

# CI/CD and DevOps in 3 Weeks

## Week 1

Revision 1.9 – 02/03/24

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'. The 'Fork' button is circled in red. Below the repository details, the 'Create a new fork' form is visible, with the 'Create fork' button also circled in red.

**Create a new fork**

A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Owner \*  Repository name \*

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

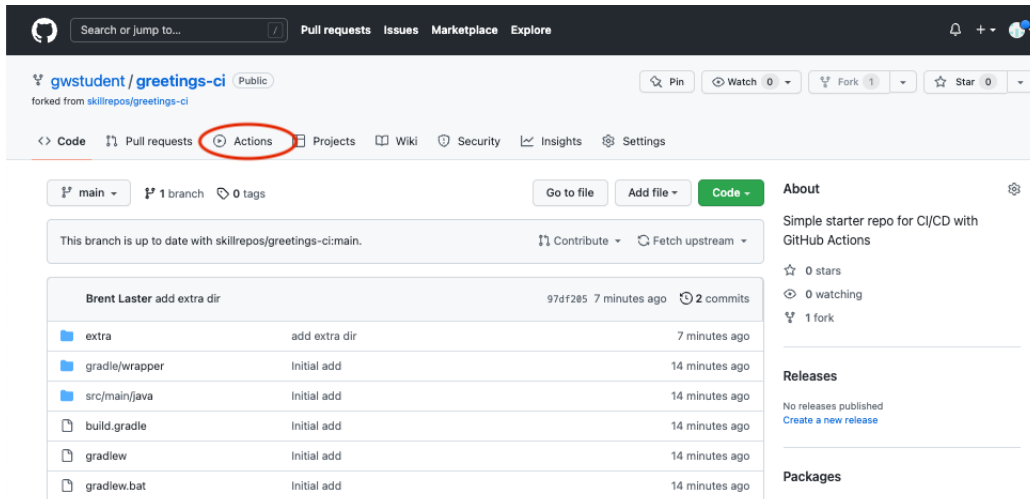
Description (optional)

☒ Copy the `main` branch only

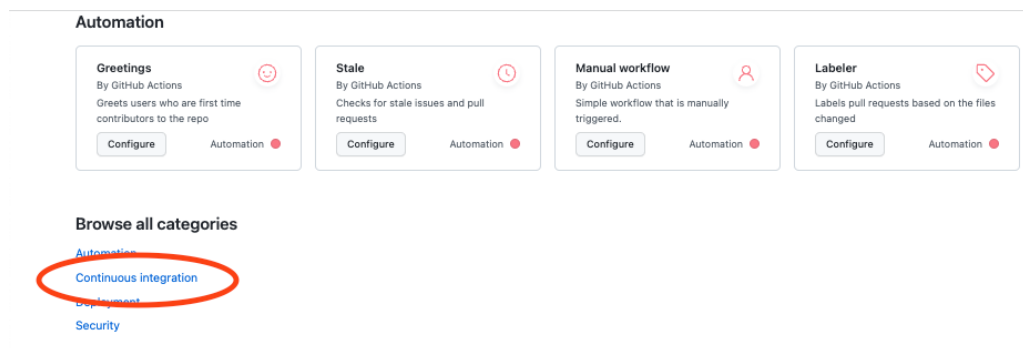
Contribute back to skillrepos/greetings-ci by adding your own branch. [Learn more.](#)

☐ You are creating a fork in your personal account.

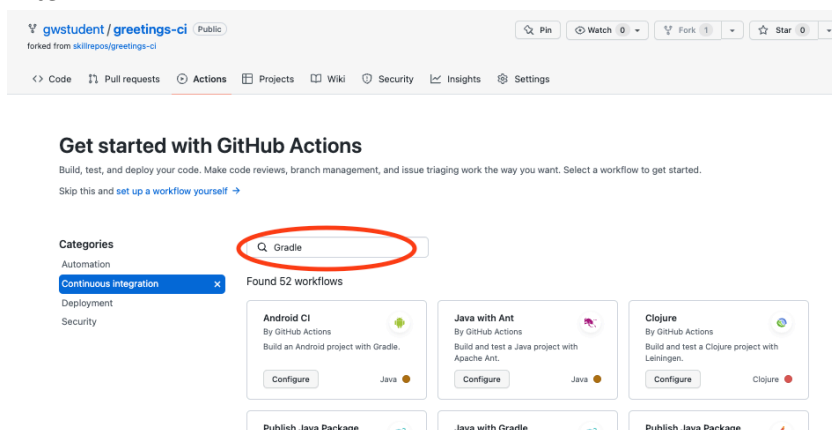
3. We have a simple java source file named *echoMsg.java* in the subdirectory *src/main/java*, a Gradle build file in the root directory named *build.gradle*, and some other supporting files. We could clone this repository and build it manually via running Gradle locally. But let's set this to build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



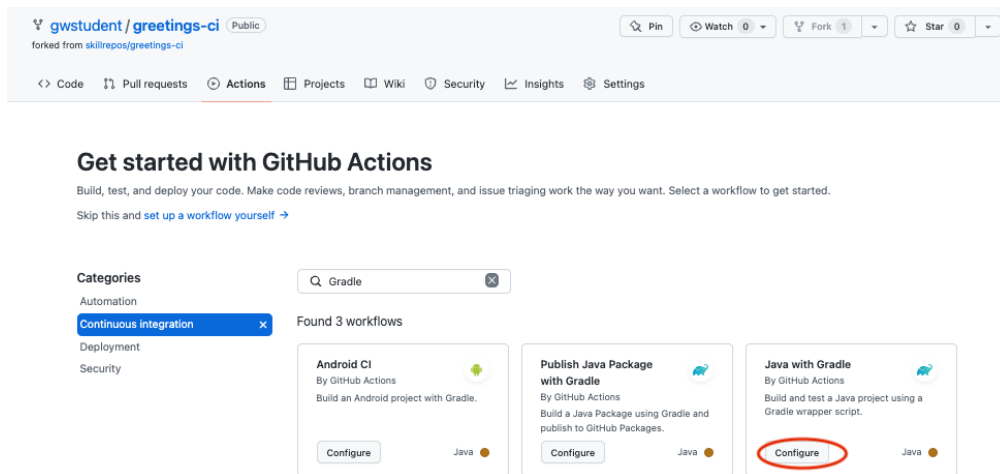
4. This will bring up a page with categories of starter actions that GitHub thinks might work based on the contents of the repository. We'll select a specific CI one. Scroll down to near the bottom of the page under "Browse all categories" and select "Continuous integration".



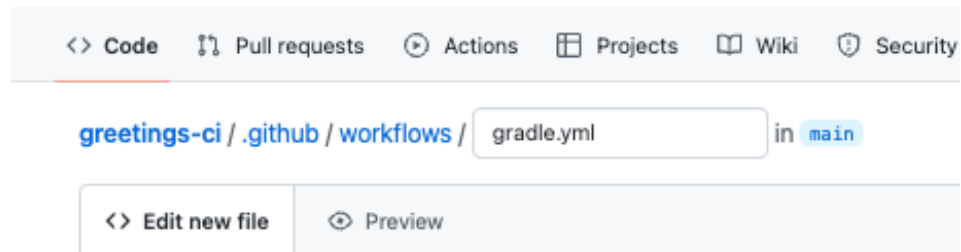
5. In the CI category page, let's search for one that will work with Gradle. Type "Gradle" in the search box and press Enter.



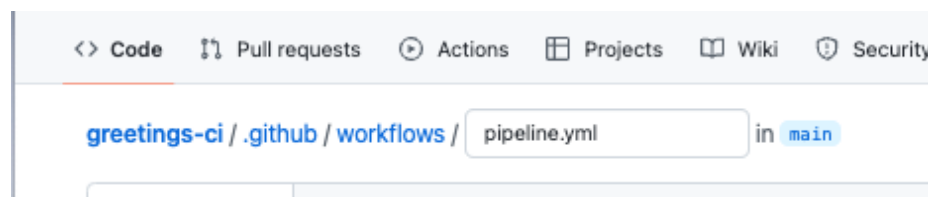
6. From the results, select the “Java with Gradle” one and click the “Configure” button to open a predefined workflow for this.



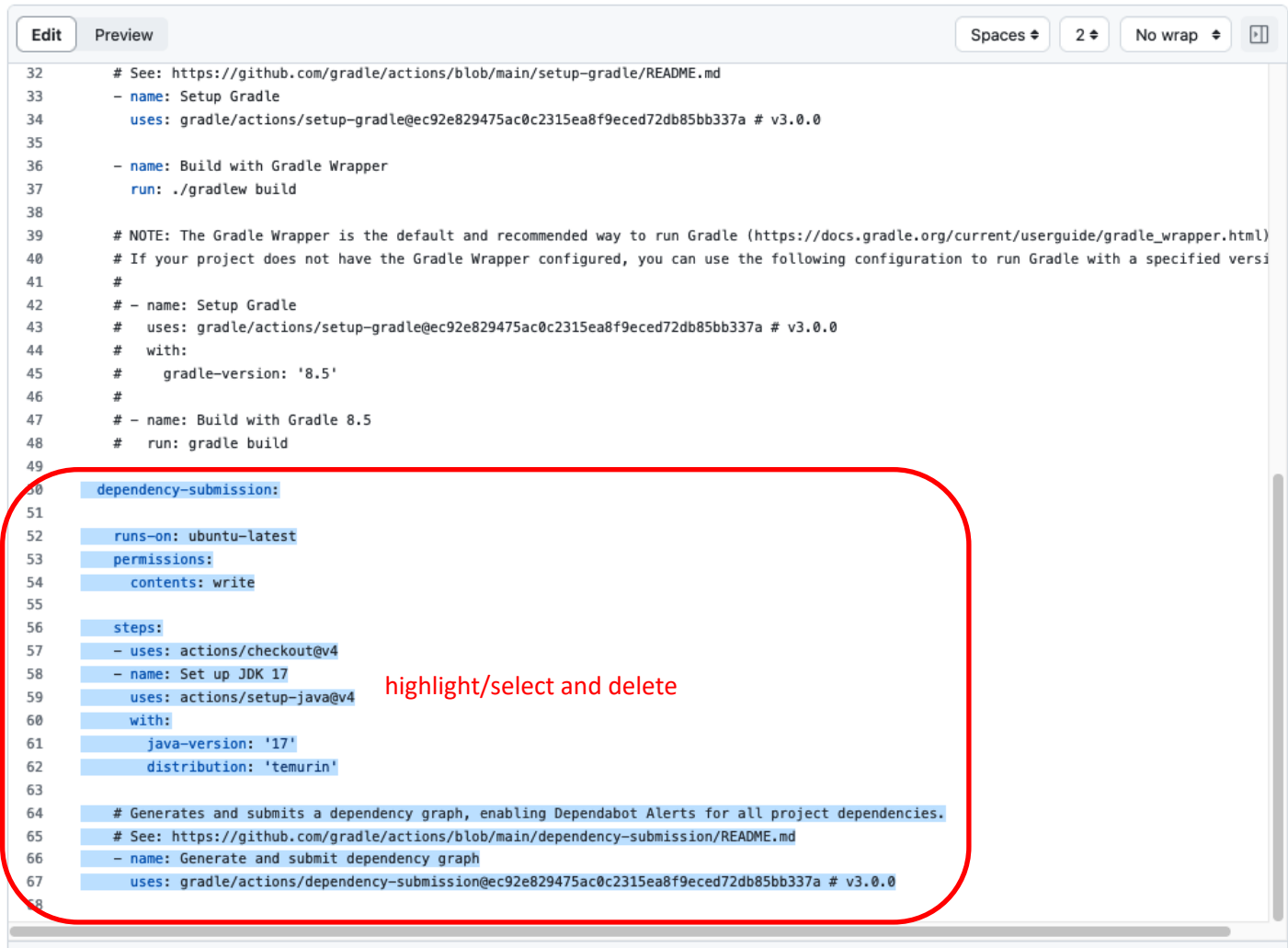
7. This will bring up a page with a starter workflow for CI that we can edit as needed. There are two edits we want to make here. The first is to change the name. In the top section where the path is, notice that there is a text entry box around “gradle.yml”. This is the current name of the workflow. 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.)



TO



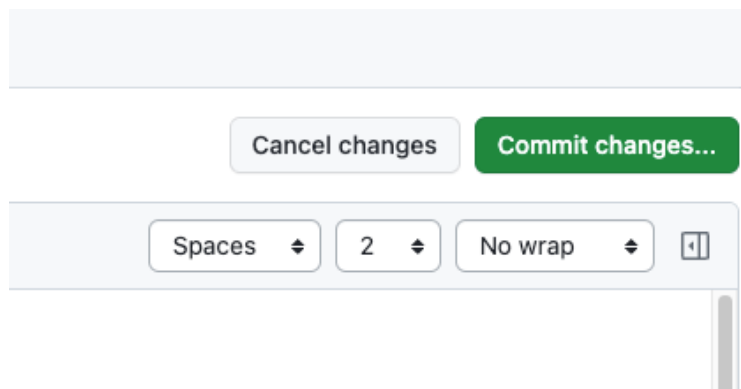
8. The second edit is to remove the second job in this workflow since it currently has issues. To do this we will just highlight/select the code from line 50 on and hit delete. (If you have trouble just selecting that code, try starting at the bottom and selecting/highlighting from the bottom up.) The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



```
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
```

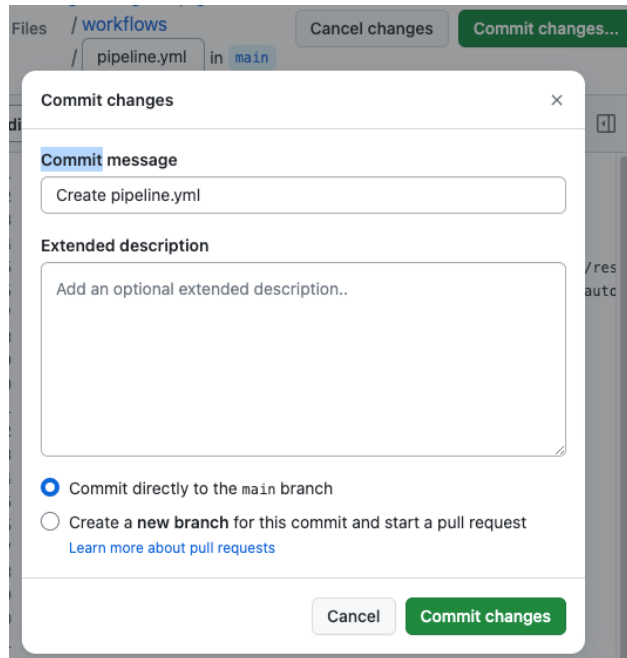
highlight/select and delete

9. 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.

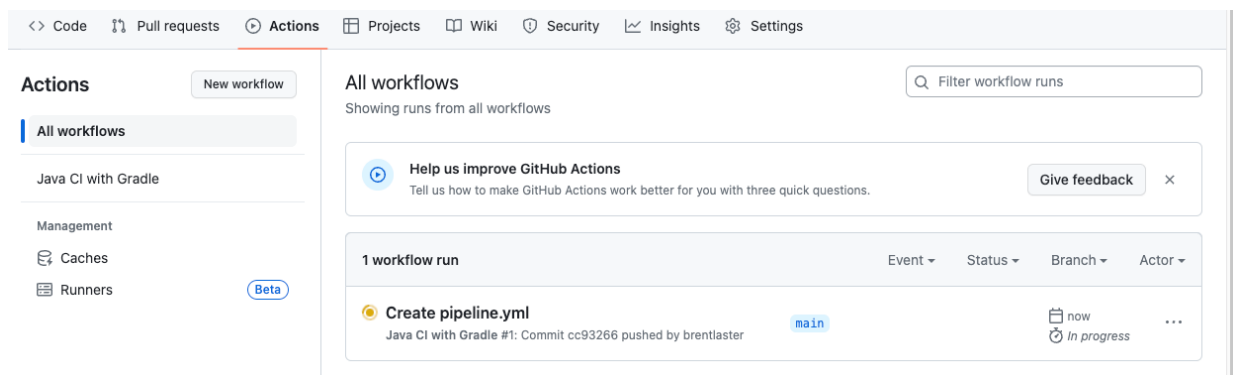


Cancel changes Commit changes...

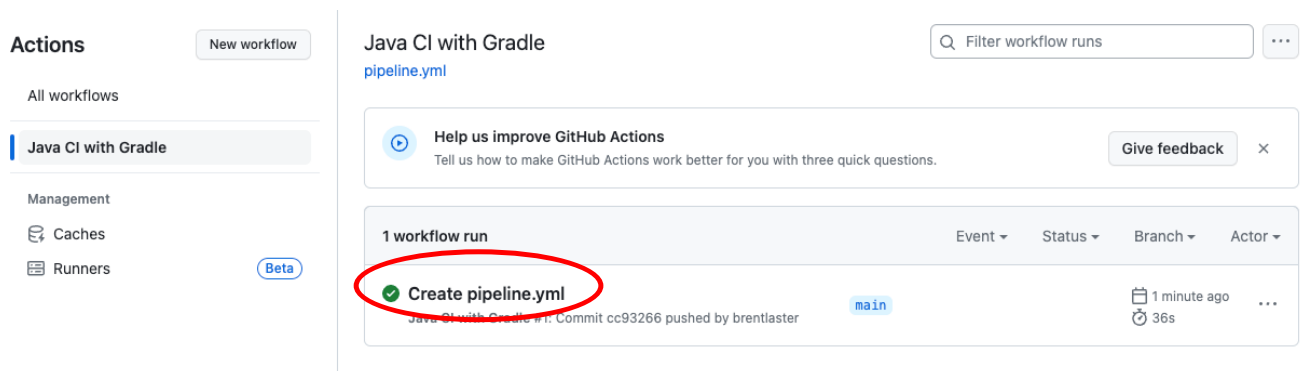
Spaces 2 No wrap



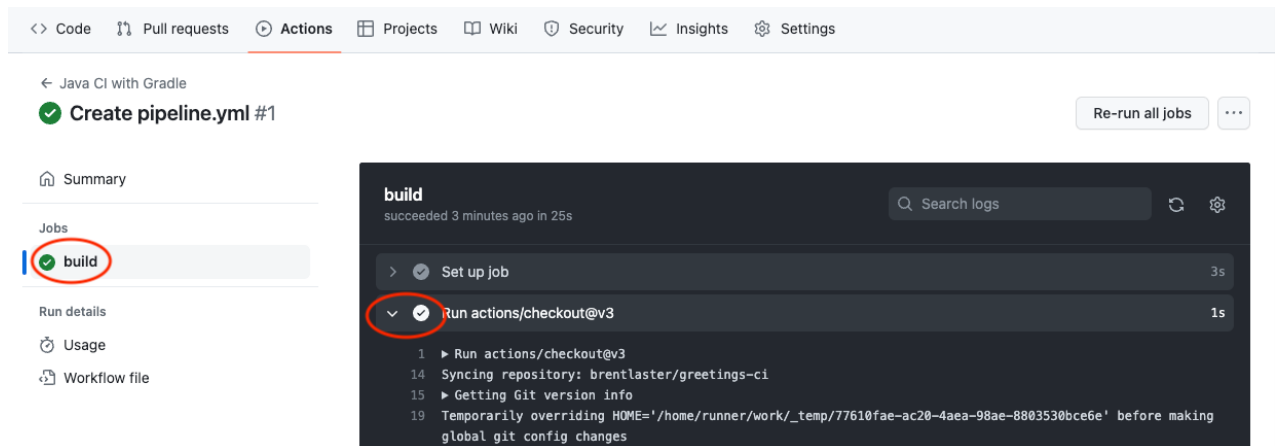
10. Since we've committed a new file and this workflow is now in place, the "on: push:" event is triggered and the CI automation kicks in. Click on the Actions menu again to see the automated processing happening.



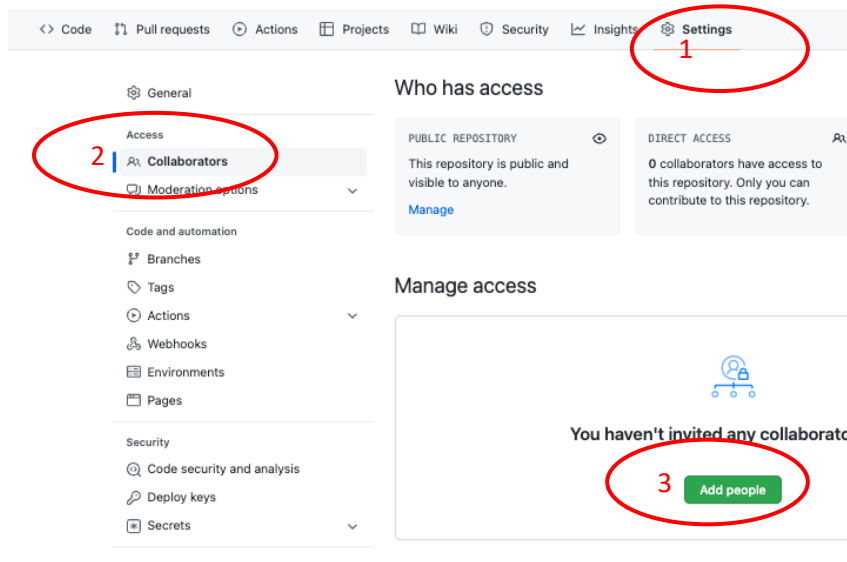
10. After a few moments, the workflow should succeed. (You may need to refresh your browser.) After it is done, you can click on the commit message for the run to get to the details for that particular run.



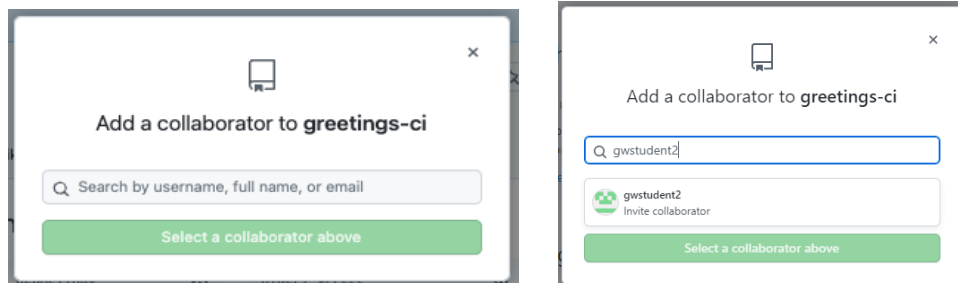
11. From here, you can click on the build job in the graph or the “build” item in the list of jobs to get more details on what occurred on the runner system. You can expand any of the steps in the list to see more details.



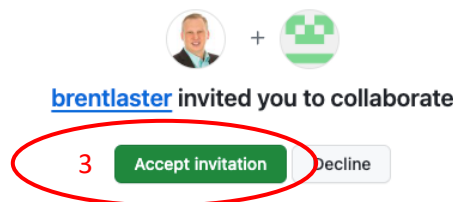
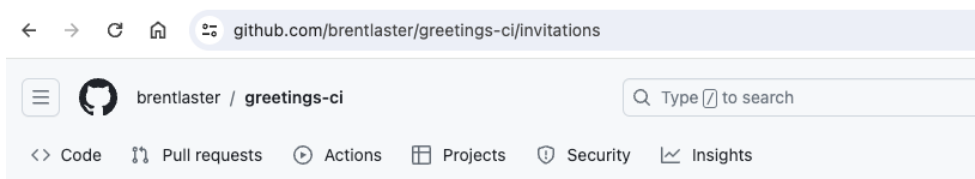
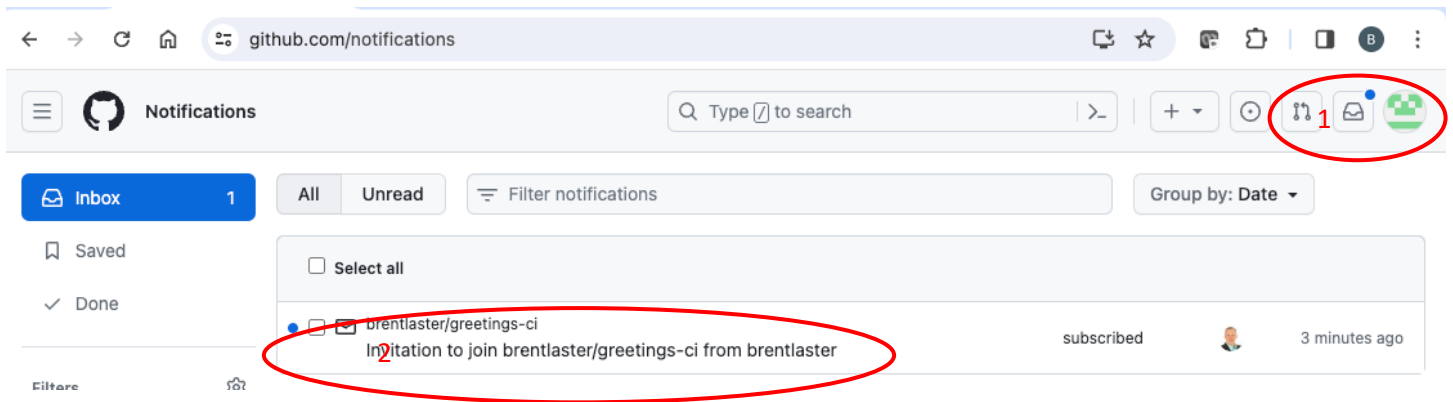
12. In preparation for the next lab, we need to add your second github userid as a collaborator to this repository. Go to the repository’s “Settings” and then select “Collaborators” on the left under “Access”. Then click the “Add people” button.



13. In the dialog box that pops up, enter the other GitHub userid you have and then click on the specific id or “Select a collaborator above”. Then, click on “Add <userid> to this repository”. That userid should then receive an email with the invite which you can accept.



14. Make sure to respond to the email and accept the invitation **OR** log into your other account and accept via the notifications.

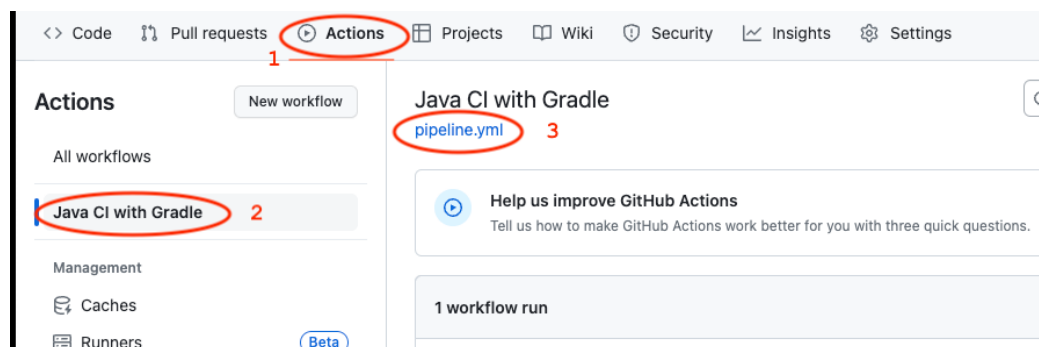


END OF LAB

## Lab 2 – Best practices: CI pre-checks – Part 1

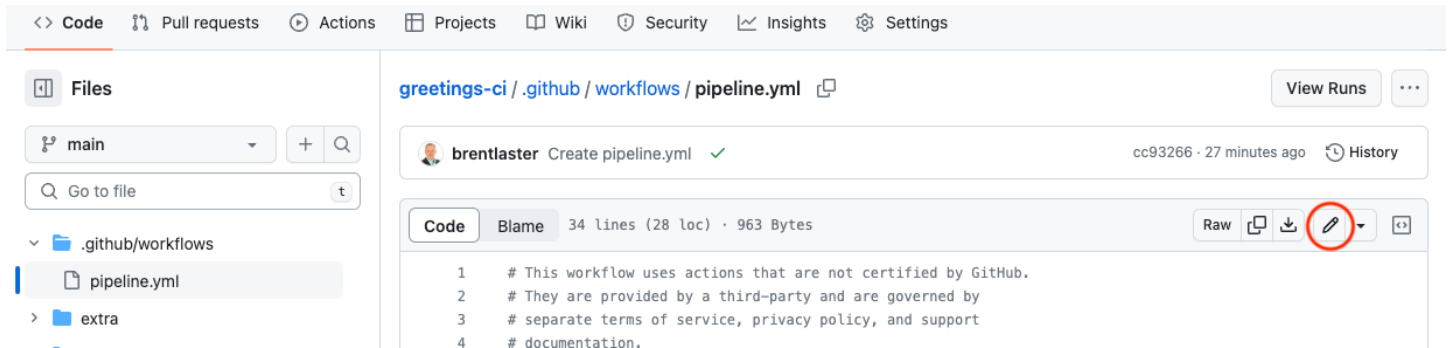
**Purpose:** In this lab, we'll utilize GitHub's code review functions to create a Pull Request and add reviewers

1. Let's suppose we want to use a more general version of the Gradle build action. Still in the greetings-ci repo, start out by selecting the "Actions" menu (if not already there), then select the "Java CI with Gradle" workflow and then click on the "pipeline.yml" file above the list of runs.



- You'll now have the "pipeline.yml" file open in the editor. Notice the various parts of the workflow. We have the "on:" section starting at line 10 where the triggering events for CI are detected. Then we have a section that sets permissions for the workflow relative to the repository. That is followed by the "jobs" section that contains two steps – one to setup the Java environment to compile the code and one to execute the actual build step. Both use predefined actions to do the work.

- Now let's make some changes in the file. Click on the pencil icon to start editing it.



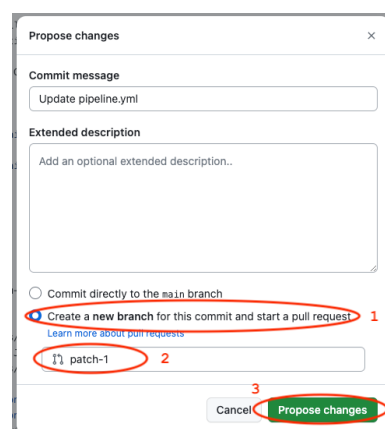
- Pretend we need to roll back to a different action and version. Let's edit the file to change the name and version of the action.

**uses: gradle/actions/setup-gradle@ec92e829475ac0c2315ea8f9eced72db85bb337a # v3.0.0**

and change it to (just copy and replace the text or type in the editor)

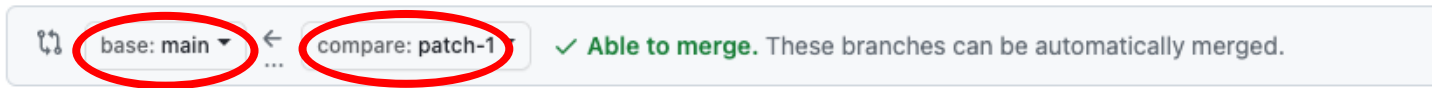
**uses: gradle/gradle-build-action@v2**

- As you did in Lab 1, click on the green "Commit changes..." button in the upper right corner. In the dialog, enter a comment if you want and select the option to "Create a new branch..." You can change the generated branch name if you want. In this case, I've changed it to "patch-1". Then click "Propose changes".

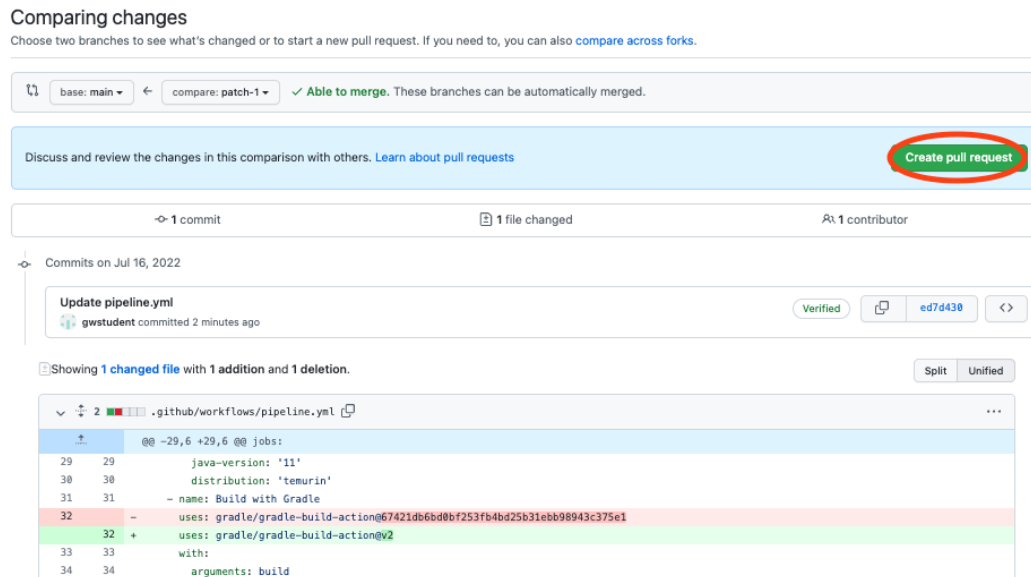




6. At this point, you'll see a screen showing you the changes and what's being compared at the top. It should also show a green checkmark with "Able to merge." next to it. We're going to create a pull request to be reviewed. **IMPORTANT:** Make sure that you select your <userid/greetings-ci:main> on the left side and <userid/greetings-ci:patch-1> on the right side. The gray bar should look like the one below.



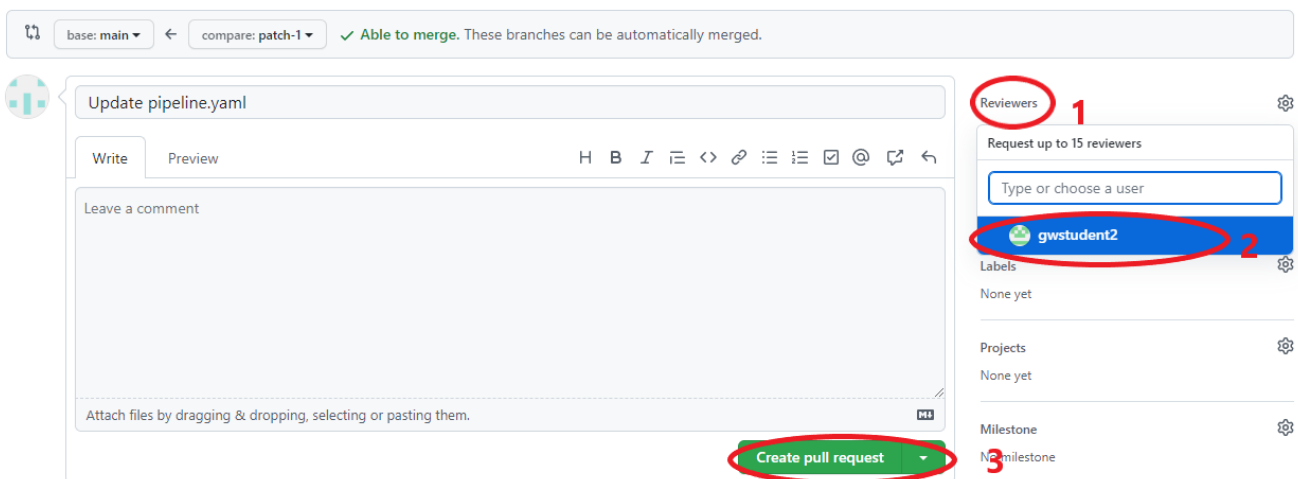
7. Click on the green "Create pull request" button.



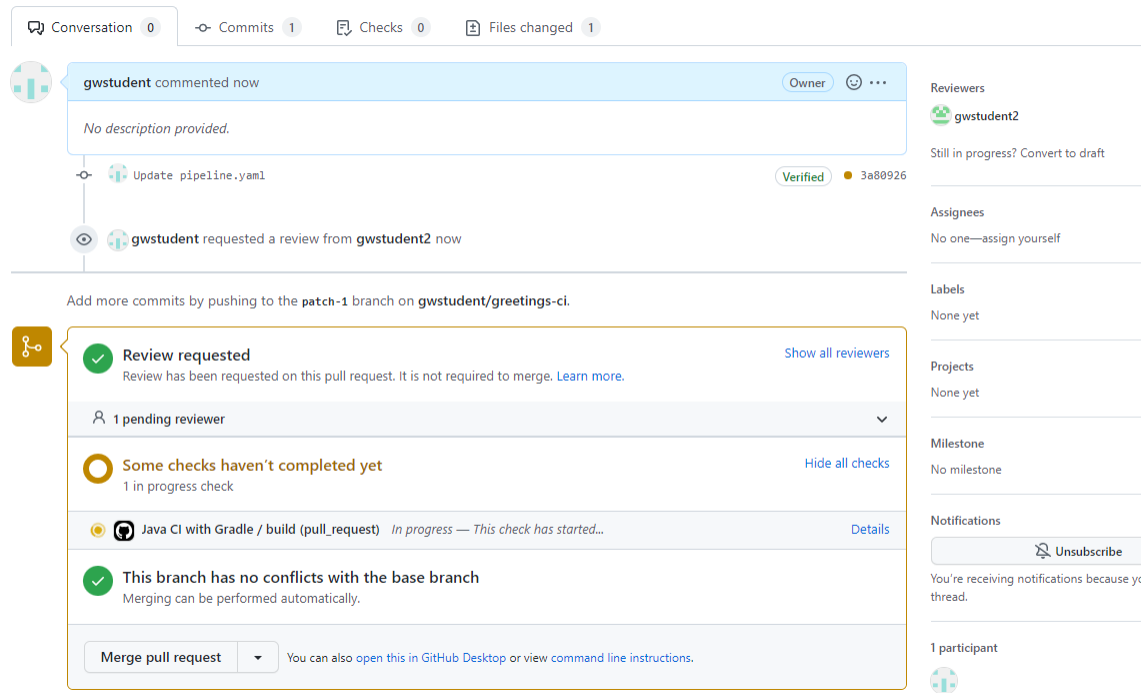
8. You'll now be on the screen to create the pull request. Let's add your secondary GitHub id as a reviewer. In the upper right, click on the "Reviewers" link, then select your other id from the list. (You can just make sure it's checked and hit ESC or type it into the field.) Make sure your other userid shows up in the Reviewers section now. Then click on "Create pull 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](#).



8. Afterwards, you'll be on the screen for the pull request. Note further down on the page, you can see where the automated checking has been kicked off. This checking is the workflow file pipeline.yaml. It was kicked off via the CI process because of the "on" clause that matched a pull request on main. Eventually, this check should succeed. (You may need to refresh your browser screen to see the update.). You can also see the one pending review you have from your secondary GitHub userid.

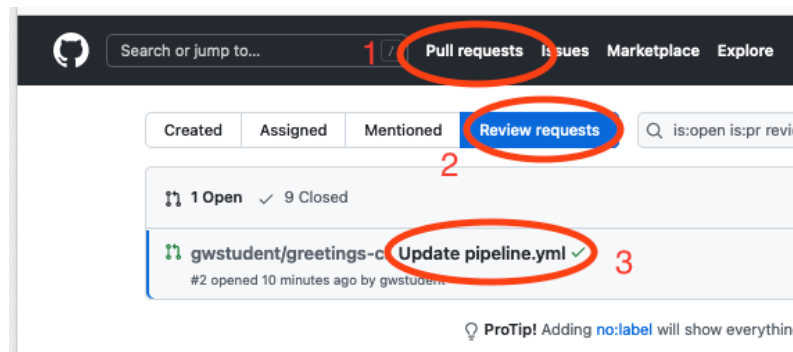


END OF LAB

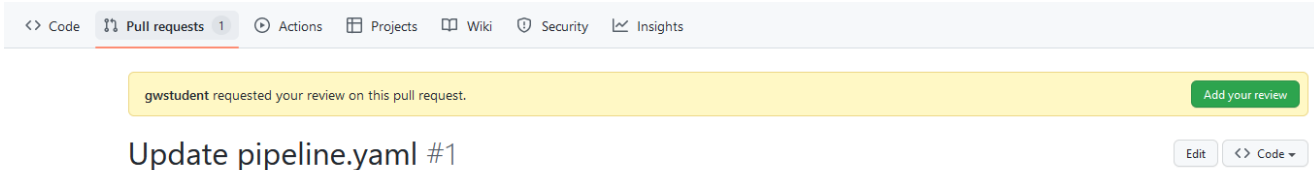
### Lab 3 – Best practices: CI pre-checks – part 2

**Purpose:** In this lab, we'll utilize GitHub's code review functions to finish reviewing, validating, and approving the code

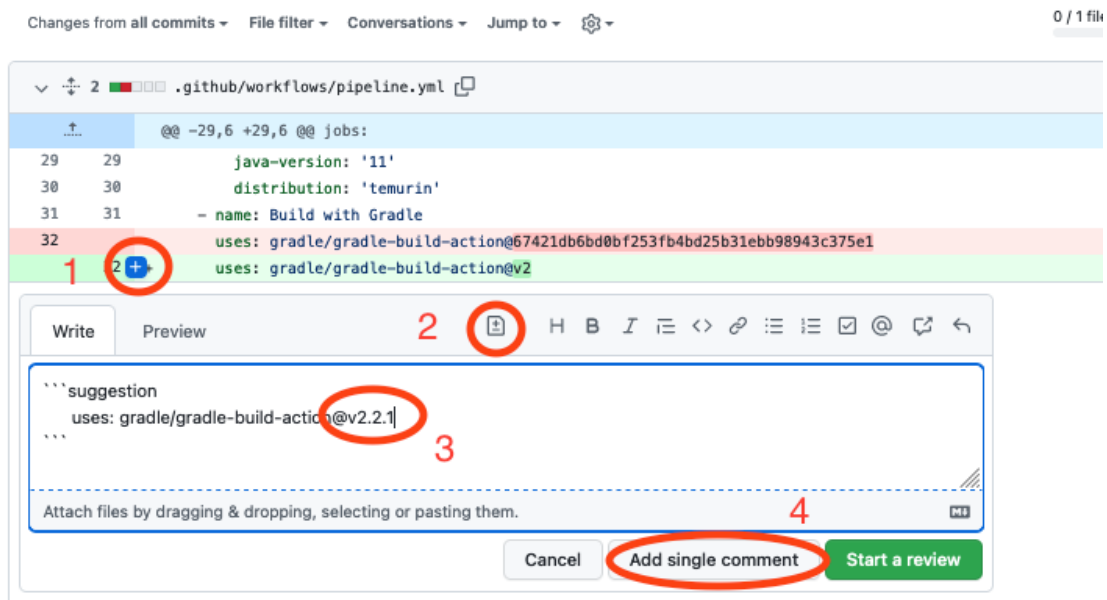
1. Picking up where we left off in the last lab, now we'll do the review. In a separate browser session or tab, log in to your secondary GitHub userid (the one you added as a collaborator and a reviewer). After you log in, you can either go to <https://github.com/pulls/review-requested> or click on the "Pull requests" item at the top, then "Review requests". Then click on the commit message for the pull request.



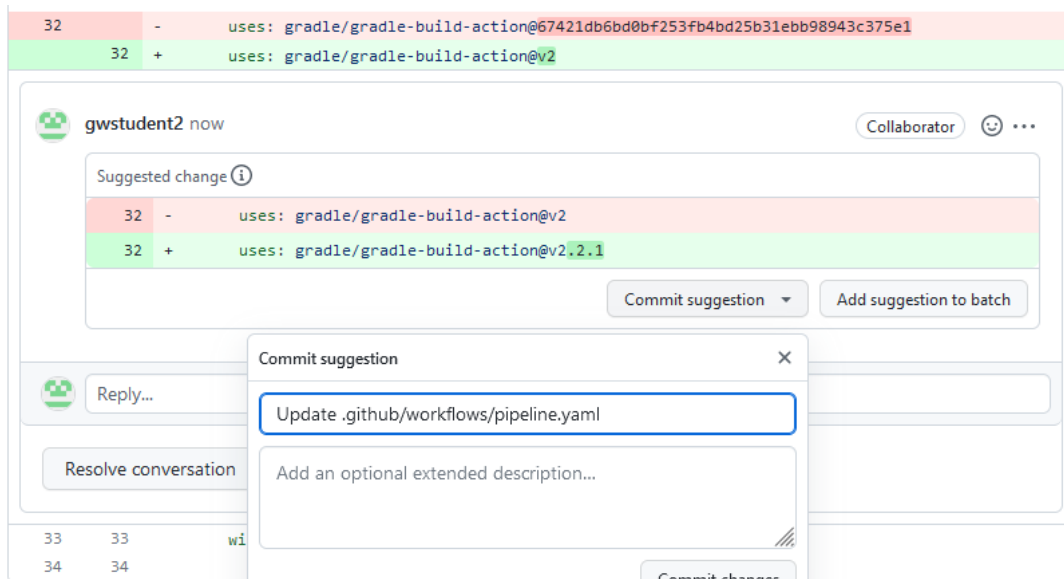
2. This will open up the pull request. There is a button at the top to “Add your review”. Click on that.



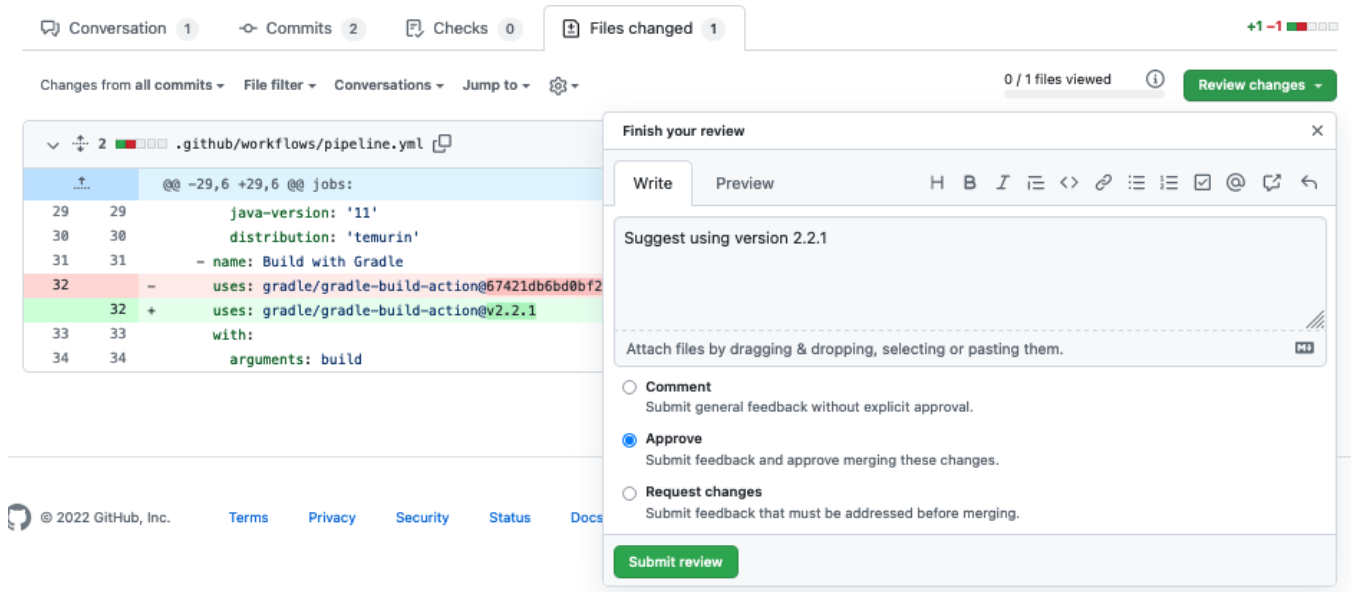
3. Let’s add a suggestion to use v2.2.1 of the build action. In the file differences section, click on the line with the “v2” at the end. Then click on the blue “+” sign that pops up. Now click on the icon to add a suggestion and, in the text that gets filled in, edit the “v2” to be “v2.2.1”. Finally, click on the “Add single comment” since we’re only doing one comment here.



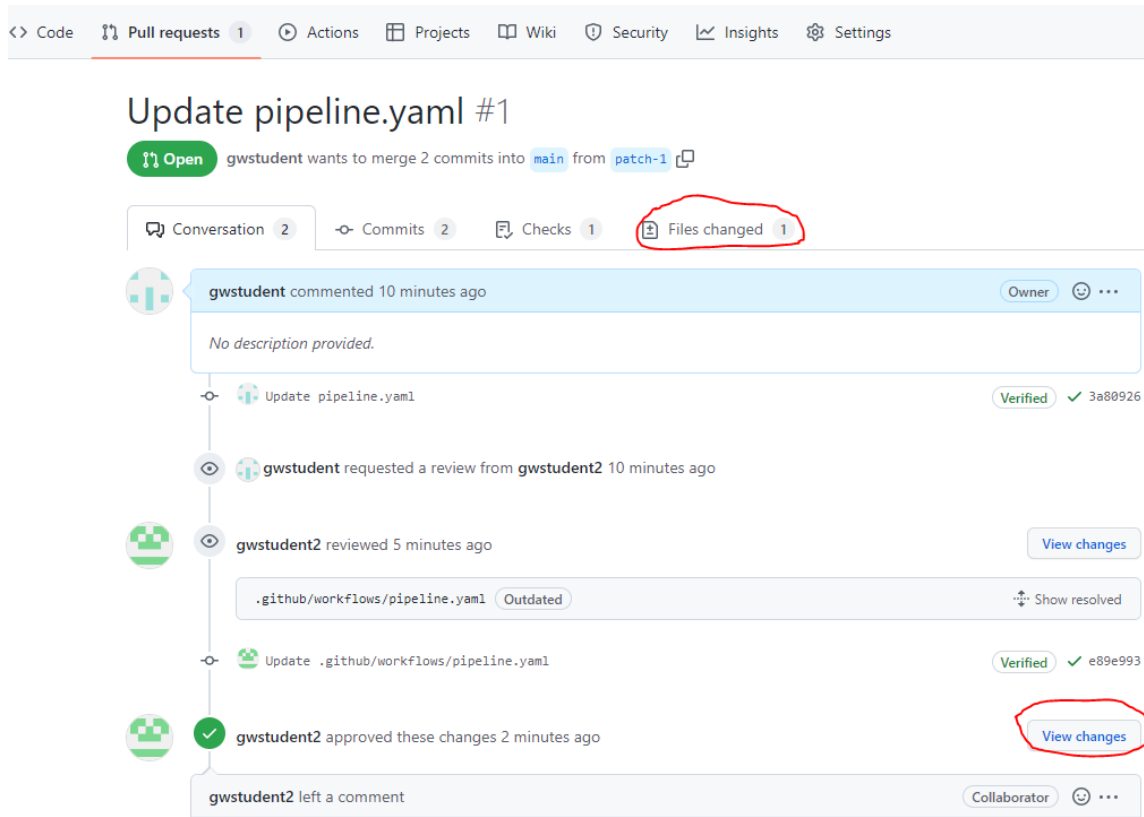
4. In the next screen, click on “Commit suggestion” and then “Commit changes” in the dialog box that comes up.



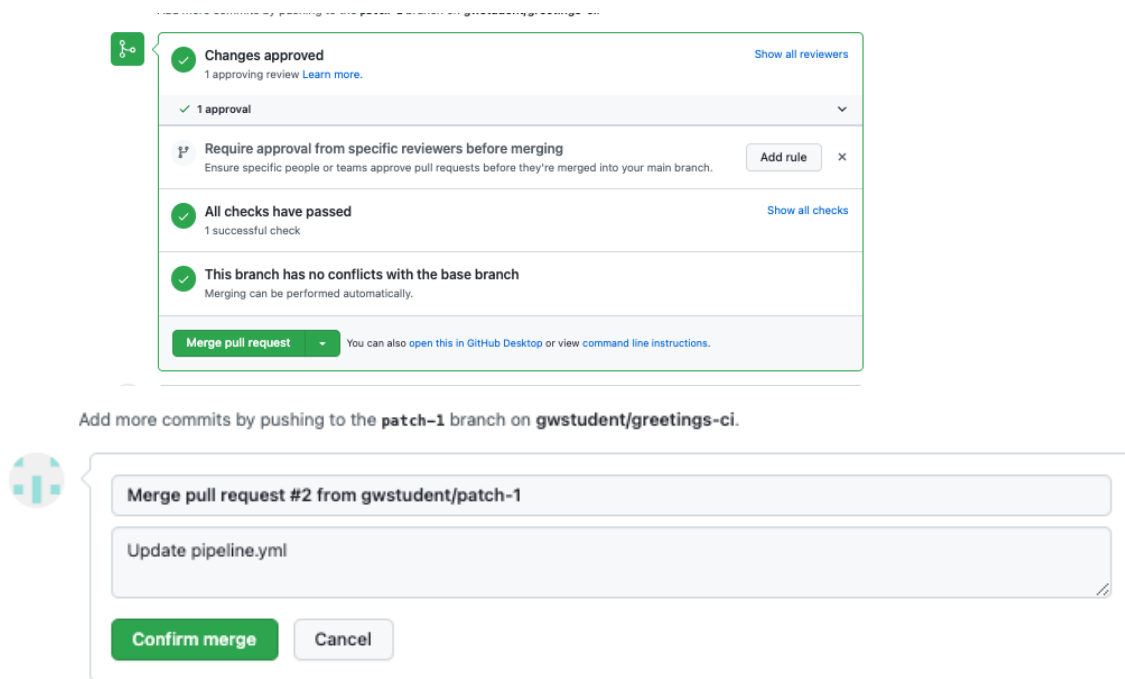
5. On the next screen, select the “Review changes” button. In the dialog, you can add a comment, and then select Approve (since this is only a suggestion) and then click on “Submit review” at the bottom of the dialog.



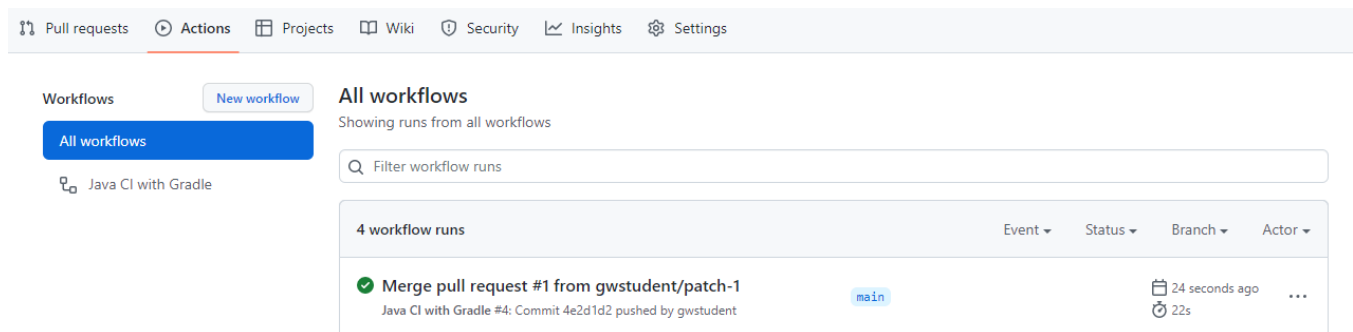
6. Go to the session with your original GitHub userid or log out of the other one and log back in if you need to. Go to the *Pull requests* menu at the top, find the pull request and click on the commit message. Then you should see a screen like below. You can use either the “Files changed” tab or the “View changes” button to see the changes that have been made.



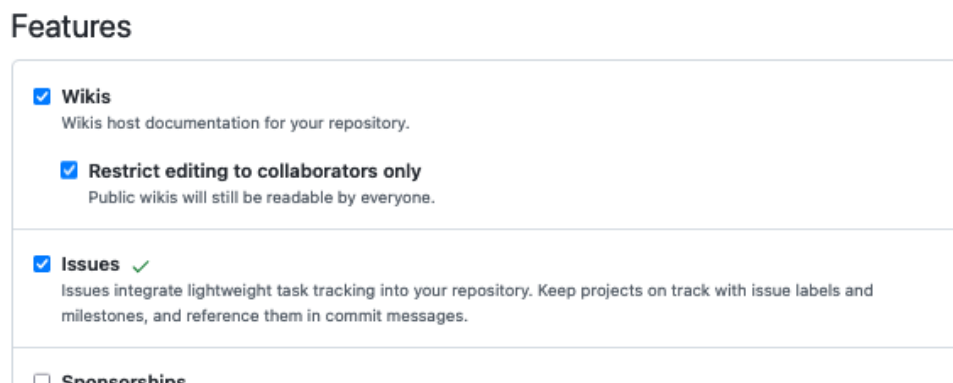
7. When done looking at the changes, go back to the "Conversation" tab. Now, you can go ahead and merge the pull request by clicking on the "Merge pull request" button and then the "Confirm merge" button.



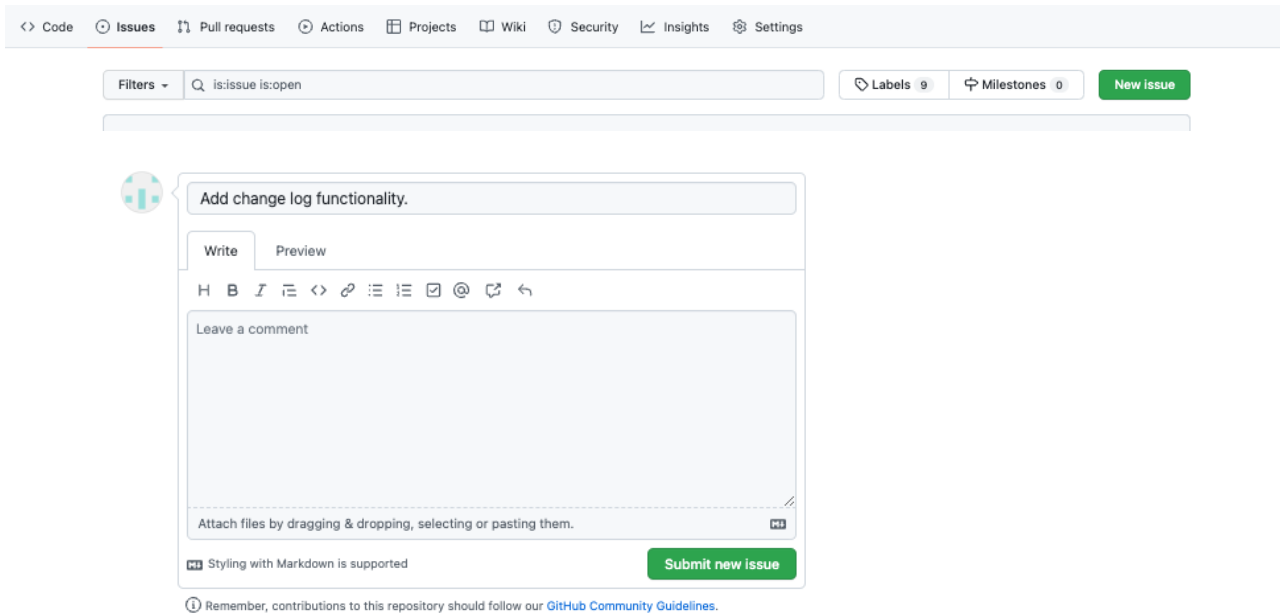
8. Click back on the Actions menu at the top and you should see another run of our workflow that will be using the new version of Gradle.



9. In the next lab, we'll look at adding functionality for a Change Log. Let's open an issue to track that. First, ensure that the repository has the Issues feature turned on. Go to the repository's "Settings" and then scroll down until you see the "Features" section. Then, check the box for "Issues".



10. Now, click on the “Issues” tab at the top of the repository page, then the “New issue” button on the right. Then fill in the title with something like “Add change log functionality” and click the “Submit new issue” button.



Filters  Labels 9 Milestones 0 [New issue](#)

[Add change log functionality.](#)

Write Preview

H B I

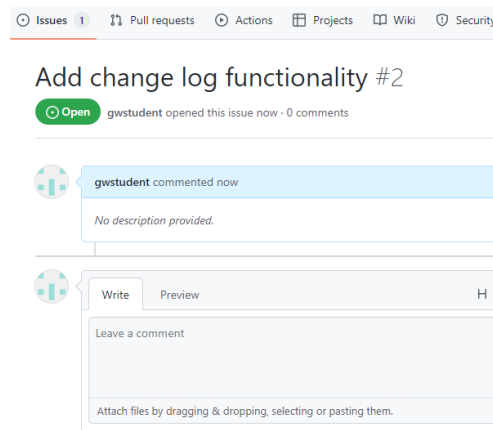
Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Styling with Markdown is supported [Submit new issue](#)

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

11. Take note of what number is assigned to the issue – you will need it in a later lab. (It will probably be #2 for you)



END OF LAB

## Lab 4 – Merge requests and conventional commits

**Purpose:** In this lab, we'll see how to do merge requests and use conventional commits.

- For this lab, we'll make a fork of your current greetings-ci repo into your secondary GitHub account. In another tab or session, log in to GitHub with your secondary GitHub ID. Then go to <https://github.com/<primary-github-id>/greetings-ci> and fork that project for your secondary ID. Refer back to Lab 1, step 2 if you need a reminder of how to do this.
- As prep for running actions, switch to the Actions tab, you'll see a message that says “Workflows aren't being run on this forked repository”. Just go ahead and click the big green button that says “I understand...”



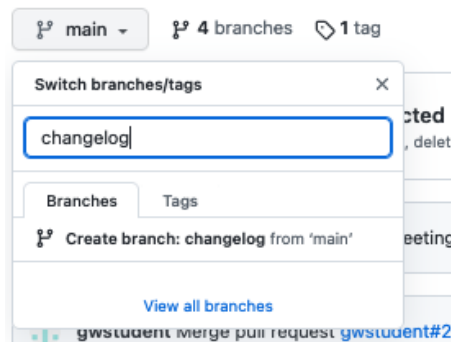
### Workflows aren't being run on this forked repository

Because this repository contained workflow files when it was forked, we have disabled them from running on this fork. Make sure you understand the configured workflows and their expected usage before enabling Actions on this repository.

[I understand my workflows, go ahead and enable them](#)

[View the workflows directory](#)

- We want to propose adding a changelog and using conventional commits in the repository through our workflow file. Let's create a new branch to try out any changes on. We'll call it "changelog". 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 "changelog". Then click on the "Create branch: changelog from 'main'" link.



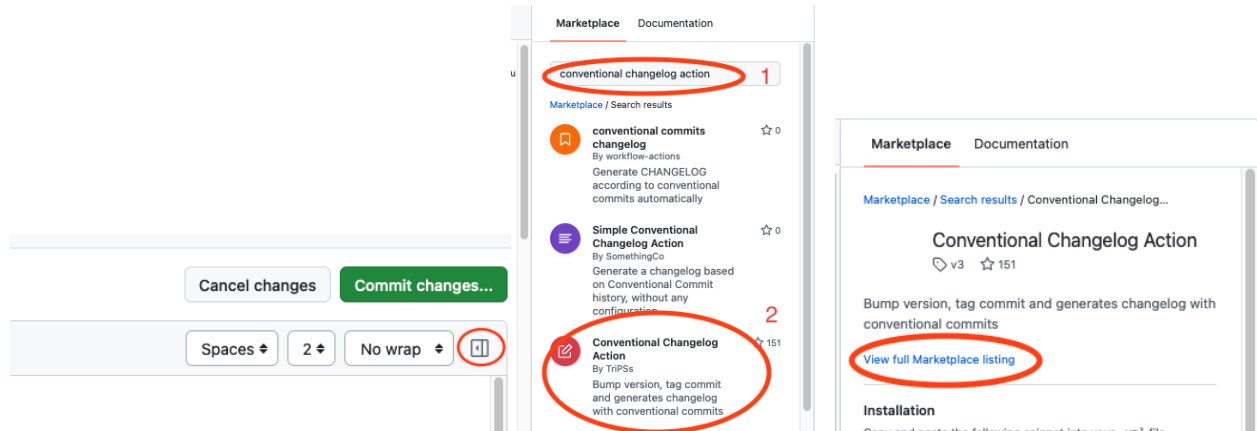
- Now you should be on the "changelog" branch. Open the `.github/workflows/pipeline.yaml` file (hint: click on `.githubs/workflow` row). and edit it by clicking on the pencil icon again. Refer back to Lab 2, step 2 if you need a reminder of how to do this. Then change the reference in the "on:" clause to add the "changelog" branch in addition to "main". (Leave the pull\_request one as-is for now.) Make sure you are on the *changelog* branch before you proceed.

```

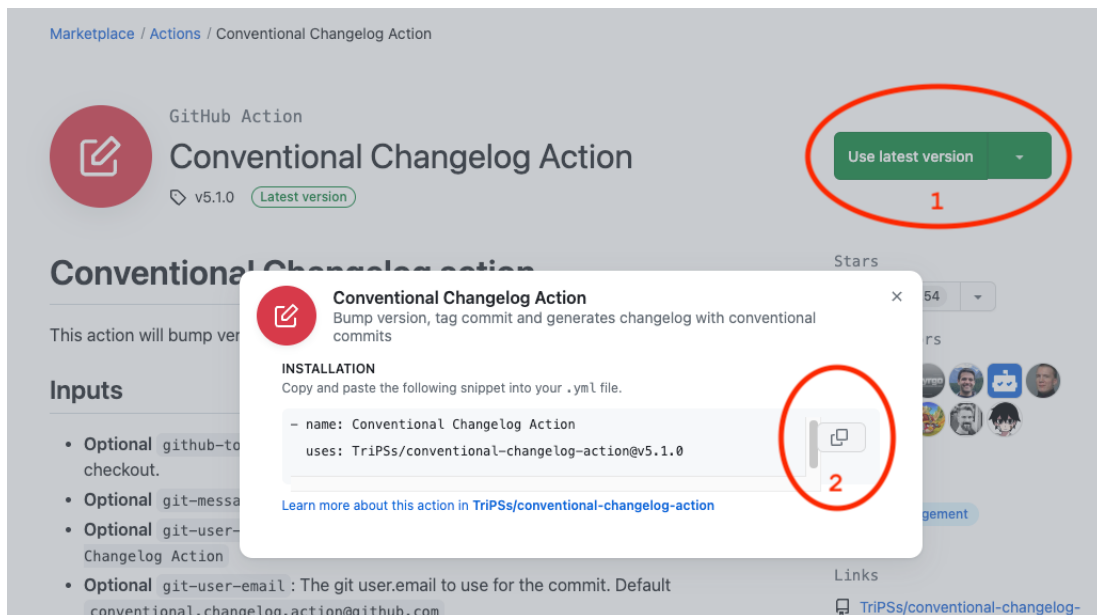
8  name: Java CI with Gradle
9
10 on:
11   push:
12     branches: [ "main", "changelog" ]
13   pull_request:
14     branches: [ "main" ]
15

```

5. Let's see if there's an action that can help us with the changelog functionality and also use conventional commits. If you don't see the search pane, click on the small box to the right of the edit icon to show the search pane. Then, on the right side will be a pane where you can search for Marketplace actions. In the search box, enter "**conventional changelog action**". And then click on the one that is named "**Conventional Changelog Action By TriPSs**". On the next screen, click on the option to "View full Marketplace listing" to see the full information about the action.



6. On the marketplace page for the action, you can read more about it if you want. We'll just use the latest version. Click on the green button that says, "Use latest version". Then click on the copy icon next to the right of the text in the dialog that pops up.



7. Go back to the tab where you are editing the pipeline.yaml file under your secondary GitHub userid. Paste the usage info you just copied from the marketplace action as a new step in the build job **AFTER** the *checkout* step and **BEFORE** the step that *sets up JDK 17*. Pay attention to get the indentation right as shown below.



```

16 jobs:
17   build:
18
19     runs-on: ubuntu-latest
20     permissions:
21       contents: write
22
23     steps:
24     - uses: actions/checkout@v4
25
26     - name: Conventional Changelog Action
27       uses: TriPSs/conventional-changelog-action@v5.1.0
28
29     - name: Set up JDK 17
30       uses: actions/setup-java@v4
31       with:
32         java-version: '17'

```

8. Since this action will ultimately be creating a changelog in our repo, we need to allow our workflow now to have write permissions on content. To enable this, change the line in the “permissions” clause that is “contents: read” to “contents: write”.

```

10 on:
11   push:
12     branches: [ "changelog" ]
13   pull_request:
14     branches: [ "main" ]
15
16 permissions:
17   contents: write
18

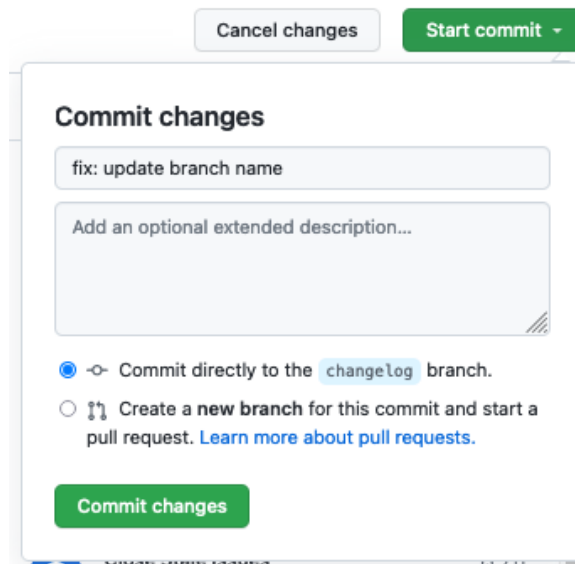
```

9. After these changes are made, start a new commit by clicking on the button in the upper right. You can commit directly to the “changelog” branch since it is a separate branch for trying this out. Edit the commit message to be “feat: add changelog” to identify this as a feature using conventional commits. Then click “Commit changes”.

10. After this is done, switch back to the “Code” tab and the “changelog” branch. After the workflow completes, you should see two new files in your repository in the “changelog” branch: CHANGELOG.md and package.json. The change log is the automatically generated record of changes. And the package.json file is the current version. You can look at the files to see the contents. Also notice on the right side of the screen under releases, you now have automatic tags that have been applied. You can click on the “tags” link to see the tags.
11. Let’s make one more update to the code to see the process. This time we’ll make a simple fix. Edit the pipeline.yaml file again in .github/workflows (on the *changelog* branch) and change the “pull\_request” part of the “on:” clause to also use the “changelog” branch.

```
9
10 on:
11   push:
12     branches: [ "main", "changelog" ]
13   pull_request:
14     branches: [ "main", "changelog" ]
15
```

12. Now let’s commit these changes to the “changelog” branch with the “fix:” label. Start a commit and enter “**fix: update branch name**” for the commit message. Then click “Commit changes”. After the workflow completes, take a look at the CHANGELOG.md file to see how the fix was recorded. (Remember you’ll need to be in the “changelog” branch.) You can also look at the package.json file and tags if you want.



END OF LAB