

# CI/CD with GitHub Actions

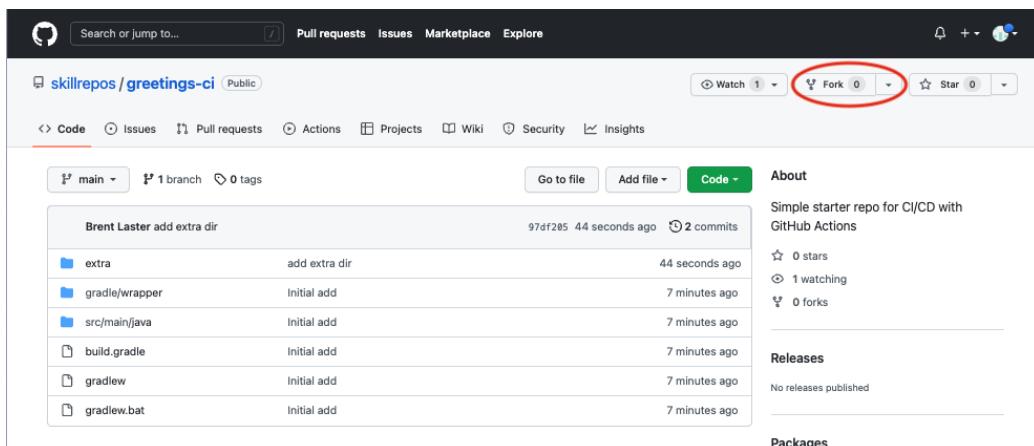
Revision 1.5 – 11/14/24

Tech Skills Transformations LLC / Brent Laster

## 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.



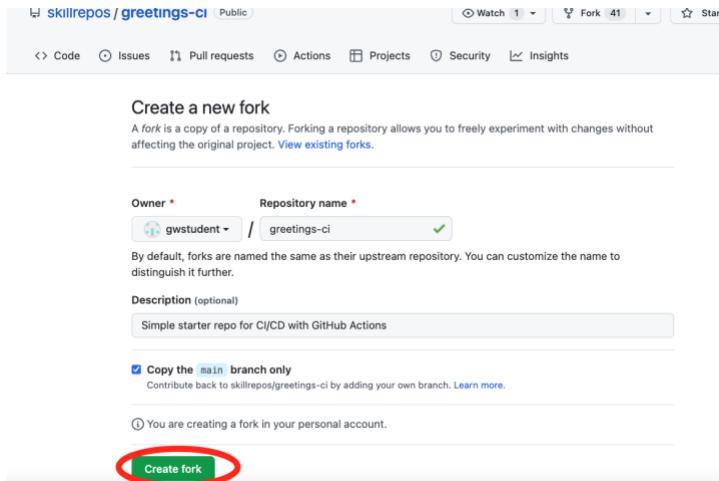
The screenshot shows the GitHub repository page for 'skillrepos/greetings-ci'. At the top, there is a 'Fork' button with a value of '0', which is circled in red. Below the header, there is a list of commits from 'Brent Laster':

Commit Message	Date	Author
add extra dir	97df205 44 seconds ago	Brent Laster
extra	44 seconds ago	
gradle/wrapper	Initial add	
src/main/java	Initial add	
build.gradle	Initial add	
gradlew	Initial add	
gradlew.bat	Initial add	

On the right side of the page, there is an 'About' section with the following details:

- Simple starter repo for CI/CD with GitHub Actions
- 0 stars
- 1 watching
- 0 forks

Below the 'About' section, there are sections for 'Releases' and 'Packages'.

The screenshot shows the 'Create a new fork' dialog for the 'greetings-ci' repository. At the bottom of the form, there is a large 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.

This screenshot shows a GitHub repository page for 'gwstudent/greetings-ci'. The 'Actions' tab is highlighted with a red circle. Below it, a list of recent commits is shown, all of which are related to setting up CI/CD with GitHub Actions.

- 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".

This screenshot shows the 'Categories' section of the GitHub Actions page. The 'Continuous integration' category is highlighted with a red circle. Other categories like Automation, Deployment, and Security are also listed.

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

This screenshot shows a GitHub repository page for 'gwstudent/greetings-ci'. The 'Actions' tab is selected. A red circle highlights the search bar at the top, which contains the text 'Gradle'.

This screenshot shows the 'Continuous integration' category page for GitHub Actions. A red circle highlights the search input field at the top, which contains the text 'Gradle'. Below the search bar, a list of workflows is displayed, including 'Android CI', 'Java with Ant', 'Clojure', and 'Publish Java Package'.

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



7. 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.)

The top screenshot shows the GitHub interface with the URL bar containing 'greetings-ci/.github/workflows/gradle.yml'. A cursor is positioned over the 'gradle.yml' part of the path. The bottom screenshot shows the same interface with the URL bar now containing 'greetings-ci/.github/workflows/pipeline.yml', indicating the file has been renamed.

8. 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 39 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.

greetings-ci/.github/workflows/pipeline.yml in main

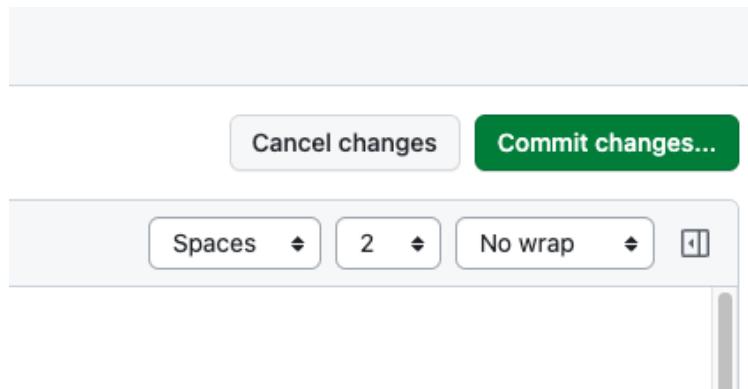
Edit Preview

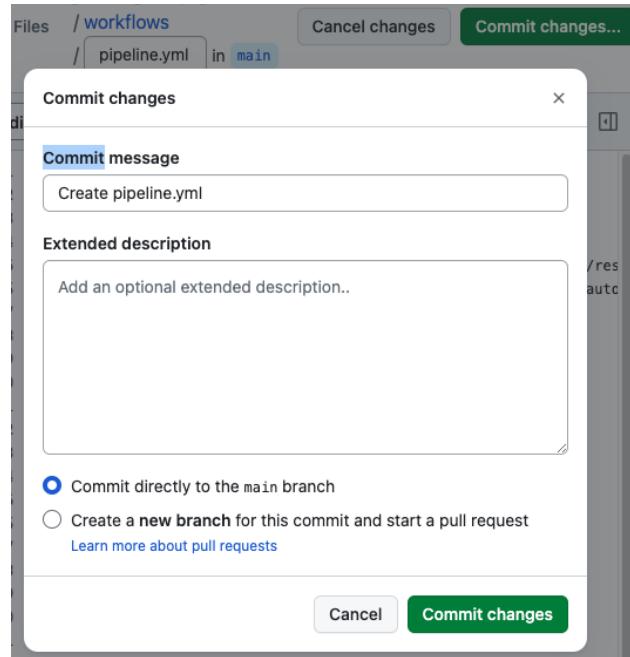
```

27   with:
28     java-version: '17'
29     distribution: 'temurin'
30
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   # 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@af1da67850ed9a4cedd57bfd976089dd991e2582 # v4.0.0
44   #   with:
45   #     gradle-version: '6.9'
46   #
47   # - name: Build with Gradle 8.0
48   #   run: gradle build
49
50   dependency-submission:
51
52   runs-on: ubuntu-latest
53   permissions: highlight/select and delete
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@af1da67850ed9a4cedd57bfd976089dd991e2582 # v4.0.0
68

```

- 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'. 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 is titled 'build' and shows a summary: 'succeeded 3 minutes ago in 25s'. Below this is a log viewer with a search bar and a refresh button. The log shows the following steps:

```
Set up job
Run actions/checkout@v3
  Run actions/checkout@v3
  Syncing repository: brentlaster/greetings-ci
  Getting Git version info
  Temporarily overriding HOME='/home/runner/work/_temp/77610fae-ac20-4aea-8803530bce6e' before making global git config changes
```

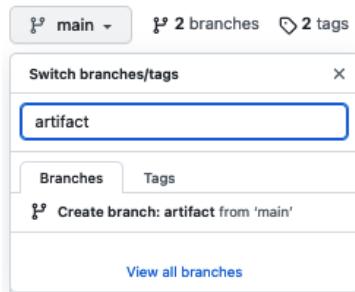
There are three red circles highlighting the 'build' job in the sidebar, the 'Run actions/checkout@v3' step in the log, and the first step 'Run actions/checkout@v3' in the log itself.

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 editor.

```
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 the GitHub Marketplace search results for "Upload build". The first result, "Upload a Build Artifact" by actions, is highlighted with a red circle. This action has 2.7k stars and is described as "Upload a build artifact that can be used by subsequent workflow steps". To the right, a second red circle highlights the "View full Marketplace listing" button. The listing page shows the action's details, including its version (v2.2.4), installation instructions (copying a snippet into a .yml file), and available options (warn, error, ignore).

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>.

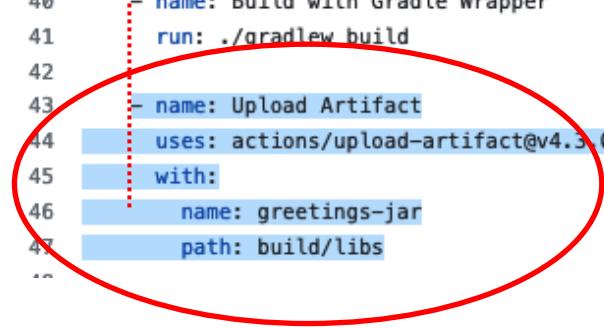
The screenshot shows the GitHub Marketplace listing for the "Upload a Build Artifact" action. The action is identified as a GitHub Action with version v3.1.0. It is described as "Upload-Artifact v3", which uploads artifacts from your workflow. The "What's new" section lists several improvements, including easier upload via wildcard patterns, support for individual files, directories, and multi-path uploads. The listing also indicates that the creator is verified, has 1.7k stars, and is associated with the "Utilities" category. There are links to pull requests, issues, and the GitHub repository.

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.)

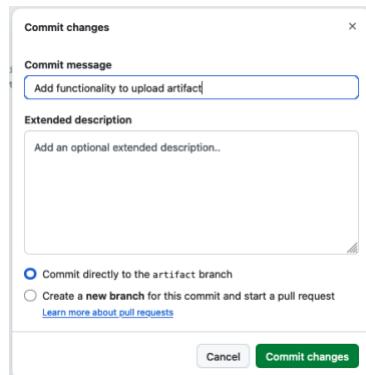
The code to add is immediately below. You can copy and paste but may need to adjust the indenting. Notice this should go after the *Build with Gradle Wrapper* step.

```
- 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" near the bottom of the page 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 has tabs for Code, Pull requests, Actions (which is highlighted), 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 specific run for "Add functionality to upload artifact" is expanded. It shows a "Summary" card with details like trigger (push), author (gwstudent2), status (Success), duration (32s), and artifacts (1). The "pipeline.yml" card shows the configuration for the workflow. The "build" step is shown with a duration of 20s. The "build summary" card provides a detailed view of the build process, showing a Gradle Root Project named "greetings-ci" with tasks "build", Gradle Version "4.10", and Build Outcome marked as successful. The "Artifacts" section at the bottom lists a single artifact named "greetings-jar" with a size of 1010 Bytes. This artifact is also circled in red.

The screenshot shows a CI/CD pipeline summary for a 'greetings-ci' project. The table includes columns for 'Gradle Root Project', 'Requested Tasks', 'Gradle Version', and 'Build Outcome'. The 'Requested Tasks' column shows 'build', 'Gradle Version' is '4.10', and the 'Build Outcome' is marked with a green checkmark. Below the table, a note states 'Caching for Gradle actions was read-only - expand for details'. A link 'Job summary generated at run-time' is provided. On the right, a list of artifacts is shown:

- greetings-jar(18).zip (Completed — 1.0 KB)
- greetings-jar(17).zip (Completed — 1.0 KB)
- cicd-custom-labs-1.pdf (Completed — 4.5 MB)
- cicd-custom-labs.pdf (Completed — 4.5 MB)
- greetings-jar(16).zip (Completed — 1.3 KB)

Links for 'Show all downloads' and 'Size' are available. At the bottom, there is a download button labeled 'Download greetings-jar'.

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

- 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. The URL in the address bar is https://github.com/gwstudent2/greetings-ci/compare/main...artifact?expand=1. The main heading is 'Open a pull request'. It says 'Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about creating pull requests'.

Branch selection dropdowns show 'base: main' and 'compare: artifact'. A note says 'Able to merge. These branches can be automatically merged.'

Form fields include 'Add a title' (placeholder 'Add functionality to upload artifact') and 'Add a description' (placeholder 'Add your description here...'). A rich text editor toolbar is above the description field. At the bottom, there is a '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

**gwstudent2 commented 5 minutes ago**

Closes #2

**Add functionality to upload artifact** Verified ✓ d9f7ea8

**Require approval from specific reviewers before merging**  
Rulesets ensure specific people approve pull requests before they're merged.

**All checks have passed** 1 successful check [Show all checks](#)

**All checks have passed** 1 successful check [Hide all checks](#)

**Java CI with Gradle / build (push)** Successful in 20s [Details](#)

**This branch has no conflicts with the base branch**  
Merging can be performed automatically.

**build** succeeded 2 hours ago in 20s

- > 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
- > Complete job

3. 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.

```

Add functionality to upload artifact #3
Edit | < Code
Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings
Open gwstudent2 wants to merge 1 commit into main from artifact
Conversation 0 | Commits 1 | Checks 1 | Files changed 1 | +8 -2
Changes from all commits - File filter - Conversations - Jump to - Review changes
. 11 .github/workflows/pipeline.yml
9 9 @@ -9,9 +9,9 @@ name: Java CI with Gradle
10 10 on:
11 11 push:
12 - branches: [ "main" ]
13 + branches: [ "artifact" ]
14 - branches: [ "main" ]
14 + branches: [ "artifact" ]
15 15 jobs:
16 16 build:
17 17
18 @@ -36,6 +36,13 @@ jobs:
36 36 - name: Build with Gradle Wrapper
37 37 run: ./gradlew build
38 38
39 + - name: Upload Artifact
40 + uses: actions/upload-artifact@v4.3.0
41 + with:

```

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.

Changes from all commits - File filter - Conversations - Jump to -

```

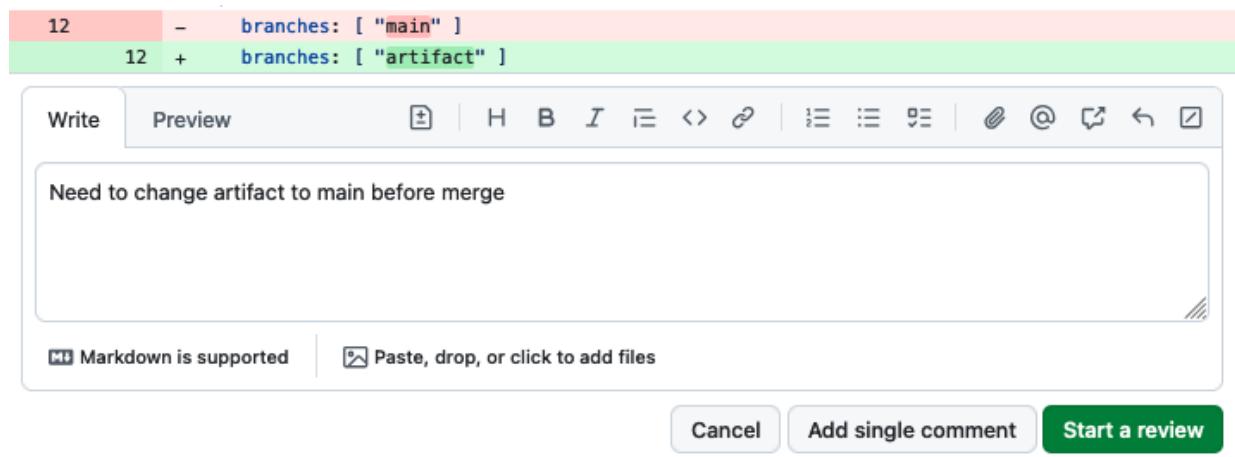
. 11 .github/workflows/pipeline.yml
@@ -9,9 +9,9 @@ name: Java CI with Gradle
9 9
10 10 on:
11 11 push:
12 - branches: [ "main" ]
12 + branches: [ "artifact" ]
13 13 pull_request:
14 - branches: [ "main" ]

```

**Leave a comment**

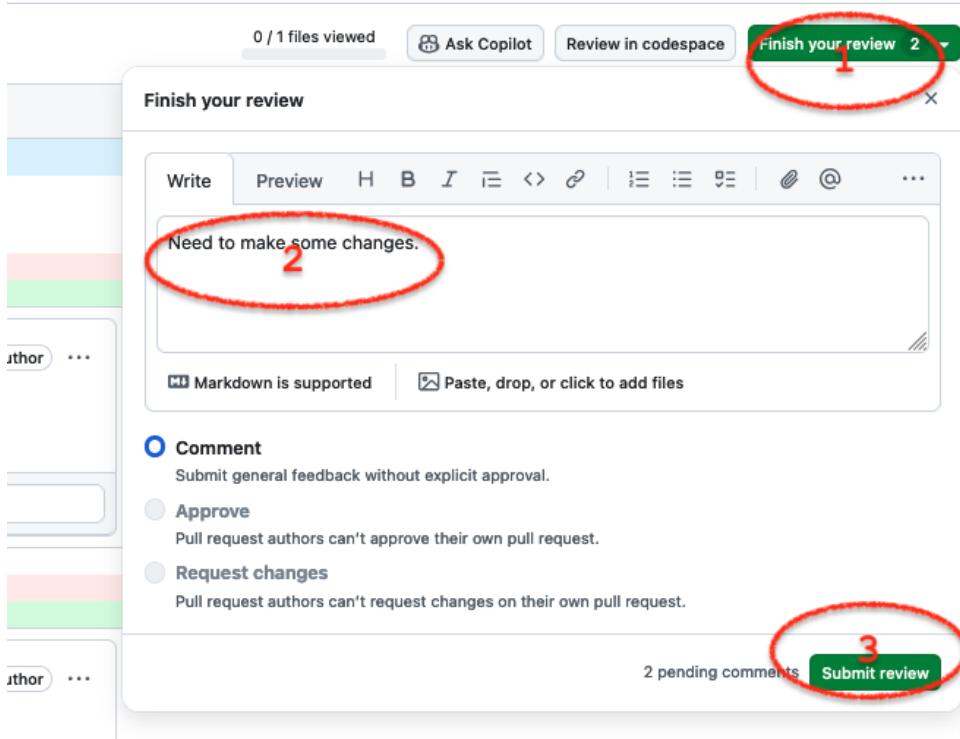
Markdown is supported | Paste, drop, or click to add files | Please fill out this field.

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



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

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>. You can do this in a different browser tab if you want.

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

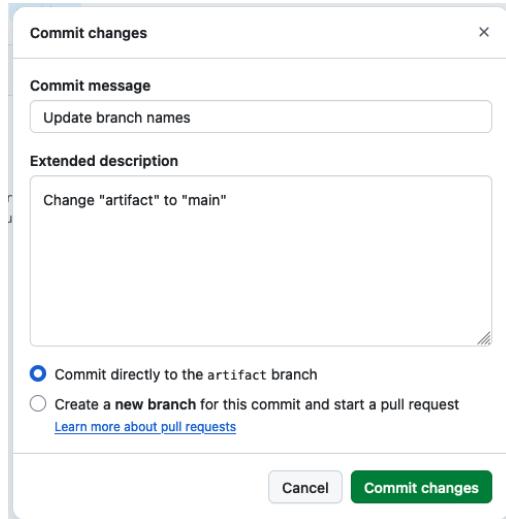
9. In the edit screen, change [ "artifact" ] to [ "main" ] in both lines.

```

<code>
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://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-gradle
7
8 name: Java CI with Gradle
9
10 on:
11   push:
12     branches: [ "main" ]
13   pull_request:
14     branches: [ "main" ]
</code>

```

10.. Click on the ***Commit changes...*** button in the upper right. In the dialog that comes up, you can change the commit message and 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 "1" with a different number if your pull request has a different number.)

<https://github.com/<github-userid>/greetings-ci/pull/1/commits>

<https://github.com/<github-userid>/greetings-ci/pull/1/files>

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

Workflow Run	Event	Status	Branch	Actor
Add functionality to upload artifact (Java CI with Gradle #3)	Pull request #3 synchronise	Completed	artifact	gestudent2
Add functionality to upload artifact (Java CI with Gradle #2)	Commit d5f7eaf pushed	Completed	artifact	gestudent2
Create pipeline.yml (Java CI with Gradle #1)	Commit f70bc3d pushed	Completed	main	gestudent2

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/1>

The screenshot shows the GitHub interface for a pull request. At the top, it says "Add functionality to upload artifact #3" and "gwstudent2 wants to merge 2 commits into `main` from `artifact`". Below this, there's a "Resolve conversation" button and a link to "Update pipeline.yml". A green "Verified" badge with the commit hash "7060c61" is shown. The main area displays review rules: "Require approval from specific reviewers before merging" (with a "Rulesets" link) and "All checks have passed" (1 successful check). It also states "This branch has no conflicts with the base branch". A red circle highlights the "Merge pull request" button. Below it, a note says "You can also open this in GitHub Desktop or view command line instructions." A confirmation dialog box is shown with the message "This commit will be authored by bl1@nclasters.org" and two buttons: "Confirm merge" (highlighted with a red circle) and "Cancel". Finally, a success message says "Pull request successfully merged and closed" and "You're all set—the `artifact` branch can be safely deleted.", with a "Delete branch" button highlighted with a red circle.

(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 repository "gwstudent2 / greetings-ci". The "Actions" tab is selected, showing the "All workflows" section. Under "All workflows", there is one entry: "Java CI with Gradle". The "All workflow runs" section shows four runs:

Run	Event	Status	Branch	Actor
Merge pull request #3 from gwstudent2/artifact	main	<span style="color: green;">Success</span>	main	46s
Add functionality to upload artifact	artifact	<span style="color: green;">Success</span>	artifact	24s
Add functionality to upload artifact	artifact	<span style="color: green;">Success</span>	artifact	32s
Create pipeline.yml	main	<span style="color: green;">Success</span>	main	53s

## 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*.

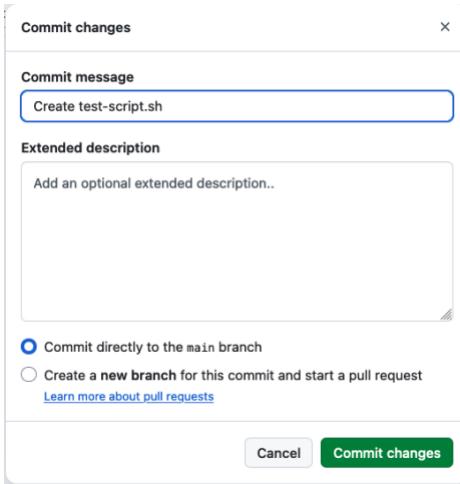
2. In the new editor that pops up, you'll be at the location to type in a name. You can name this "test-script.sh". Then copy and paste the following code into the new file. (A screenshot is shown after the code so you can see how things line up.) (This code is also available at <https://gist.github.com/brentlaster/f1c922ff4266882f0e5f2982da053cde>)

```
# 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
```

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: >
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:

```

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

```

- 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>)

7. Commit the changes as before ( to the "main" branch using the "Commit changes..." button).
8. 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 got executed.

Note: If you have any errors, go back and check the code, especially the alignment.

The screenshot shows the GitHub Actions pipeline summary for a workflow named "Update pipeline.yml #10". The pipeline has two jobs: "build" and "test-run". The "build" job completed successfully in 20 seconds. The "test-run" job completed successfully in 4 seconds. The total duration of the run was 41 seconds. There is one artifact produced.

The screenshot shows the GitHub Actions test-run logs for a workflow named "Update pipeline.yml #16". The logs detail the execution of the "Execute test" step, which succeeded in 0 seconds. The log output shows the command being run: `chmod +x ./test-script.sh && ./test-script.sh 0.0.16 1 2 3`. The logs also show the setup of the JDK and the download of candidate artifacts.

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

Status: Success Total duration: 41s Artifacts: 1

**pipeline.yml**  
on: push

**build** 20s → **test-run** 4s

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

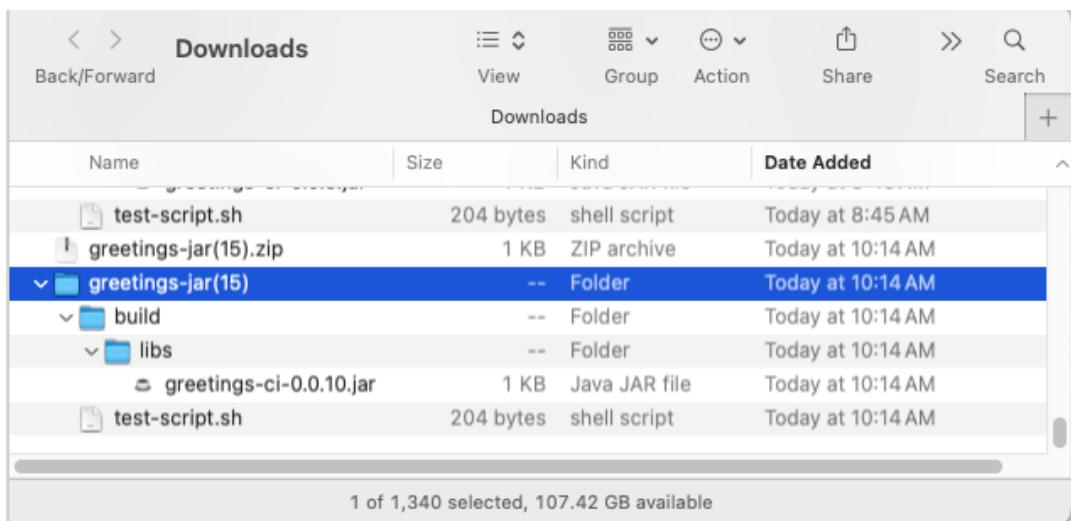
Job summary generated at run-time

**Artifacts**  
Produced during runtime

Name	Size
greetings-jar	1.29 KB

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

- . 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.

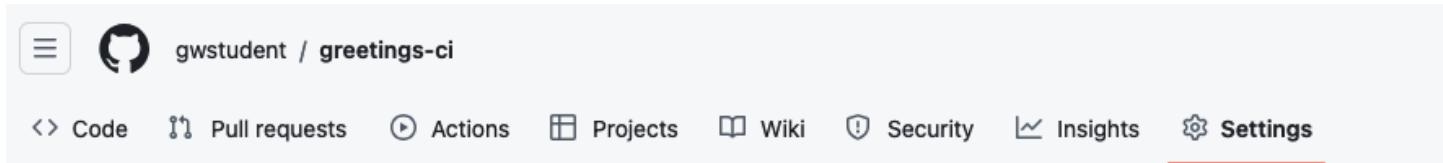


## Lab 5: Working with fast feedback and automatically reporting issues

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.

Go to the *Settings* tab for your repository or to <https://github.com/<github-userid>/greetings-ci/settings>.



Then scroll down on the *Settings* page, find the "Features" section and check the box to enable use of *Issues* on this repository.

The screenshot shows the 'Features' section of the GitHub repository settings. It includes checkboxes for Wikis, Restrict editing to collaborators only, Issues (which is checked and circled in red), and Sponsorships.

Feature	Status	Description
Wikis	<input checked="" type="checkbox"/>	Wikis host documentation for your repository.
Restrict editing to collaborators only	<input checked="" type="checkbox"/>	Public wikis will still be readable by everyone.
Issues	<input checked="" type="checkbox"/>	Issues integrate lightweight task tracking into your repository. Keep projects on track with issue labels and milestones, and reference them in commit messages.
Sponsorships	<input type="checkbox"/>	Sponsorships help your community know how to financially support this 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 into the *.github/workflows* directory so it can be used as a workflow. You can just move it via a rename in GitHub with the following steps.

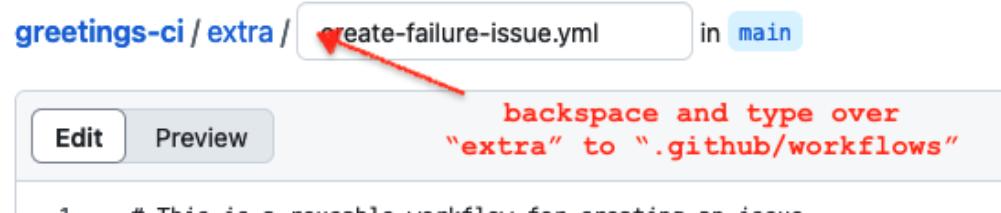
- a. In the repository, browse to the "extra" folder and open the "create-failure-issue.yml" file.
- b. Take a few moments to look over the file and see what it does. Notice that:
  - i. It has a *workflow\_call* section in the "on" area, which means it can be run from another workflow.
  - ii. It has 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.

A screenshot of a GitHub repository page. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main content area shows a file named "greetings-ci / extra / create-failure-issue.yml". A commit message from "brentlaster" is visible, stating "Update create-failure-issue.yml to be a reusable workflow". The file content is a YAML workflow definition. In the top right corner of the code editor, there is a toolbar with several icons, one of which is a pencil icon for editing, which is circled in red.

```
1 # This is a reusable workflow for creating an issue
2
3 name: create-failure-issue
4
5 # Controls when the workflow will run
6
```

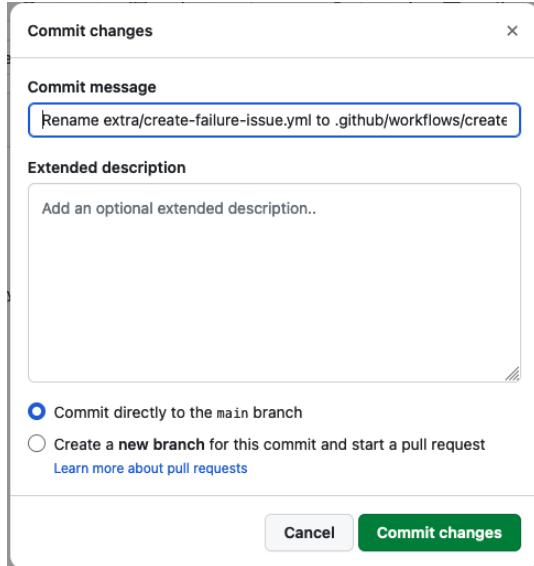
- d. In the filename field at the top, change the name of file. Use the backspace key to backspace over "extra/" making sure to backspace over the word. Then type in the path to put it in the workflows ".github/workflows/create-failure-issue.yml".



A screenshot of the GitHub code editor interface after renaming the file. The URL in the address bar is now "greetings-ci / .github / workflows / create-failure-issue.yml". The file content remains the same as the previous screenshot.

```
1 # This is a reusable workflow for creating an issue
2
3 name: create-failure-issue
4
```

- 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.

- 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.

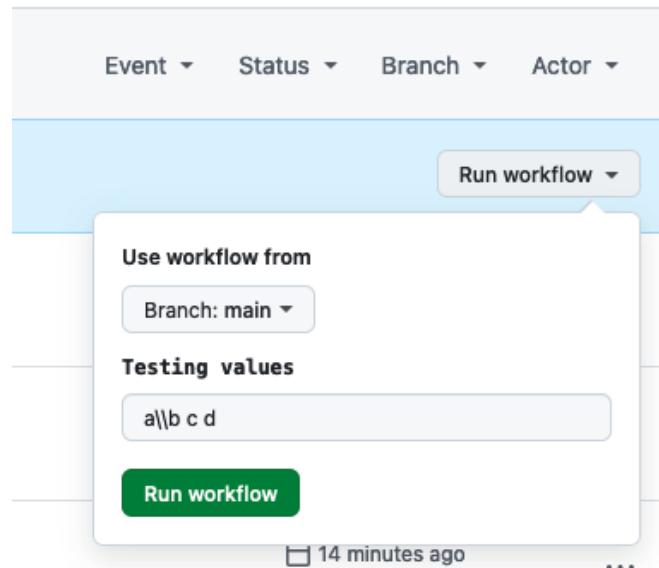
A blue bar will show up with a "Run workflow" button to the right. Click on the "Run workflow" button.

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 labeled '1'), Projects, Wiki, Security, Insights, and Settings. The left sidebar under the 'Actions' tab lists 'All workflows': 'create-failure-issue', 'Java CI with Gradle' (which is highlighted with a red circle labeled '2'), 'Management', 'Caches', 'Attestations', and 'Runners'. The main content area displays the 'Java CI with Gradle' workflow details. It shows '28 workflow runs' and a note: 'This workflow has a workflow\_dispatch event trigger.' A 'Run workflow' button is highlighted with a red circle labeled '3'.

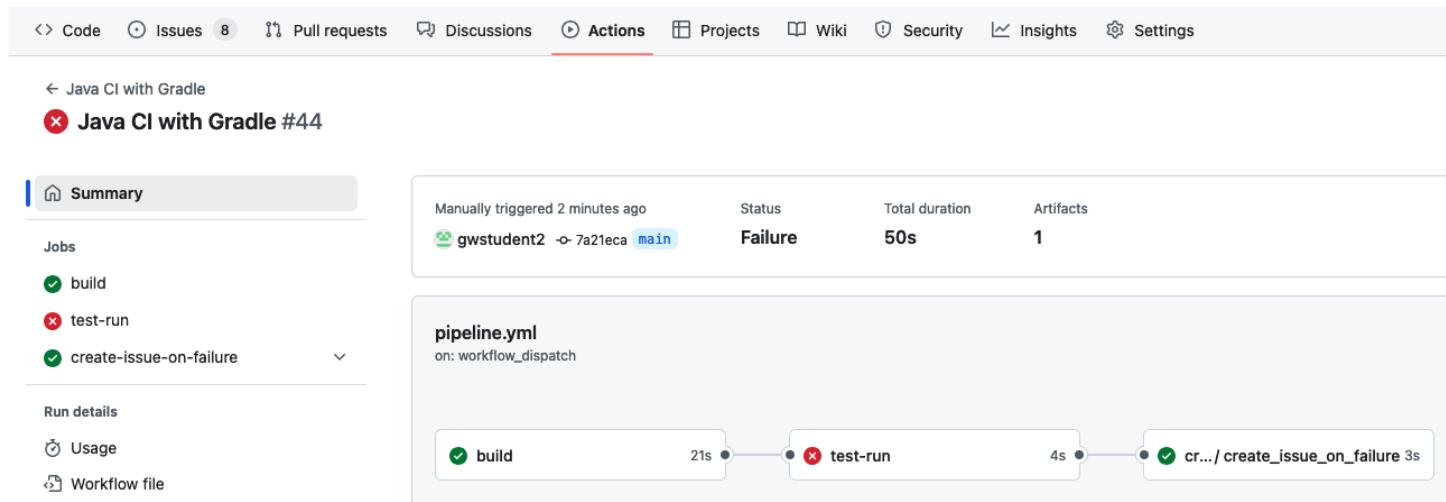
9. For the test argument values we'll put in ones that will cause an issue so that our automatic issue creation will be triggered. To do this, we'll include two backslashes in the input (they can be anywhere, but will cause the test job to fail, and thus cause an issue to be generated).

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: 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.)

```

greetings-ci / .github / workflows / pipeline.yml in main
Edit Preview Spaces 2 No wrap
87   uses: ./github/workflows/create-failure-issue.yml
88   with:
89     title: "Automated workflow failure issue for commit ${{ github.sha }}"
90     body: >
91       "This issue was automatically created by the workflow ${{ github.workflow }}"
92
93   deploy-stage:
94
95     needs: [build, test-run]
96     if: github.ref == 'refs/heads/blue' || github.ref == 'refs/heads/green'
97
98     runs-on: ubuntu-latest
99     environment:
100       name: staging
101       url: https://github.com/${{ github.repository }}/releases/tag/v0.0.${{ github.run_number }}
102     steps:
103
104       - uses: actions/checkout@v4
105         with:
106           fetch-depth: 0
107
108       - name: Download candidate artifacts
109         uses: actions/download-artifact@v4
110         with:
111           name: greetings-jar
112
113       - name: GH Release
114         uses: softprops/action-gh-release@v2
115         with:
116           tag_name: v0.0.${{ github.run_number }}
117           generate_release_notes: true
118           name: ${{ github.ref_name }}
119           files: |
120             build/libs/greetings-ci-0.0.${{ github.run_number }}.jar

```

5. Now, let's add the job for deploying a "prod" (production) environment/release from a pull-request being merged into "main". 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.)

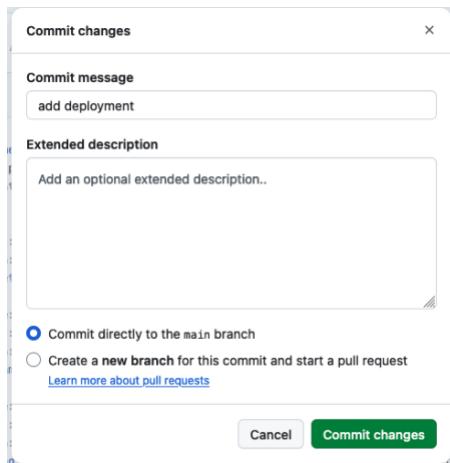
```

greetings-ci / .github / workflows / pipeline.yml in main
Edit Preview Spaces 2 No wrap □

109     uses: actions/download-artifact@v4
110     with:
111         name: greetings-jar
112
113     - name: GH Release
114         uses: softprops/action-gh-release@v2
115         with:
116             tag_name: v0.0.${{ github.run_number }}
117             generate_release_notes: true
118             name: ${{ github.ref_name }}
119             files: |
120                 build/libs/greetings-ci-0.0.${{ github.run_number }}.jar
121
122
123     deploy-prod:
124
125     needs: [build, test-run]
126     if: github.ref == 'refs/heads/main'
127
128     runs-on: ubuntu-latest
129     environment:
130         name: production
131         url: https://github.com/${{ github.repository }}/releases/tag/v1.0.${{ github.run_number }}
132     steps:
133
134         - uses: actions/checkout@v4
135             with:
136                 fetch-depth: 0
137
138         - name: Download candidate artifacts
139             uses: actions/download-artifact@v4
140             with:
141                 name: greetings-jar

```

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.

Triggered via push 13 minutes ago

Status: Success Total duration: 1m 16s Artifacts: 1

**pipeline.yml**  
on: push

```

graph LR
    build[build] -- 37s --> testRun[test-run]
    testRun -- 7s --> deployProd[deploy-prod]
    testRun -- 7s --> deployStage[deploy-stage]
    testRun -- 7s --> createIssue[create-issue-on-failure]
    
```

<https://github.com/gwstudent/greetings...>

Releases / v1.0.10

**Production** Latest

github-actions released this 15 minutes ago v1.0.10 -> 055429b

**What's Changed**

- Add functionality to upload artifact by [@gwstudent](#) in #1

**New Contributors**

- [@gwstudent](#) made their first contribution in #1

Full Changelog: <https://github.com/gwstudent/greetings-ci/releases/v1.0.10>

**Contributors**

gwstudent

**Assets** 3

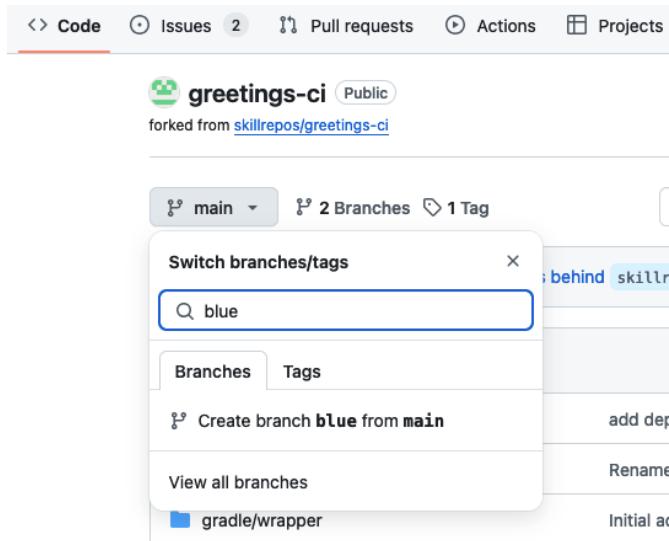
	Size	Uploaded
<a href="#">greetings-ci-1.0.10.jar</a>	1007 Bytes	15 minutes ago
<a href="#">Source code (zip)</a>		16 minutes ago
<a href="#">Source code (tar.gz)</a>		16 minutes ago

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.

- For 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. Make sure "main" shows up and you are on the "branches" tab, not the "tags" tab.)

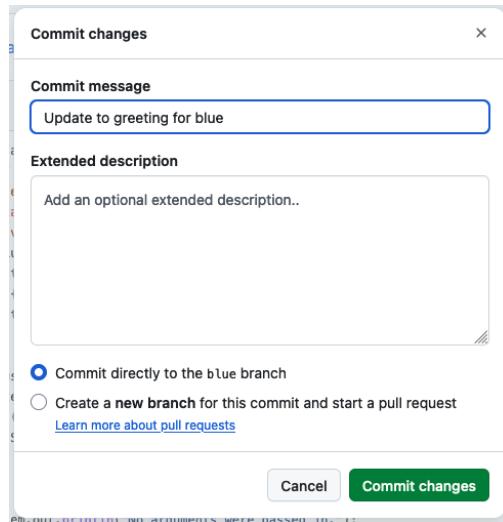


- Make sure you are in the "blue" branch, and edit the file "src/main/java/echoMsg.java". Make a simple, non-breaking change like adding "blue" to the lines that print out "Greetings". See text and figure below.

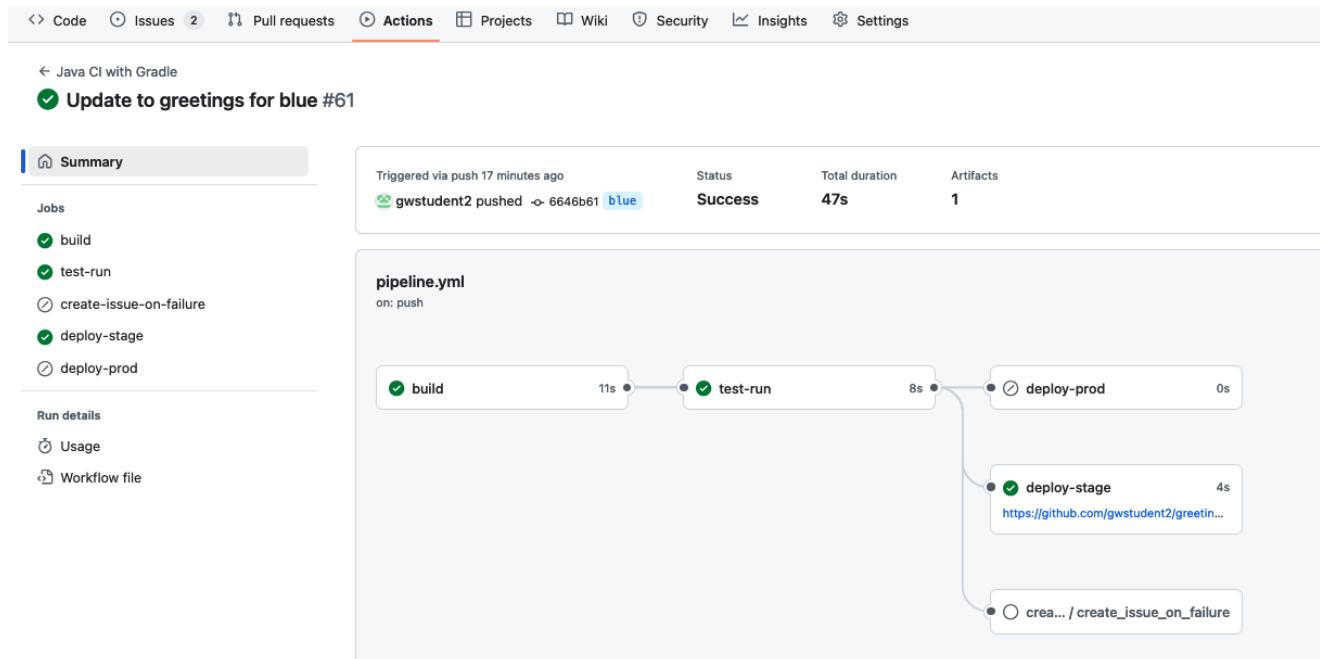
```
if (value != null) {  
    System.out.format("Greetings blue %s!\n",value);  
} else {  
    System.out.println("Greetings blue!");  
}
```

```
1  /* Echo Message Java Program */  
2  /* 8-31-21 */  
3  public class echoMsg {  
4      public static void main(String[] args) {  
5          String value = System.getenv("USERNAME");  
6          if (value != null) {  
7              System.out.format("Greetings blue %s!\n",value);  
8          } else {  
9              System.out.println("Greetings blue!");  
10         }  
11     }  
12     if (args.length > 0) {
```

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 staging release page for the *blue* deployment.

The screenshot shows a GitHub release page for the repository 'blue'. At the top, there's a navigation bar with links like 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. Below the navigation, it says 'Releases / v0.0.61'. The main content area is titled 'blue' with a 'Latest' badge. It shows a deployment to 'staging' by 'github-actions' 15 minutes ago, with version 'v0.0.61' and commit hash '25f5ad8'. A link to the 'Full Changelog' is provided. Under the 'Assets' section, there are three items: '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's also a small placeholder icon for a missing asset.

6. 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.

This screenshot shows the main repository page for 'gwstudent2/greetings-ci'. On the left, there's a sidebar with sections for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The 'Code' tab is selected. In the center, it says 'No packages published' with a link to 'Publish your first package'. Below that is a 'Deployments' section with 14 entries. The first two are 'staging' and 'production' from 19 and 34 minutes ago respectively. There's also a link to '+ 12 deployments'. At the bottom, there's a 'Languages' section.

This screenshot shows the 'Deployments' page specifically for the 'staging' environment. The left sidebar has 'All deployments' under 'Environments' and 'staging' selected. The main area is titled 'staging deployments' and shows 'Latest deployments'. It lists four recent staging deployments: one from 21 minutes ago, one from 23 minutes ago, one from 47 minutes ago, and one from 49 minutes ago. Each deployment entry includes a link to its details page.

7. 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>

The screenshot shows a GitHub interface for comparing branches. At the top, the URL is https://github.com/gwstudent2/greetings-ci/compare/main...blue. The repository name is gwstudent2 / greetings-ci. Below the URL, there are navigation links for Code, Issues (2), Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Code' link is underlined, indicating the active tab.

**Comparing changes**

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks or learn more about diff comparisons.

base: main ▾ ← compare: blue ▾ ✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. [Learn about pull requests](#) [Create pull request](#)

-o 1 commit 1 file changed 1 contributor

Commits on Nov 9, 2024

8. 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](#)

base: main ▾ ← compare: blue ▾ ✓ Able to merge. These branches can be automatically merged.

**Add a title**

Update to greetings for blue

**Add a description**

Write Preview

Add your description here...

Markdown is supported Paste, drop, or click to add files

Create pull request

9. 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.

10. 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.

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

The screenshot shows a GitHub release page for the repository 'gwstudent2 / greetings-ci'. The URL is https://github.com/gwstudent2/greetings-ci/releases/tag/v1.0.62. The 'Production' tab is selected, showing the latest release v1.0.62. The page includes sections for 'What's Changed' (listing an update to greetings for blue), 'Contributors' (listing gwstudent2), and 'Assets' (listing three files: greetings-ci-1.0.62.jar, Source code (zip), and Source code (tar.gz)).

END OF LAB

THANKS!

## 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**

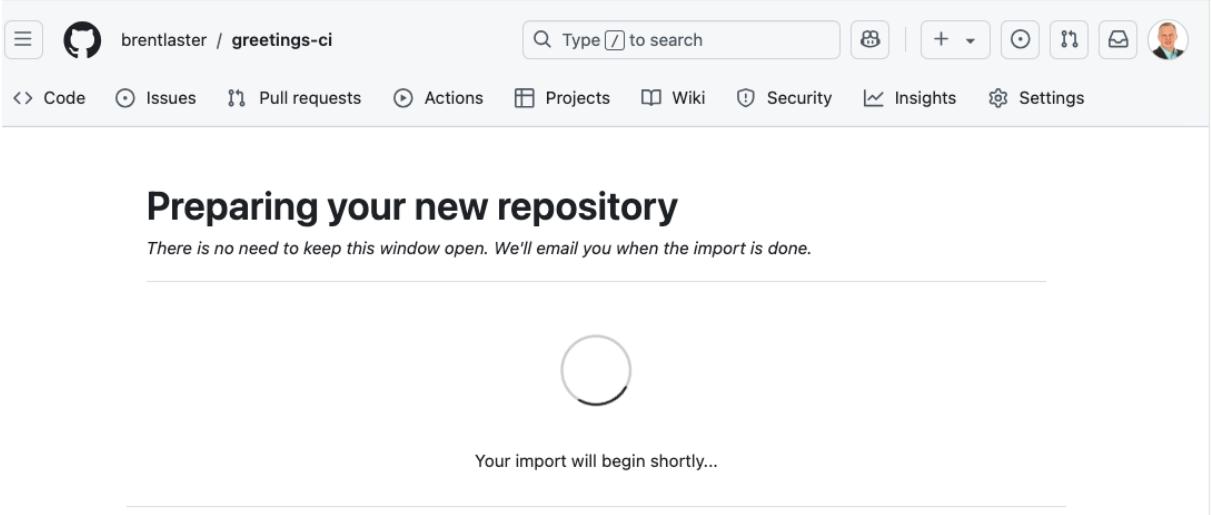
in the "Repository name \*" field.

The visibility field should be set to "Public".

Then click on the green "Begin import" button.

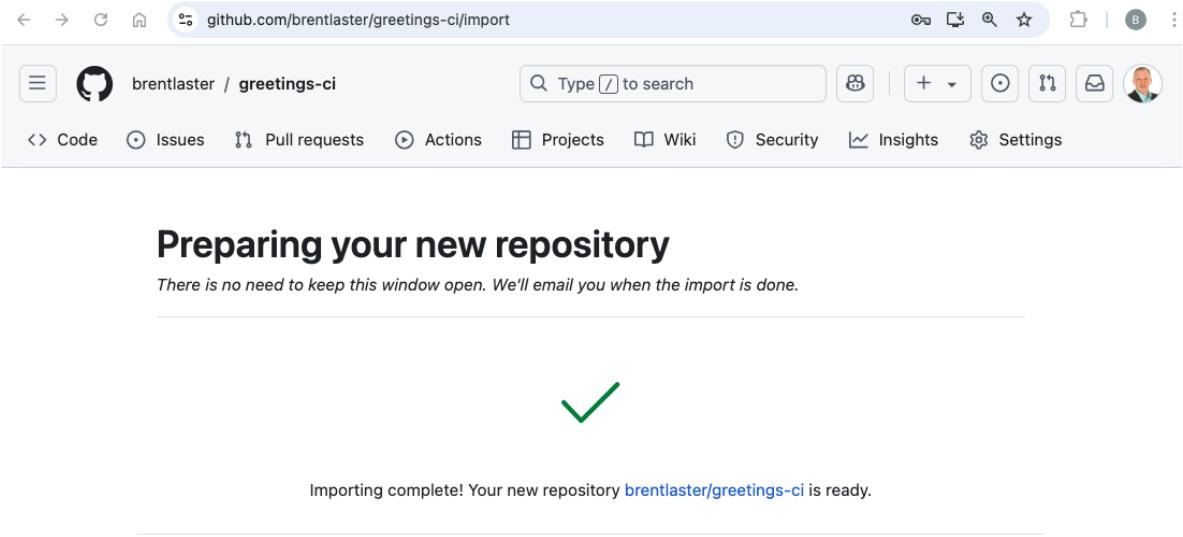
The screenshot shows the GitHub 'Import' page. At the top, there's a message: 'ended on April 12, 2024. For more details, see the [changelog](#)'. Below this, the 'Your source repository details' section has a red box around the 'The URL for your source repository \*' input field, which contains 'https://github.com/skillrepos/greetings-ci'. 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 import progress page. The URL in the address bar is `github.com/brentlaster/greetings-ci/import`. The page title is "Preparing your new repository". A sub-header says "There is no need to keep this window open. We'll email you when the import is done." Below this, there is a large circular progress indicator with a smaller circle inside it. Underneath the indicator, the text "Your import will begin shortly..." is displayed.

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



A screenshot of a GitHub repository import complete page. The URL in the address bar is `github.com/brentlaster/greetings-ci/import`. The page title is "Preparing your new repository". A sub-header says "There is no need to keep this window open. We'll email you when the import is done." Below this, there is a large green checkmark icon. Underneath the checkmark, the text "Importing complete! Your new repository [brentlaster/greetings-ci](#) is ready." is displayed.

## OPTION 2: 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 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

© 2024 Tech Skills Transformations, LLC & Brent Laster

git clone [git@github.com:skillrepos/greetings-ci](https://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 header with a back arrow, a refresh icon, and a search bar. Below it, a navigation bar includes a user icon, 'New repository', a search bar with placeholder 'Type / to search', and various icons for issues, pull requests, and more. The main form has a title 'Create a new repository' and a subtitle explaining what a repository is. It asks for 'Required fields are marked with an asterisk (\*).'. Under 'Repository template', a dropdown says 'No template'. Below it, a note says 'Start your repository with a template repository's contents.' The 'Owner \*' section shows 'gwstudent' and a repository name input field containing 'greetings-ci' (which is circled in red and labeled '1'). A note below says 'greetings-ci is available.' To the right of the owner field is a large red box containing the number '1'. The 'Description (optional)' field is empty. Below that, there are two radio buttons: 'Public' (selected) and 'Private'. A note under 'Public' says 'Anyone on the internet can see this repository. You choose who can commit.' A note under 'Private' says 'You choose who can see and commit to this repository.' The next section, 'Initialize this repository with:', has a checkbox for 'Add a README file' which is unchecked. A note says 'This is where you can write a long description for your project. [Learn more about READMEs.](#)' The 'Add .gitignore' section has a dropdown set to 'None'. A note says 'Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)' The 'Choose a license' section has a dropdown set to 'None'. A note says 'A license tells others what they can and can't do with your code. [Learn more about licenses.](#)' At the bottom, a note says 'You are creating a public repository in your personal account.' A large red box containing the number '2' is positioned to the left of the 'Create repository' button, which is highlighted with a red oval.

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'. At the top, there's a search bar and navigation links for Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the header, there's a 'greetings-ci' repository card with a 'Create a codespace' button. To the right, there's a 'Add collaborators' section with an 'Invite collaborators' button.

**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>

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.