.run_number }}
78       ${{ github.event.inputs.myTestArgs || '1 2 3' }}
79

```

(Note: You can also get this code from  
<https://gist.github.com/techupskills/abb65b4d56ddf8758aad34ecee8f62c4>)

- Commit the changes as before ( to the "main" branch using the "Commit changes..." button).
- After the commit, if you switch to the *Actions* tab, you should see a new run of the workflow that executes both the *build* and *test-run* jobs. You can click on the run, then drill down via the job names to see what actually got executed.

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

← Java CI with Gradle

### Update pipeline.yml #10

[Summary](#)

Triggered via push 46 minutes ago  
gwstudent2 pushed → f6c8710 main

Status	Total duration	Artifacts
Success	41s	1

**pipeline.yml**  
on: push

```

graph LR
    build[build] -- 20s --> testRun[test-run]
    testRun -- 4s --> end

```

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

← Java CI with Gradle

### Update pipeline.yml #16

[Re-run all jobs](#) [...](#)

[Summary](#)

**test-run**  
succeeded 1 minute ago in 3s

[Search logs](#)

Jobs

- build
- test-run**

Run details

Usage

Workflow file

```

graph TD
    subgraph testRun [test-run]
        direction TB
        A[Set up job] --> B[Download candidate artifacts]
        B --> C[Set up JDK 17]
        C --> D[Execute test]
        D --> E[Post Set up JDK 17]
        E --> F[Complete job]
    end

```

1 Run chmod +x ./test-script.sh && ./test-script.sh 0.0.16 1 2 3  
2 chmod +x ./test-script.sh && ./test-script.sh 0.0.16 1 2 3  
3 shell: /usr/bin/bash --norc -e -o pipefail {0}  
4 env:  
5 JAVA\_HOME: /opt/hostedtoolcache/Java\_Temurin-Hotspot\_jdk/17.0.13-11/x64  
6 JAVA\_HOME\_17\_X64: /opt/hostedtoolcache/Java\_Temurin-Hotspot\_jdk/17.0.13-11/x64  
7 1  
8 2  
9 3

> Set up job 1s  
> Download candidate artifacts 1s  
> Set up JDK 17 0s  
**Execute test 0s**  
1 Run chmod +x ./test-script.sh && ./test-script.sh 0.0.16 1 2 3  
2 chmod +x ./test-script.sh && ./test-script.sh 0.0.16 1 2 3  
3 shell: /usr/bin/bash --norc -e -o pipefail {0}  
4 env:  
5 JAVA\_HOME: /opt/hostedtoolcache/Java\_Temurin-Hotspot\_jdk/17.0.13-11/x64  
6 JAVA\_HOME\_17\_X64: /opt/hostedtoolcache/Java\_Temurin-Hotspot\_jdk/17.0.13-11/x64  
7 1  
8 2  
9 3

> Post Set up JDK 17 0s  
> Complete job 0s

9. Go back to the page for the workflow run and scroll to the bottom. There you'll find our artifact named "greetings-jar". Click on either the download link or the artifact name (red circles) and download the artifact locally to your machine.

Triggered via push 49 minutes ago  
gwstudent2 pushed -> f6c8710 main

Status: Success | Total duration: 41s | Artifacts: 1

**pipeline.yml**  
on: push

**build** 20s → **test-run** 4s

**build summary**

Gradle Root Project	Requested Tasks	Gradle Version	Build Outcome	Build Scan®
greetings-ci	build	4.10	✓	Not published

► Caching for Gradle actions was enabled - expand for details

Job summary generated at run-time

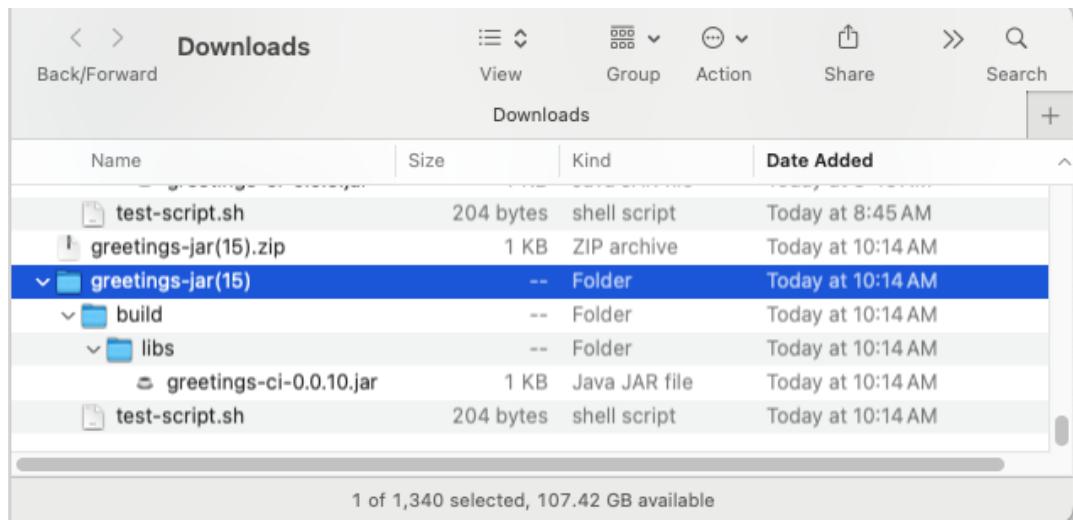
**Artifacts**  
Produced during runtime

Name	Size
greetings-jar	1.29 KB

<https://github.com/gwstudent2/greetings-ci/actions/runs/11743932877/artifacts/2163450194>

Download greetings-jar

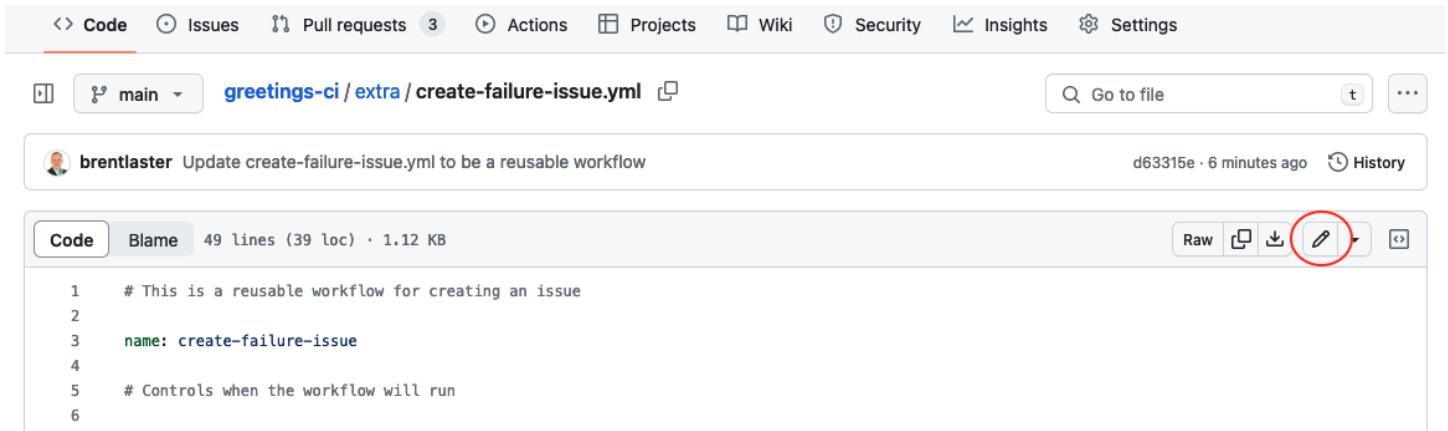
- . 10. After downloading the artifact, you will have a zip file locally on your machine. Expand that and you should be able to see the test script, and also the versioned artifact.



## Lab 5: Working with fast feedback and automatically reporting issues

Purpose: Learning how to get fast feedback and automatic failure reporting in our pipeline

1. For this lab, we're going to leverage a reusable workflow that will be able to automatically create a GitHub issue in our repository. And then we will invoke that workflow from our current workflow. First though, we need to make sure that the *Issues* feature is turned on for your repository.
  
2. The workflow to create the issue using a REST API call is already written to save time. It is in the main project under "extra/create-failure-issue.yml". You need to get this file in the .github/workflows directory. You can just move it via GitHub with the following steps.
  - a. In the repository, browse to the "extra" folder and to the "create-failure-issue.yml" file.
  - b. Take a few moments to look over the file and see what it does. Notice that:
    - i. It has a *workflow\_call* section in the "on" area, which means it can be run from another workflow.
    - ii. It has a *workflow\_dispatch* section in the "on" area, which means it can be run manually.
    - iii. It has two inputs - a title and body for the issue.
    - iv. The primary part of the body is simply a REST call (using the GITHUB\_TOKEN) to create a new issue.
  - c. Click the pencil icon to edit it.



The screenshot shows a GitHub repository interface. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the navigation is a search bar with a 'Go to file' button and a three-dot menu. The main content area displays a file named 'greetings-ci / extra / create-failure-issue.yml'. A commit history is shown, with one entry by 'brentlaster' titled 'Update create-failure-issue.yml to be a reusable workflow'. The file itself contains the following YAML code:

```
1 # This is a reusable workflow for creating an issue
2
3 name: create-failure-issue
4
5 # Controls when the workflow will run
```

- d. In the filename field at the top, change the name of file. Use the backspace key to backspace over "extra/" making sure to backspace over the word. Then type in the path to put it in the workflows ".github/workflows/create-failure.yml".

brentlaster / greetings-ci Public

forked from skillrepos/greetings-ci

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [!](#)

[greetings-ci / .github / workflows / create-failure-issue.yml](#) in [main](#)

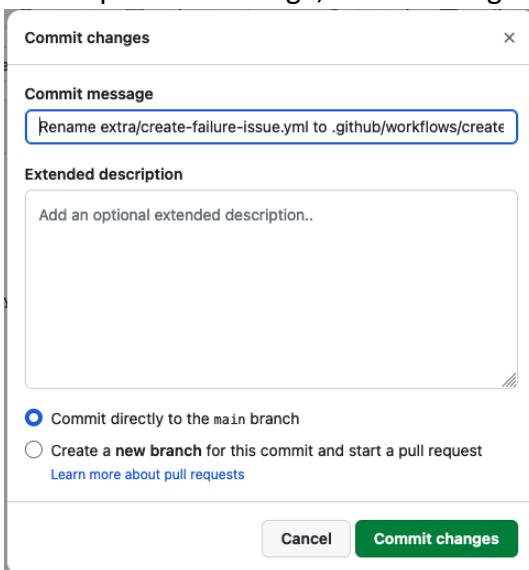
[Edit](#) [Preview](#)

```

1  # This is a reusable workflow for creating an issue
2
3  name: create-failure-issue

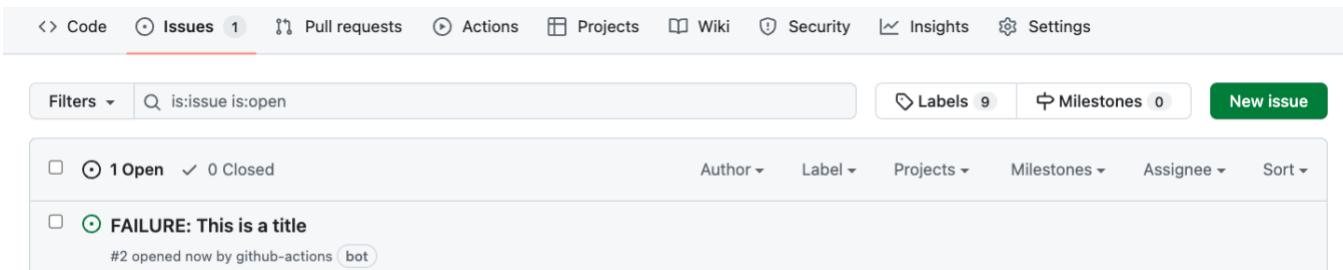
```

- e. To complete the change, click on the green "Commit changes" button



3. Go back to the Actions tab. You'll see a new workflow execution due to the rename. Also, in the Workflows section on the left, you should now see a new workflow titled "create-failure-issue". Click on that. Since it has a workflow\_dispatch event trigger available, we can try it out. Click on the "Run workflow" button and enter in some text for the "title" and "body" fields. Then click "Run workflow".

4. After a moment, you should see the workflow run start and then complete. If you now click on the Issues tab at the top, you should see your new issue there.



The screenshot shows the GitHub Issues page with the following details:

- Header navigation: Code, Issues (1), Pull requests, Actions, Projects, Wiki, Security, Insights, Settings.
- Search bar: is:issue is:open
- Filters: 1 Open, 0 Closed.
- Issue list:
  - 1 issue found: FAILURE: This is a title
  - Details: #2 opened now by github-actions (bot)
- Sort dropdown: Author, Label, Projects, Milestones, Assignee, Sort.

5. Now that we know that our new workflow works as expected, we can make the changes to the previous workflow to "call" this if we fail. Edit the pipeline.yml file and add the following lines as a new job and set of steps at the end of the workflow. (For convenience, these lines are also in the file "extra/job-create-issue-on-failure.txt" if you want to copy and paste from there.). The "create-issue-on-failure" line should line up with the other job names.

We have some other changes to make so don't commit yet.

```
create-issue-on-failure:
```

```
needs: test-run

permissions:
  issues: write

if: always() && failure()
uses: ./github/workflows/create-failure-issue.yml
with:
  title: "Automated workflow failure issue for commit ${{ github.sha }}"
  body: >
    "This issue was automatically created by the workflow ${{ github.workflow }}"
```

```

68   uses: actions/setup-java@v4
69   with:
70     java-version: '17'
71     distribution: 'temurin'
72   - name: Print Inputs
73     shell: bash
74     run: echo github.event.inputs.myTestArgs = "${{ github.event.inputs.myTestArgs }}"
75
76   - name: Execute test
77     shell: bash
78     run: >
79       chmod +x ./test-script.sh &&
80       ./test-script.sh 0.0.${{ github.run_number }}
81       ${{ github.event.inputs.myTestArgs || '1 2 3' }}
82
83 | create-issue-on-failure:
84
85   needs: test-run
86
87   permissions:
88     issues: write
89
90   if: always() && failure()
91   uses: ./github/workflows/create-failure-issue.yml
92   with:
93     title: "Automated workflow failure issue for commit ${{ github.sha }}"
94     body: "This issue was automatically created by the workflow ${{ github.workflow }}"
--
```

6. Let's make one more change to make it easier to run our workflow manually to try things out, start runs, etc. In the "on:" section near the top, add the code below at the bottom of the "on" section. ("workflow\_dispatch" should line up with "pull" and "push") and then commit the changes.

```

workflow_dispatch:
  inputs:
    myTestArgs:
      description: 'Testing values'
```

```

8   name: Java CI with Gradle
9
10  on:
11    push:
12      branches: [ "main" ]
13    pull_request:
14      branches: [ "main" ]
15    workflow_dispatch:
16      inputs:
17        myTestArgs:
18          description: 'Testing values'
19
20
21  jobs:
22    build:
23
24      runs-on: ubuntu-latest
25      permissions:
26        contents: read
~~

```

7. Commit the changes back to the main branch. This will cause a new workflow run (that should succeed) that you can ignore.

8. The addition of the `workflow_dispatch` trigger means that you can now run the workflow manually and input some test arguments to override the "1 2 3" ones we have for a default. To see how this works, change to the `Actions` tab and then click on the left side under "All workflows", click on the "`Java CI with Gradle`" workflow link.

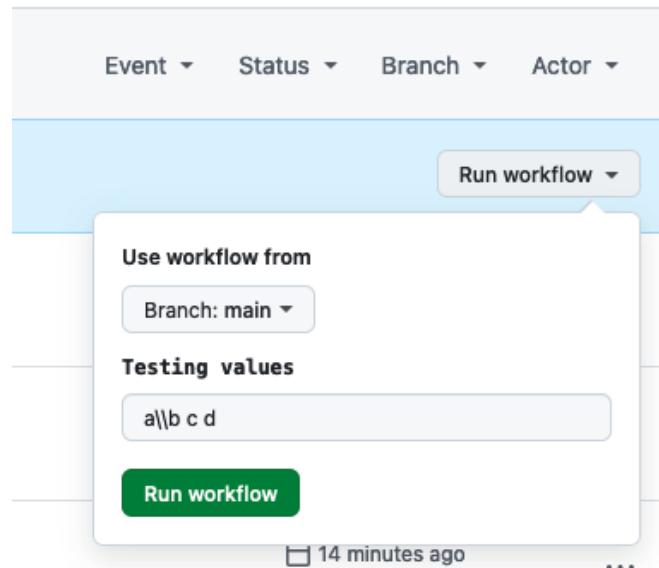
A blue bar will show up with a "Run workflow" button to the right. Click on the "Run workflow" button.

The screenshot shows the GitHub Actions interface. The top navigation bar includes links for Code, Issues, Pull requests, Actions (which is highlighted with a red circle and labeled #1), Projects, Wiki, Security, Insights, and Settings. The left sidebar under 'Actions' shows 'All workflows' with 'create-failure-issue' and 'Java CI with Gradle' (link circled with red #2). The main content area displays the 'Java CI with Gradle' workflow with its pipeline file 'pipeline.yml'. It shows '28 workflow runs' and a note: 'This workflow has a workflow\_dispatch event trigger.' Below this, a specific run is listed: 'Java CI with Gradle #28: Manually run by gwstudent2' (status 'main'), with a timestamp of '40 minutes ago ...' and a duration of '42s'. A 'Run workflow' button is circled with red #3.

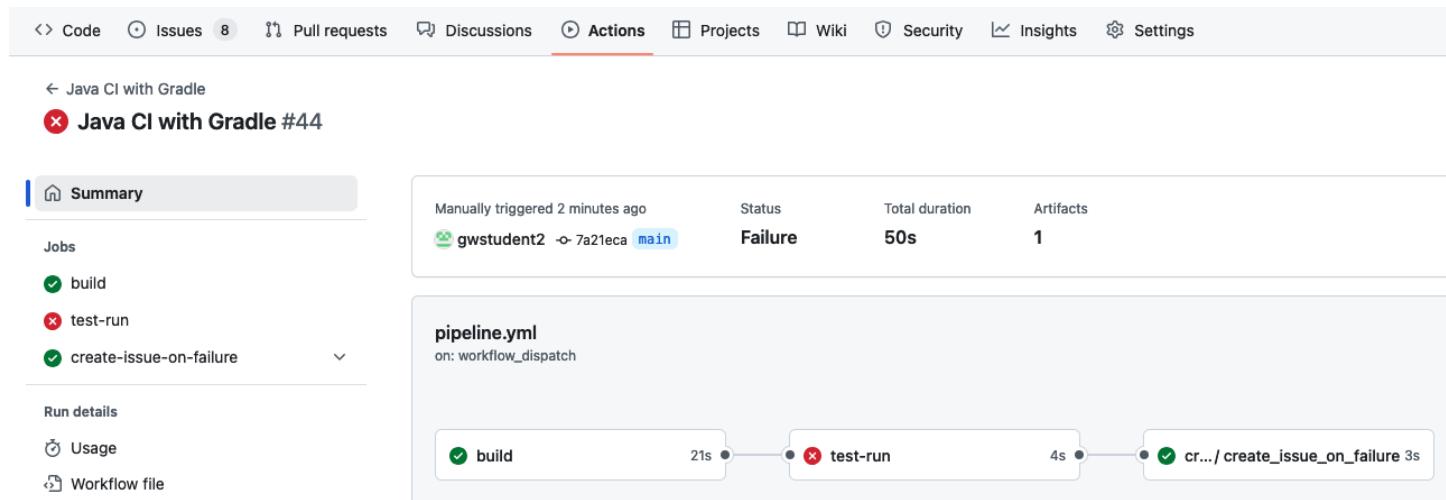
9. For the test argument values we'll put in ones that will cause an issue so that our automatic issue creation will be triggered. To do this, we'll include two backslashes in the input (they can be anywhere, but will cause the test job to fail, and thus cause an issue to be generated).

So, in the dialog that comes up from the *Run workflow* button, enter in something like the string below and then click on the *Run workflow* button.

a\\b c d



10. After this, the workflow will run, the *test-run* job should fail, and the *create\_issue\_on\_failure* job should succeed in detecting the failure. You'll be able to see this by looking at the job graph.



11. You should also be able to look under the *Issues* tab and see the new issue that resulted.

The screenshot shows the GitHub Issues page with the following details:

- Header navigation: Code, Issues (2), Pull requests, Actions, Projects, Wiki, Security, Insights, Settings.
- Search bar: is:issue is:open.
- Filters: 2 Open, 8 Closed.
- Issues listed:
  - Failure: Automated workflow failure issue for commit 7a21eca8461b3b9915700007ea7611e89b9d28a1 #12 opened 4 minutes ago by github-actions (bot)
  - Failure: This is a title #5 opened 17 hours ago by github-actions (bot)
- Bottom navigation: Author, Label, Projects, Milestones, Assignee, Sort.
- Buttons: Labels (9), Milestones (0), New issue.

END OF LAB

## Lab 6 – Adding Environments and Releases

Purpose: In this lab, we'll look at how to add staging (blue, green) and production environments and releases.

1. Let's add some deploy jobs to our pipeline.yaml file. Edit the .github/workflows/pipeline.yaml file. For simplicity, we can just do this in the main branch.

The screenshot shows the GitHub pipeline.yaml editor with the following details:

- Header navigation: Code, Issues (41), Pull requests, Discussions, Actions, Projects, Wiki, Security, Insights, Settings.
- File path: greetings-ci/.github/workflows/pipeline.yaml.
- Branch: main.
- Code content:

```
1 # This workflow uses actions that are not certified by GitHub.
2 # They are provided by a third-party and are governed by
3 # separate terms of service, privacy policy, and support
4 # documentation.
5 # This workflow will build a Java project with Gradle and cache/restore any dependencies to improve the workflow execution
6 # For more information see: https://help.github.com/actions/language-and-framework-guides/building-and-testing-java-with-gr
7
8 name: Java CI with Gradle
9
10 on:
11   push:
```
- Right sidebar: Marketplace (selected), Documentation, Cache action card.

2. We can illustrate blue/green deployment with new branches such as "blue" and "green". So, let's modify the "on:" section first to run the workflow on a push to any of these. Modify the **on: push:** command to be like the following.

```
on:
  push:
    branches: [ "main", "blue", "green" ]
    ...
    7
    8   name: Java CI with Gradle
    9
    10  on:
    11    push:
    12      branches: [ "main", "blue", "green" ]
    13    pull_request:
    14      branches: [ "main" ]
```

3. You can also remove the "pull\_request" portion.

```

9
10    on:
11      push:
12        branches: [ "main", "blue", "green" ]
13        workflow_dispatch:
14          inputs:
15            myTestArgs:
16              description: 'Testing values'
17

```

4. Now, let's add the job for deploying a "stage" environment/release. The code for this job is already done for you and can be copied from the file [extra/deploy-stage.txt](#). You can go to the same repository in another tab, open that file, and then just copy and paste. (Note double-check that the "create-issue-on-failure" job name didn't get moved to the end of the paste.)

(Note: You may need to add a space or two at the front of the first line of output once pasted to get it to line up correctly.)

This code essentially does the following:

- Waits for the build and test jobs to complete (line 79)
- Checks to see if the branch being pushed to is "blue" or "green" (line 80)
- Establishes an environment called "staging" (line 83)
- Sets the associated URL for the environment to the releases page (line 85)
- Checkouts the source code (line 87-90)
- Downloads the jar we built (line 92-95)
- Calls a GitHub Action to create a release that: (line 97-105)
  - is based on the tag we got from the build
  - is set as a draft and prerelease
  - includes the jar file we've built

```

66   steps:
67
68   - uses: actions/checkout@v3
69
70   - name: run-test
71     uses: ./github/actions/test-action
72     with:
73       artifact-version: ${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
74       arguments-to-print: ${{ github.event.inputs.myValues }}
75
76
77   deploy-stage:
78
79   needs: [build, test-run]
80   if: github.ref == 'refs/heads/blue' || github.ref == 'refs/heads/green'
81
82   runs-on: ubuntu-latest
83   environment:
84     name: staging
85     url: https://github.com/${{ github.repository }}/releases/tag/v${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
86
87   steps:
88   - uses: actions/checkout@v3
89     with:
90       fetch-depth: 0
91
92   - name: Download candidate artifacts
93     uses: actions/download-artifact@v3
94     with:
95       name: greetings-jar
96
97   - name: GH Release
98     uses: softprops/action-gh-release@v0.1.14
99     with:
100       tag_name: v${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
101       prerelease: true
102       draft: true
103
104       name: ${{ github.ref_name }}
105
106     files: |
107       greetings-ci-${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}.jar
108

```

- Now, let's add the job for deploying a "prod" (production) environment/release from a pull-request being merged into "main". This job can be inserted **between the "deploy-stage" job and the "create-issue-on-failure" job**. The code for this job is already done for you and can be copied from the file [extra/deploy-prod.txt](#). Just copy and paste. You can copy and paste this one the same way from the other tab.

(Note: You may need to add a space or two at the front of the first line of output once pasted to get it to line up correctly.)

This code essentially does the following:

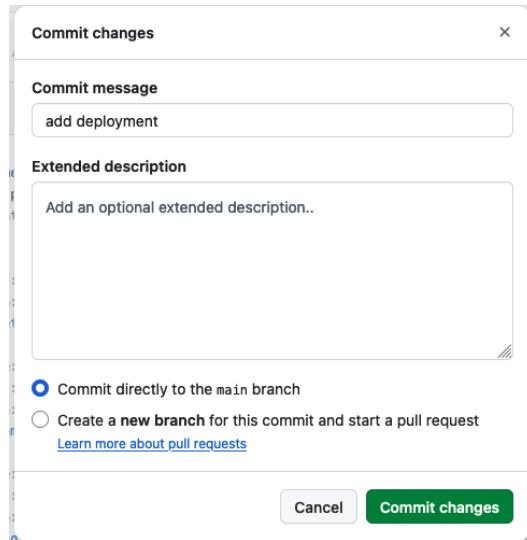
- Waits for the build and test jobs to complete (line 114)
- Checks to see if we got here on the main branch (line 115)
- Establishes an environment called "production" (line 119)
- Sets the associated URL for the environment to the releases page (line 120)
- Checkouts the source code (line 123-125)
- Downloads the jar we built (line 127-130)
- Calls a GitHub Action to create a release that: (line 132-140)
  - is based on the tag we got from the build
  - is named as "Production"
  - includes the jar file we've built and the CHANGELOG

```

105
106     files: |
107         greetings-ci-${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}.jar
108
109
110
111
112 deploy-prod:
113
114     needs: [build, test-run]
115     if: github.ref == 'refs/heads/main'
116
117     runs-on: ubuntu-latest
118     environment:
119         name: production
120         url: https://github.com/${{ github.repository }}/releases/tag/v${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
121     steps:
122
123         - uses: actions/checkout@v3
124             with:
125                 fetch-depth: 0
126
127         - name: Download candidate artifacts
128             uses: actions/download-artifact@v3
129             with:
130                 name: greetings-jar
131
132         - name: GH Release
133             uses: softprops/action-gh-release@v0.1.14
134             with:
135                 tag_name: v${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
136                 generate_release_notes: true
137                 name: Production
138                 files: |
139                     CHANGELOG.md
140                     greetings-ci-${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}.jar
141
142
143 create-issue-on-failure:

```

6. Go ahead and commit your changes **to the main branch** with an appropriate commit message.



7. This will kick off a new run of the workflow and will create an initial production deployment because of a change in main. After the run completes, you can click on the link in the deploy-prod job in the "Jobs" view to see the release.

← Java CI with Gradle  
feat: add deployment #30

Summary

Triggered via push 2 minutes ago  
gwstudent pushed → 123d14c main Status Success Total duration 53s Artifacts 1

Jobs

- build
- test-run
- deploy-stage
- deploy-prod
- create-issue-on-failure

Run details

- Usage
- Workflow file

pipeline.yml  
on: push

```
graph LR; build[build] -- 17s --> testRun[test-run]; testRun -- 4s --> deployProd[deploy-prod]; deployProd -- 5s --> deployStage[deploy-stage]; deployProd -- 5s --> createIssueOnFailure[create-issue-on-failure];
```

gwstudent / greetings-ci Public  
forked from skillrepos/greetings-ci

Code Issues 2 Pull requests Actions Projects Wiki Security Insights Settings

Releases / v0.6.0

Production Latest

github-actions released this 8 minutes ago v0.6.0 → 7017919

What's Changed

- Test action by @gwstudent in #7

Full Changelog: v0.5.1...v0.6.0

Contributors

gwstudent

Assets 3

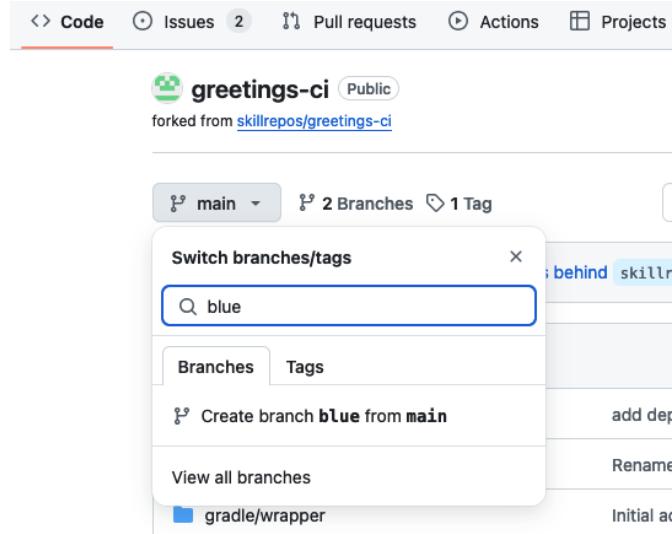
CHANGELOG.md	1.15 KB	8 minutes ago
Source code (zip)		8 minutes ago
Source code (tar.gz)		8 minutes ago

END OF LAB

## Lab 7 – Exercising the entire workflow

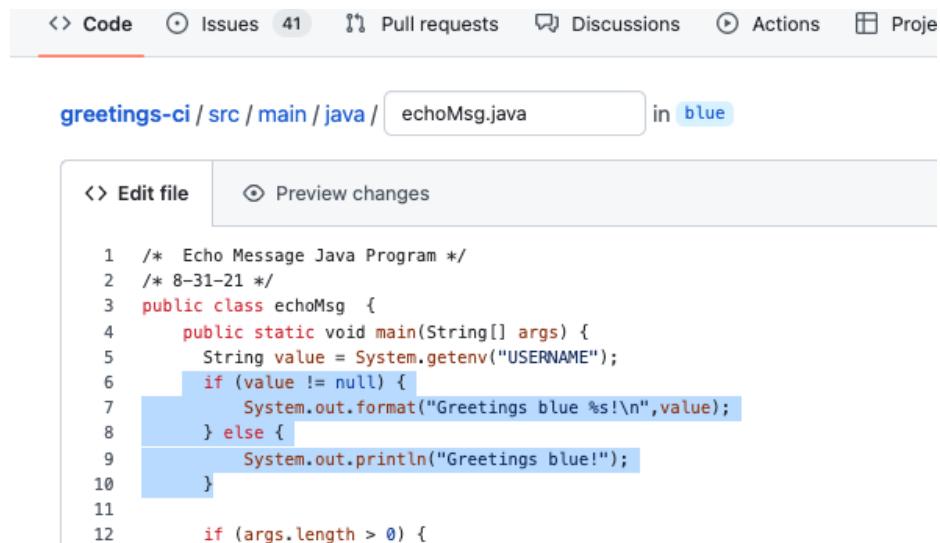
Purpose: In this lab, we'll see how to make a change in source code and have it processed through the pipeline.

1. In the example of using a "blue/green" environment, let's **create a branch called "blue" from the "main" branch** to make some changes on. Do this just as you've done before. (Note that you will need to make sure you're on the main branch before creating the blue one.)

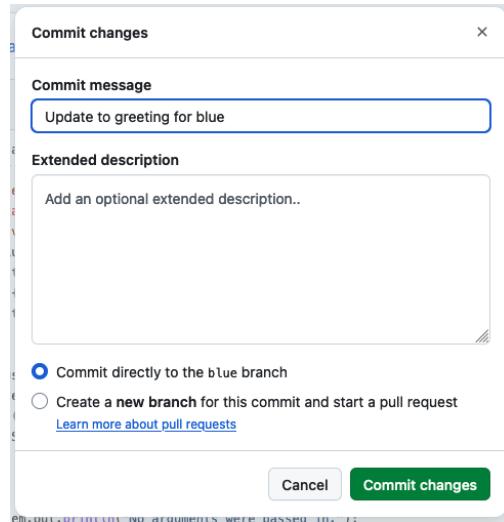


2. In the "blue" branch, edit the file src/main/java/echoMsg.java. Make a simple, non-breaking change like adding "blue" to the lines that print out "Greetings". See text and figure below.

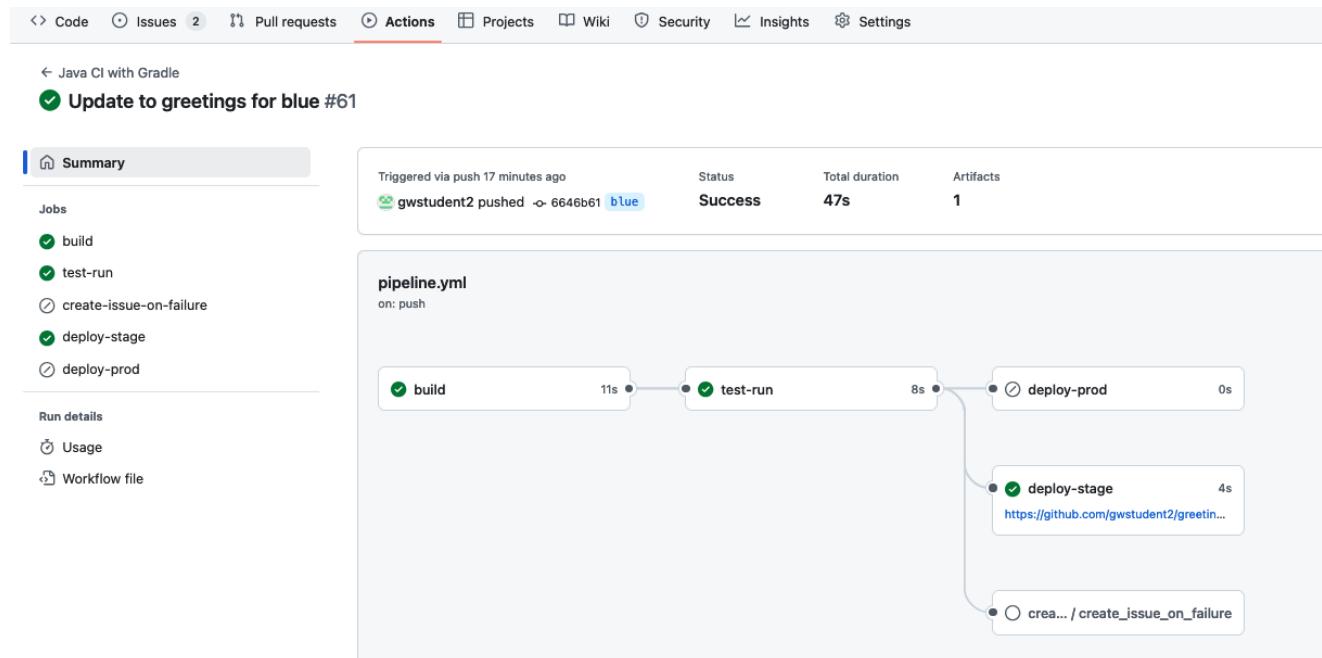
```
if (value != null) {
    System.out.format("Greetings blue %s!\n",value);
} else {
    System.out.println("Greetings blue!");
}
```



3. Commit the changes with an appropriate commit message.



4. After the workflow run completes, you can click on the run and look at the job graph. You should be able to see that it executed the build and test pieces and then deployed it to the stage environment.



5. Now, click on the **link** in the "deploy-stage" box. This will take you to the tagged version of the source repo.

The screenshot shows the GitHub Releases page for the 'blue' repository. The release 'v0.0.61' is highlighted as 'Latest'. It was created by 'github-actions' 15 minutes ago. The release notes link is 'v0.0.57...v0.0.61'. The assets section lists three files: 'greetings-ci-0.0.61.jar' (1010 Bytes, 15 minutes ago), 'Source code (zip)' (30 minutes ago), and 'Source code (tar.gz)' (30 minutes ago). There is also a 'Compare' button.

6. If you click on the "Releases" item next to "Tags", you can see the draft release that was created.

The screenshot shows the GitHub Releases page with the 'Releases' tab selected. A draft release for 'blue' is listed, created 6 minutes ago by 'github-actions'. It has version 'v0.12.2' and commit 'a5926b9'. There is a 'Compare' button. To the right, there is a 'Draft a new release' button and a search bar.

7. And, if you click on the main code page, in the lower right, you'll be able to see a new "Staging" deployment. You can click on that to see a list of recent deployments there.

The screenshot shows the GitHub Code page for the 'blue' repository. On the left, there are deployment logs for 'staging' and 'production' environments. The 'staging' log shows a deployment 19 minutes ago. The 'production' log shows a deployment 34 minutes ago. Below these logs is a link '+ 12 deployments'. On the right, there is a 'Languages' section.

Screenshot of the GitHub Deployments page for the 'greetings-ci' repository. The 'staging' environment is selected. The 'Latest deployments' section shows three successful deployments to the 'staging' environment, each with a timestamp and a link to the release page.

8. Since everything built ok, we can deploy this change to the production environment. To merge the changes, we can just create a pull request to main and merge it. To keep things simple, here's a link that you can copy and paste (substituting in *your GitHub userid* for <github-userid>)

<https://github.com/<github-userid>/greetings-ci/compare/main...blue>

Screenshot of the GitHub Comparing changes page for the 'greetings-ci' repository. The URL in the address bar is https://github.com/gwstudent2/greetings-ci/compare/main...blue. The page shows a comparison between 'main' and 'blue' branches. It indicates that the branches are 'Able to merge'. A 'Create pull request' button is visible at the top right.

- Click on the *Create pull request* button to start the new pull request. Then, click on the following *Create pull request* button to open the request.

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). Learn more about

The screenshot shows the GitHub interface for creating a pull request. At the top, there are dropdown menus for 'base: main' and 'compare: blue'. A green checkmark indicates 'Able to merge. These branches can be automatically merged.' Below this, there's a section to 'Add a title' with the placeholder 'Update to greetings for blue'. Under 'Add a description', there's a rich text editor with a toolbar and a placeholder 'Add your description here...'. At the bottom, there are buttons for 'Create pull request' and 'Markdown is supported'.

10. You should then be on a page for the pull request that shows all the checks. You can just go ahead and merge and confirm.

## Update to greetings for blue #13

The screenshot shows the GitHub pull request page for #13. It displays a comment from gwstudent2 stating 'No description provided.' Below this, it shows a deployment status: 'Update to greetings for blue' with a 'Verified' badge and commit hash '6646b61'. A deployment message says 'This branch was successfully deployed' with '1 active deployment'. At the bottom, there's a 'Merge pull request' button and a note: 'You can also open this in GitHub Desktop or view command line instructions.'

11. This should kick off another run of the action workflow in main. When it is done, it should have done a "production" deployment as you can see via the jobs graph for that run.

**Summary**

Triggered via push 2 minutes ago

gwstudent2 pushed -> bb8a555 **main**

Status: **Success** Total duration: **58s** Artifacts: **1**

**pipeline.yml**  
on: push

```

graph LR
    build[build] -- 21s --> testRun[test-run]
    testRun -- 7s --> deployProd[deploy-prod]
    deployProd -- 5s --> deployStage[deploy-stage]
    deployProd -- 0s --> deployStage

```

**Jobs**

- build
- test-run
- create-issue-on-failure
- deploy-stage
- deploy-prod

**Run details**

- Usage
- Workflow file

12. If you click in the link for the deploy-prod job, you should be able to go to the new *Production* deployment page.

Releases / v1.0.62

**Production** **Latest**

github-actions released this 2 minutes ago v1.0.62 -> bb8a555

**What's Changed**

- Update to greetings for blue by [@gwstudent2](#) in [#13](#)

Full Changelog: [v0.0.60...v1.0.62](#)

**Contributors**

gwstudent2

**Assets** 3

	Size	Created
<a href="#">greetings-ci-1.0.62.jar</a>	1010 Bytes	2 minutes ago
<a href="#">Source code (zip)</a>		3 minutes ago
<a href="#">Source code (tar.gz)</a>		3 minutes ago

END OF LAB

## Appendix 1: Alternate ways to "fork" repo if not allowed to use actual "Fork" button

### OPTION 1: Using Import

1. Sign into GitHub if not already signed in.
2. Go to <https://github.com/new/import>
3. On that page, fill out the form as follows:

In "Your source repository details", in the "The URL for your source repository \*" field, enter

<https://github.com/skillrepos/greetings-ci>

Under "Your new repository details", make sure your userid shows up in the "Owner \*" field and enter

**greetings-ci**

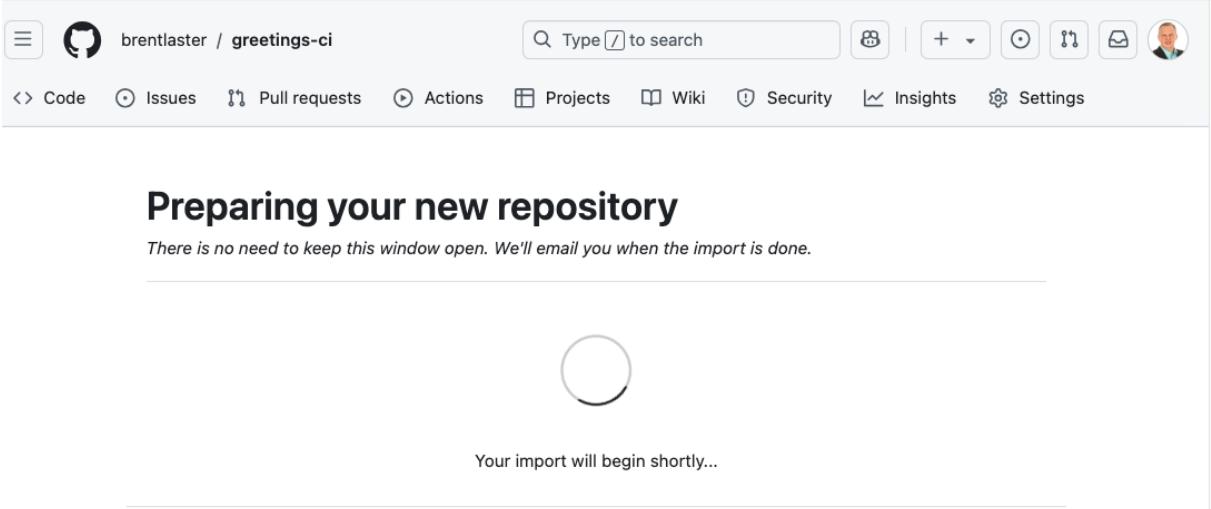
in the "Repository name \*" field.

The visibility field should be set to "Public".

Then click on the green "Begin import" button.

The screenshot shows the GitHub 'Import' page. At the top, there's a message: 'ended on April 12, 2024. For more details, see the [changelog](#)'. Below this, the 'Your source repository details' section has a red box around the 'The URL for your source repository \*' input field, which contains the URL <https://github.com/skillrepos/greetings-ci>. A red number '1' is placed to the right of this field. Below it is a link to 'Learn more about [importing git repositories](#)'. The 'Your new repository details' section has a red box around the 'Owner \*' dropdown, which shows 'brentlaster', and the 'Repository name \*' input field, which contains 'greetings-ci'. A red number '2' is placed to the right of the repository name field. Below this, there's a note: 'greetings-ci is available.' Under the visibility settings, the 'Public' option is selected (indicated by a blue radio button), and the description says 'Anyone on the internet can see this repository. You choose who can commit.' There are also 'Private' and 'Protected' options. At the bottom right of the form, there's a green 'Begin import' button with a red border, and a red number '3' is placed to its right. To the left of the 'Begin import' button are 'Cancel' and 'Next Step' buttons.

4. After this, you should see the import processing...

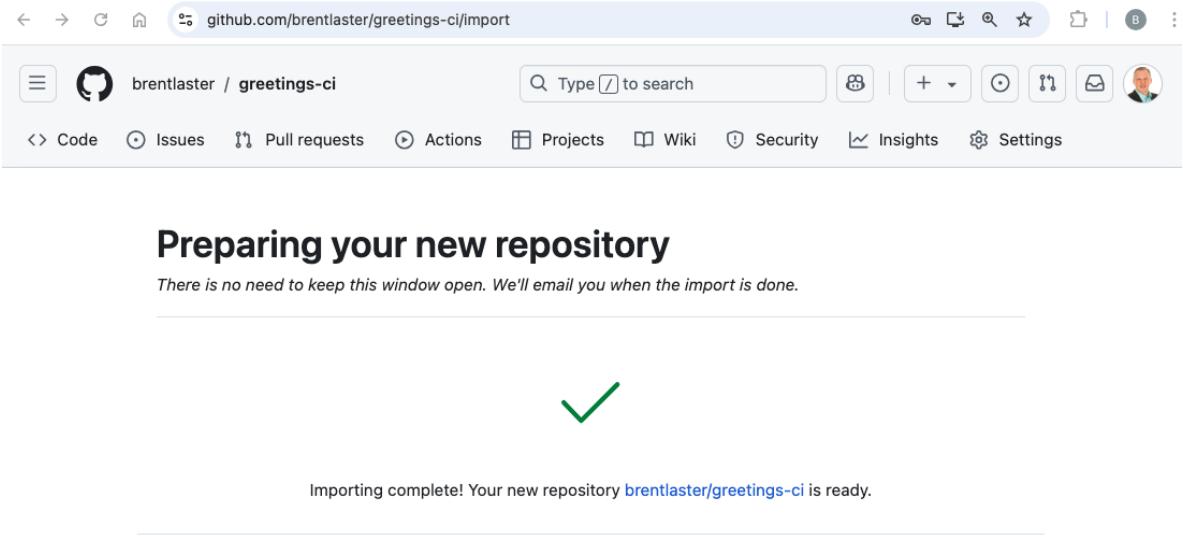


A screenshot of a GitHub repository page titled "greetings-ci". The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A search bar is at the top right. Below the title, the heading "Preparing your new repository" is displayed, followed by the subtext "There is no need to keep this window open. We'll email you when the import is done." In the center, there is a large circular loading icon with the text "Your import will begin shortly..." below it.

5. This will take several minutes to run. When done, you should see a "complete" message and your new repo will be available.

#### Appendix 1: Alternate ways to "fork" repo if not allowed to use actual "Fork" button

##### OPTION 1: Using Import



A screenshot of a GitHub repository page titled "greetings-ci". The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A search bar is at the top right. Below the title, the heading "Preparing your new repository" is displayed, followed by the subtext "There is no need to keep this window open. We'll email you when the import is done." In the center, there is a large green checkmark icon with the text "Importing complete! Your new repository [brentlaster/greetings-ci](#) is ready." below it.

#### Appendix 1: Alternate ways to "fork" repo if not allowed to use actual "Fork" button

##### OPTION 1: Using clone and push

1. Sign into GitHub if not already signed in.
2. Create a GitHub token or SSH key. If you are familiar with SSH keys, you can add your public key at <https://github.com/settings/keys>. Otherwise, you can just create a "classic" token by following the

© 2024 Tech Skills Transformations, LLC & Brent Laster

instructions at <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens#creating-a-personal-access-token-classic>. If you use a GitHub token, make sure to save a copy of it to use in the push step.

3. Clone down the [skillrepos/greetings-ci](#) repository.

```
git clone https://github.com/skillrepos/greetings-ci (if using token)
```

or

```
git clone git@github.com:skillrepos/greetings-ci (if using ssh)
```

3. Create a new repository in your GitHub space named greetings-ci. Go to <https://github.com/new>. Fill in the "repo name" field with "greetings-ci" and then click on the "Create repository" button.

The screenshot shows the GitHub 'Create a new repository' interface. At the top, there's a search bar and a 'New repository' button. The main form has fields for 'Repository template' (set to 'No template'), 'Owner' (set to 'gwstudent'), and 'Repository name' (set to 'greetings-ci'). A note says 'greetings-ci is available'. Below these, there's a section for 'Description (optional)' with a text input field. Under 'Visibility', 'Public' is selected. In the 'Initialize this repository with:' section, there's a checkbox for 'Add a README file'. The 'Add .gitignore' section shows a dropdown set to 'None'. The 'Choose a license' section shows a dropdown set to 'None'. A note at the bottom left says 'You are creating a public repository in your personal account.' The 'Create repository' button at the bottom right is highlighted with a red circle and the number '2'.

4. On the page that comes up after that, select the appropriate protocol (https or ssh) and then follow the instructions for "...or push an existing repository from the command line" to push your content

back to the GitHub repository. If you're using https you will be prompted for a password at push time. Just paste in the classic token. (Note that for security reasons, you will not see the token displayed.)

The screenshot shows a GitHub repository page for 'greetings-ci'. The URL in the address bar is <https://github.com/gwstudent/greetings-ci>. The page includes sections for 'Start coding with Codespaces', 'Add collaborators to this repository', and 'Quick setup — if you've done this kind of thing before'. A red oval highlights the 'Quick setup' section, which contains instructions for setting up the repository via desktop or command line. A second red oval highlights the command-line setup section, which includes a ProTip about using the provided URL for adding GitHub as a remote.

**Start coding with Codespaces**  
Add a README file and start coding in a secure, configurable, and dedicated development environment.  
[Create a codespace](#)

**Add collaborators to this repository**  
Search for people using their GitHub username or email address.  
[Invite collaborators](#)

**Quick setup — if you've done this kind of thing before**

Set up in Desktop or [HTTPS SSH](https://github.com/gwstudent/greetings-ci.git) <https://github.com/gwstudent/greetings-ci.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

**...or create a new repository on the command line**

```
echo "# greetings-ci" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/gwstudent/greetings-ci.git
git push -u origin main
```

**...or push an existing repository from the command line**

```
git remote add origin https://github.com/gwstudent/greetings-ci.git
git branch -M main
git push -u origin main
```

💡 **ProTip!** Use the URL for this page when adding GitHub as a remote.