

# CI/CD and DevOps in 3 Weeks

## Week 1

Revision 1.5 – 11/03/22

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".*

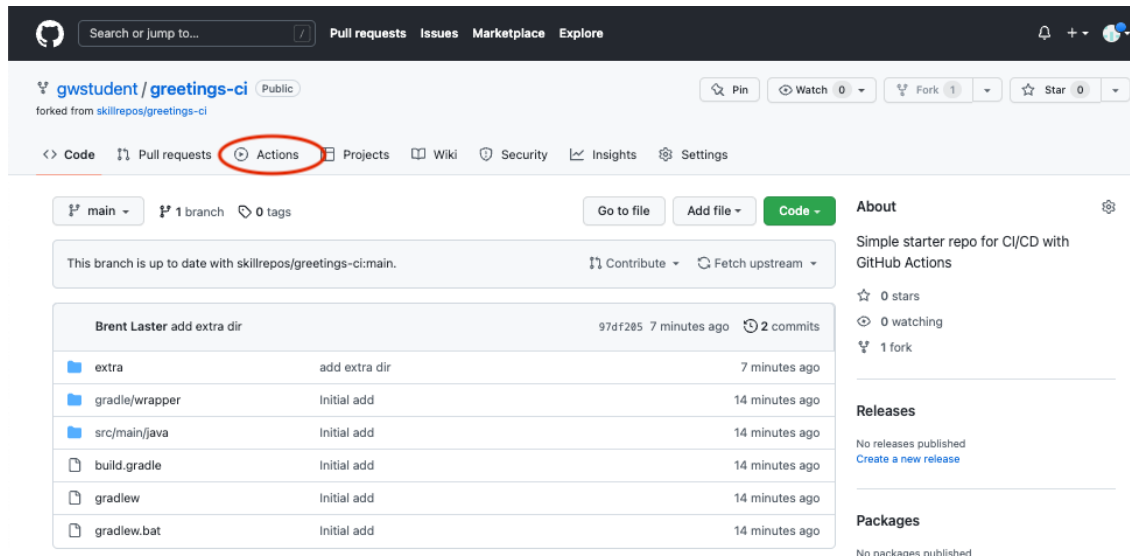
### Lab 1 – Creating a simple example

**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.

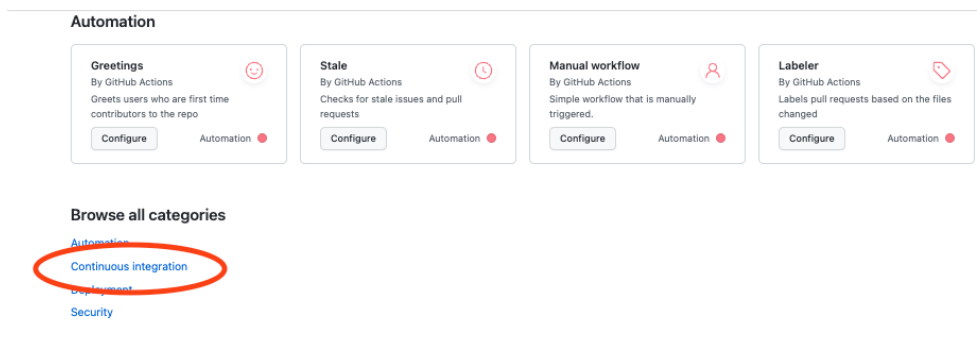
1. Log in to GitHub with your GitHub id
2. Go to <https://github.com/skillrepos/greetings-ci> and fork that project into your own GitHub space. After this, you'll be on the project in your user space.

The image shows two screenshots from the GitHub website. The top screenshot displays the repository page for 'skillrepos/greetings-ci'. The 'Fork' button, which shows '0' forks, is circled in red. The bottom screenshot shows the 'Create a new fork' dialog. In this dialog, the 'Owner' is 'gwstudent' and the 'Repository name' is 'greetings-ci'. The 'Description' field contains 'Simple starter repo for CI/CD with GitHub Actions'. The 'Copy the main branch only' checkbox is checked. At the bottom of the dialog, the 'Create fork' button is circled in red.

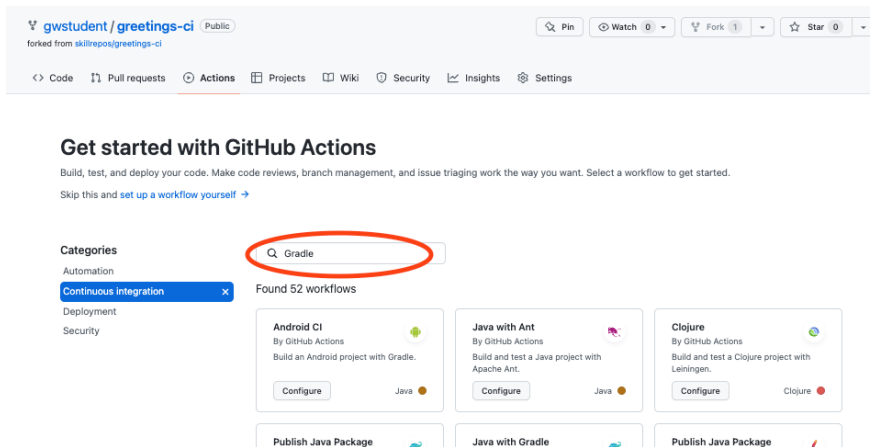
3. We have a simple java source file named *echoMsg.java* in the subdirectory *src/main/java*, a Gradle build file in the root directory named *build.gradle*, and some other supporting files. We could clone this repository and build it manually via running Gradle locally. But let's set this to build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



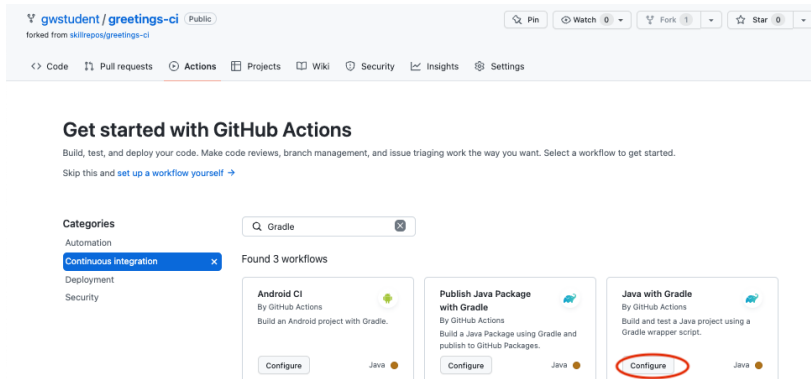
4. This will bring up a page with categories of starter actions that GitHub thinks might work based on the contents of the repository. We'll select a specific CI one. Scroll down to near the bottom of the page under "Browse all categories" and select "Continuous integration".



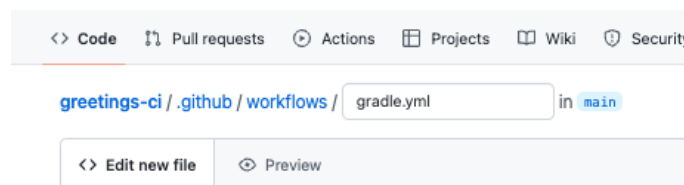
5. In the CI category page, let's search for one that will work with Gradle. Type "Gradle" in the search box and press Enter.



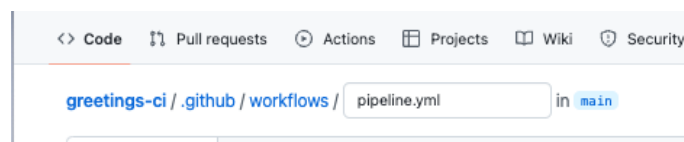
- From the results, select the “Java with Gradle” one and click the “Configure” button to open a predefined workflow for this.



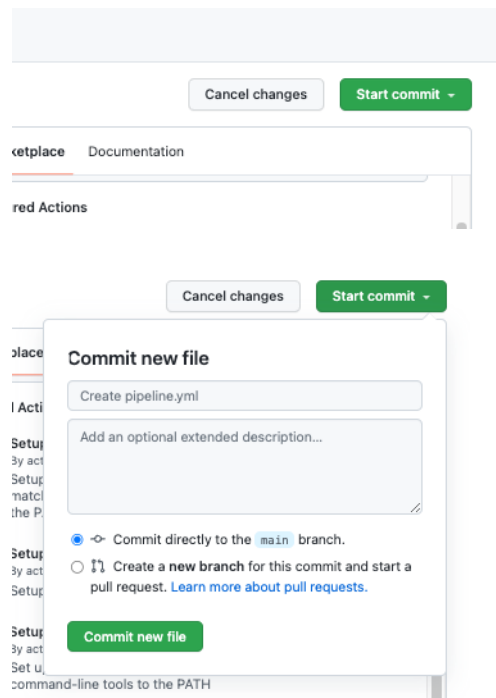
- This will bring up a page with a starter workflow for CI that we can edit as needed. The only edit we want to make here right now is to change the name. In the top section where the path is, notice that there is a text entry box around “gradle.yml”. This is the current name of the workflow. Click in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



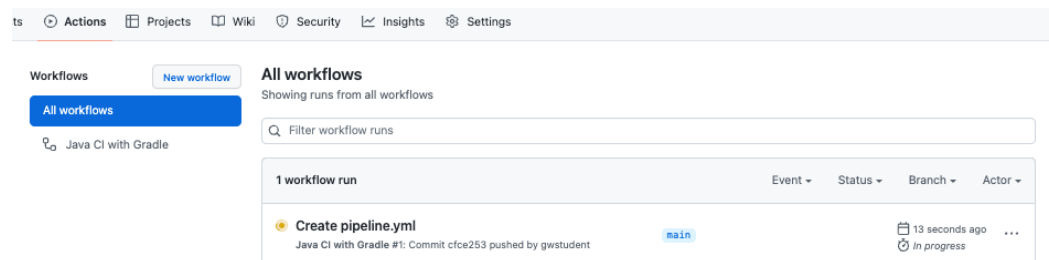
TO



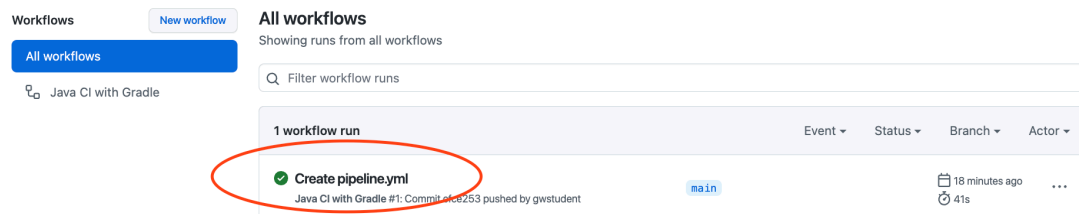
8. Now, we can go ahead and commit the new workflow via the “Start commit” button in the upper right. In the dialog that comes up, you can enter an optional comment if you want. Leave the “Commit directly...” selection checked and then click on the “Commit new file” button.



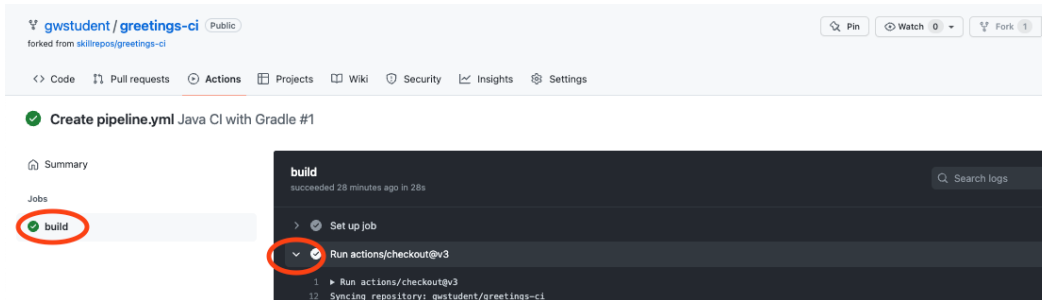
9. Since we’ve committed a new file and this workflow is now in place, the “on: push:” event is triggered and the CI automation kicks in. Click on the Actions menu again to see the automated processing happening.



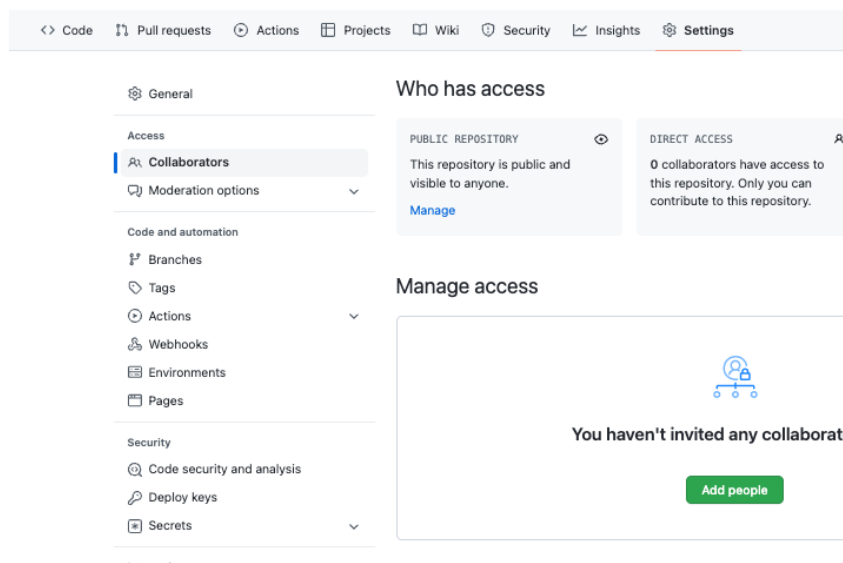
10. After a few moments, the workflow should succeed. (You may need to refresh your browser.) After it is done, you can click on the commit message for the run to get to the details for that particular run.



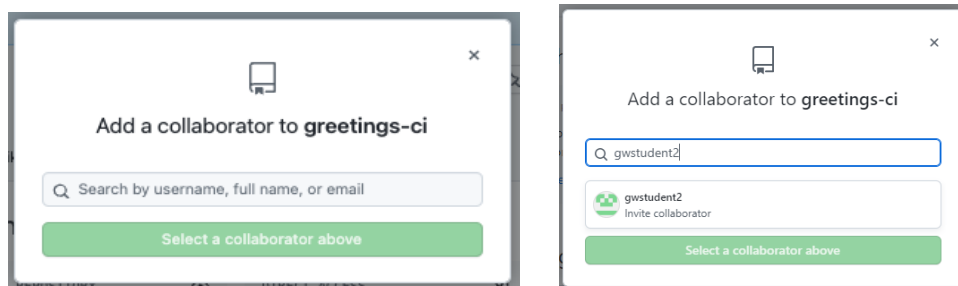
11. From here, you can click on the build job in the graph or the “build” item in the list of jobs to get more details on what occurred on the runner system. You can expand any of the steps in the list to see more details.



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 "Select a collaborator above" and then the "Add <userid> to this repository". That userid should then receive an email with the invite which you can accept.



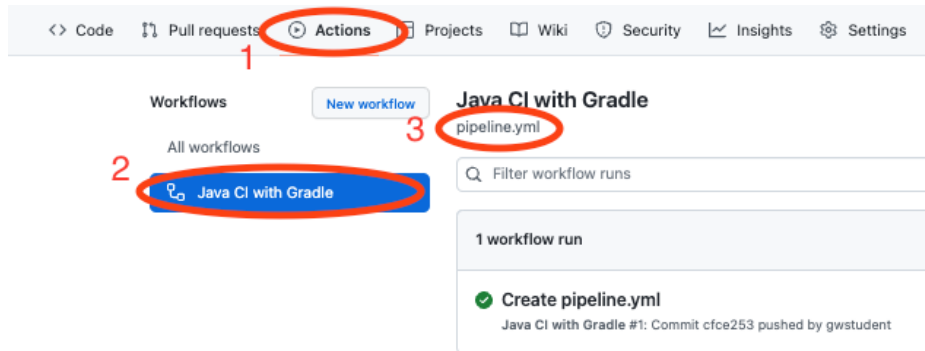
14. **Make sure to respond to the email and accept the invitation!**

END OF LAB

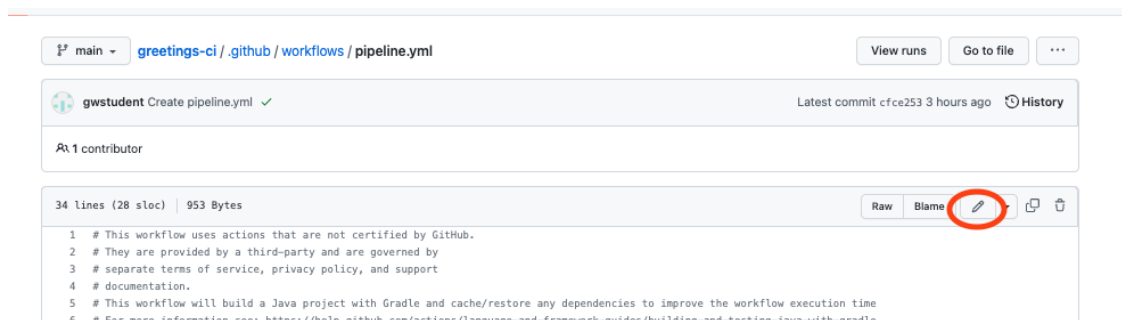
## Lab 2 – Best practices: CI pre-checks – Part 1

**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.



2. 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.
3. Now let's make some changes in the file. Click on the pencil icon to start editing it.



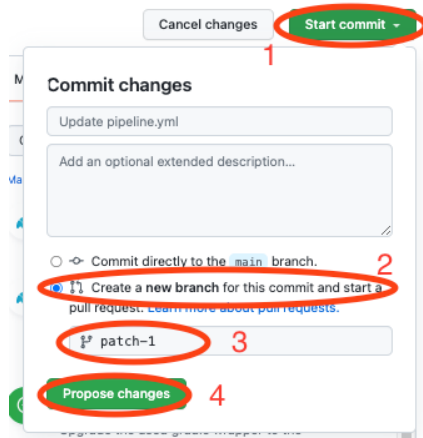
4. Let's edit the file to change the version of the Gradle action to "v2". Find the line like this:

**uses: gradle/gradle-build-action@67421db6bd0bf253fb4bd25b31ebb98943c375e1**

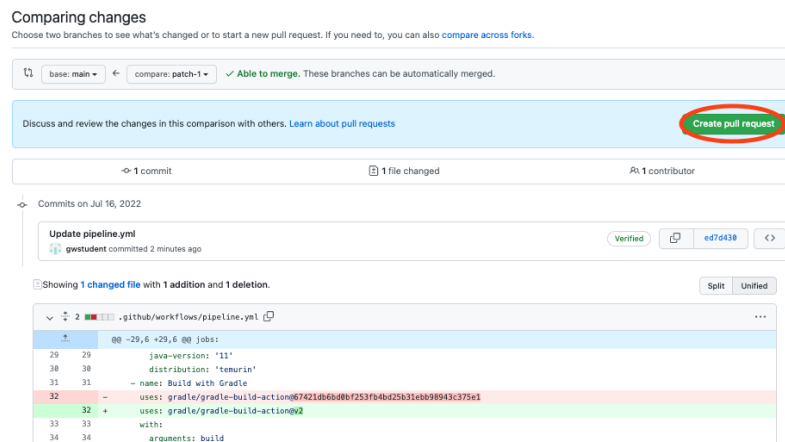
and change it to (just type and replace the text in the editor)

uses: gradle/gradle-build-action@v2

- As you did in Lab 1, click on the green “Start commit” 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”.



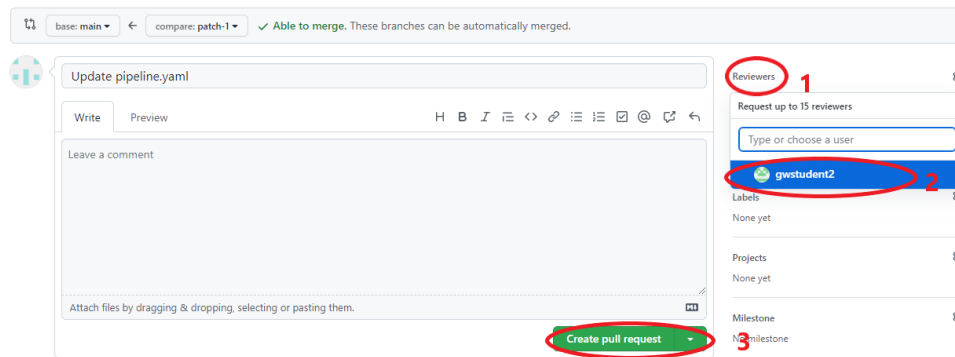
- At this point, you’ll see a screen showing you the changes and what’s being compared at the top. It should also show a green checkmark with “Able to merge.” next to it. We’re going to create a pull request to be reviewed. Click on the green “Create pull request” button.



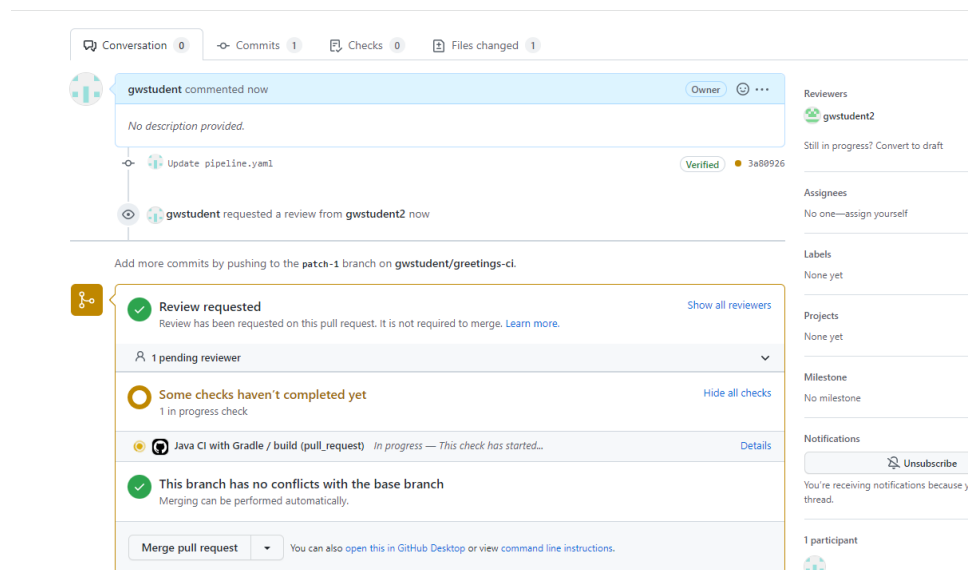
- You’ll now be on the screen to create the pull request. Let’s add your secondary GitHub id as a reviewer. In the upper right, click on the “Reviewers” link, then select your other id from the list. (You can just make sure it’s checked and hit ESC or type it into the field.) Make sure your other userid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



- 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.



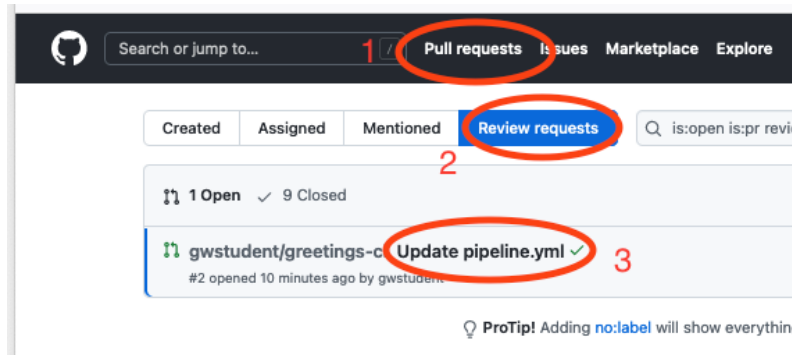
END OF LAB

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

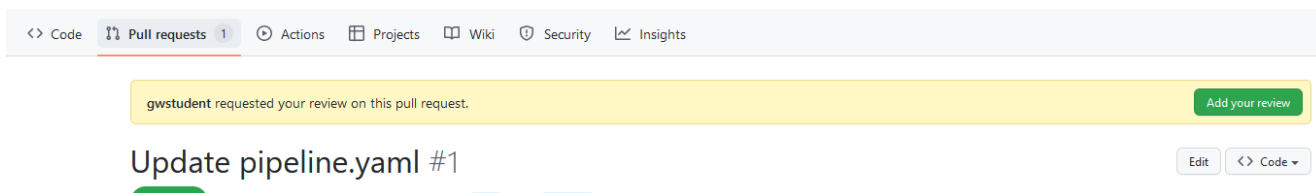
**Purpose:** In this lab, we'll utilize GitHub's code review functions to finish reviewing, validating, and approving the code

- 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.

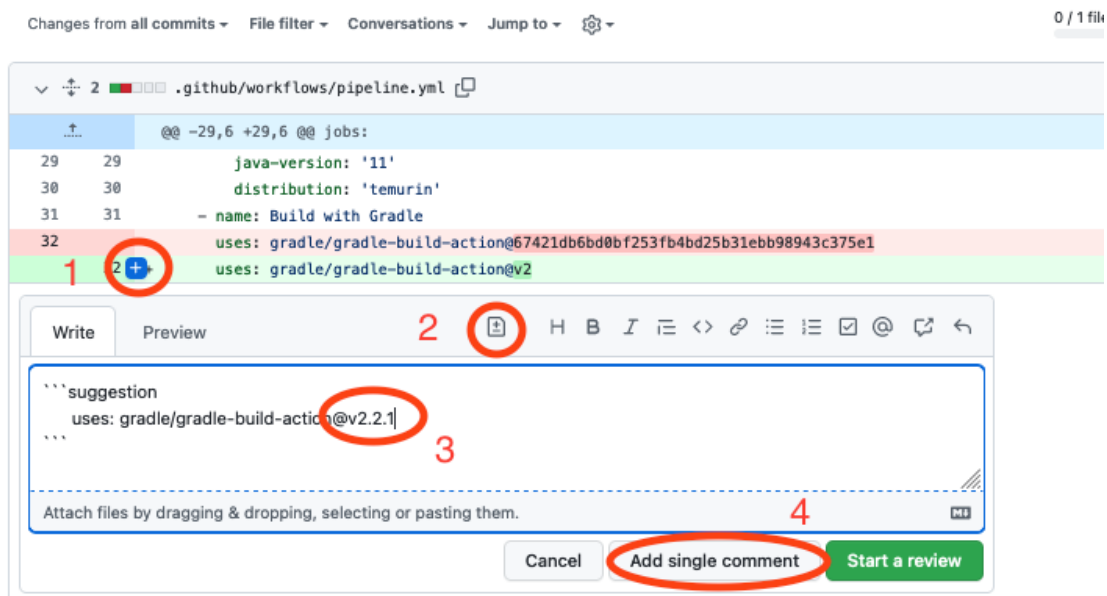




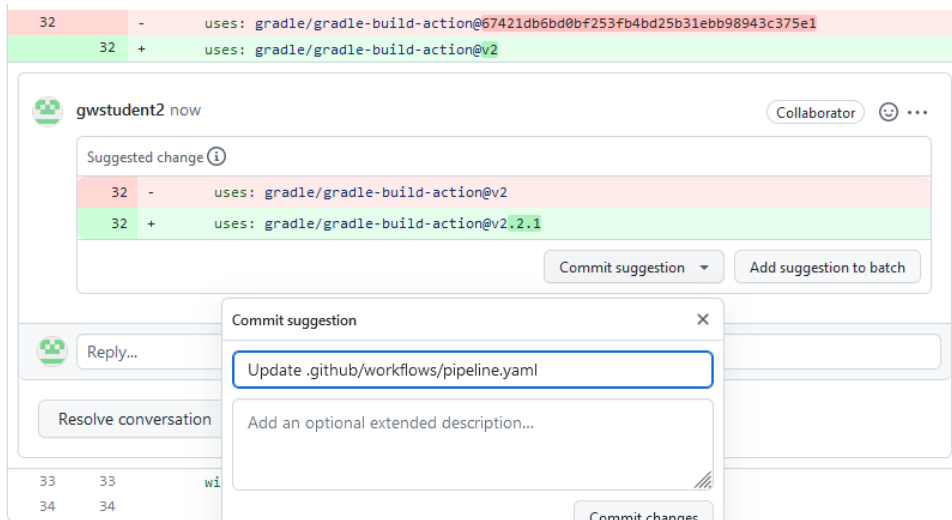
2. This will open up the pull request. There is a button at the top to “Add your review”. Click on that.



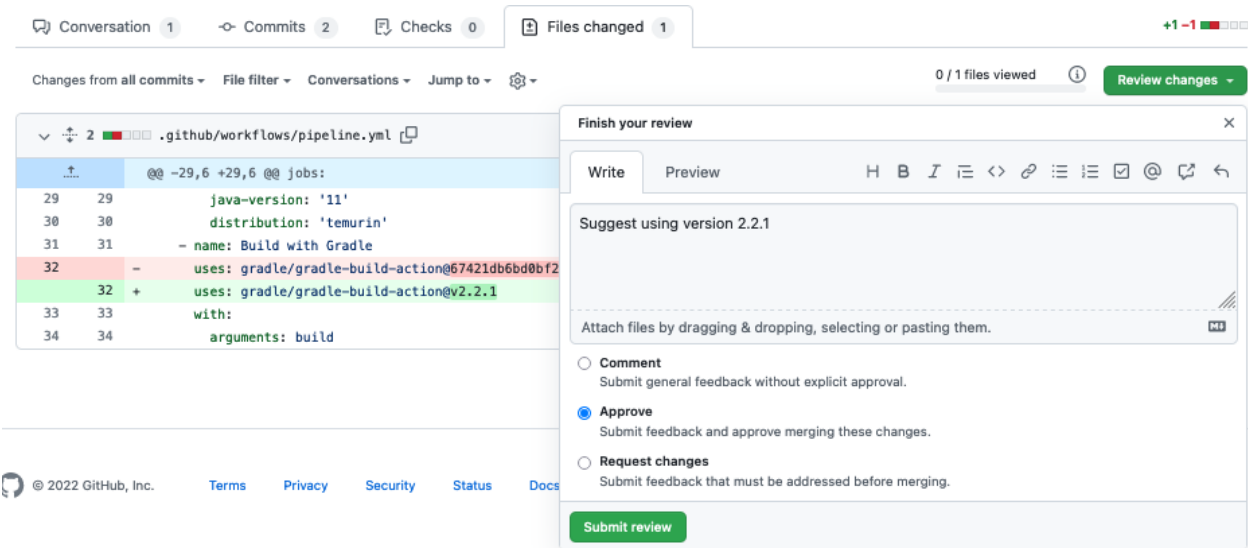
3. Let’s add a suggestion to use v2.2.1 of the build action. In the file differences section, click on the line with the “v2” at the end. Then click on the blue “+” sign that pops up. Now click on the icon to add a suggestion and, in the text that gets filled in, edit the “v2” to be “v2.2.1”. Finally,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



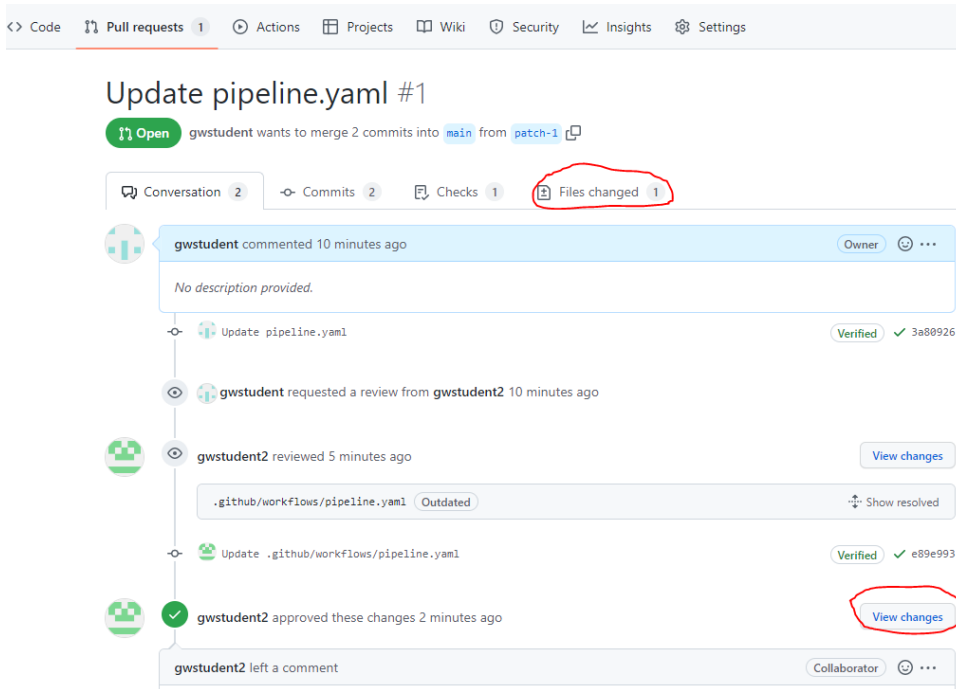
4. In the next screen, click on “Commit suggestion” and then “Commit changes” in the dialog box that comes up.



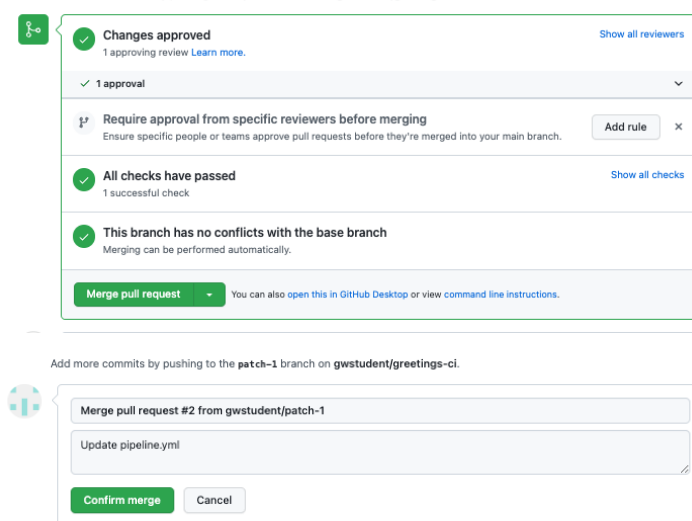
- On the next screen, select the “Review changes” button. In the dialog, you can add a comment, and then select Approve (since this is only a suggestion) and then click on “Submit review” at the bottom of the dialog.



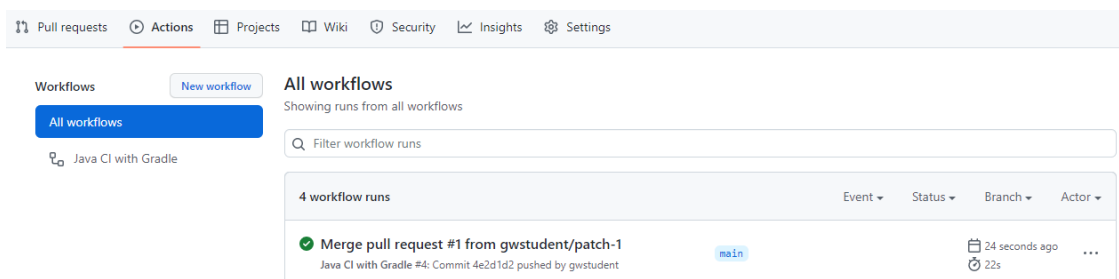
- Go to the session with your original GitHub userid or log out of the other one and log back in if you need to. Go to the *Pull requests* menu at the top, find the pull request and click on the commit message. Then you should see a screen like below. You can use either the “Files changed” tab or the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



- 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.



- 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".

## Features

☒ **Wikis**  
Wikis host documentation for your repository.

☒ **Restrict editing to collaborators only**  
Public wikis will still be readable by everyone.

☒ **Issues** ✓  
Issues integrate lightweight task tracking into your repository. Keep projects on track with issue labels and milestones, and reference them in commit messages.

☐ **Sponsorships**

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.

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Filters is:issue is:open 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)

Issues 1 Pull requests Actions Projects Wiki Security

Add change log functionality #2

Open gwstudent opened this issue now · 0 comments

gwstudent commented now

No description provided.

Write Preview H

Leave a comment

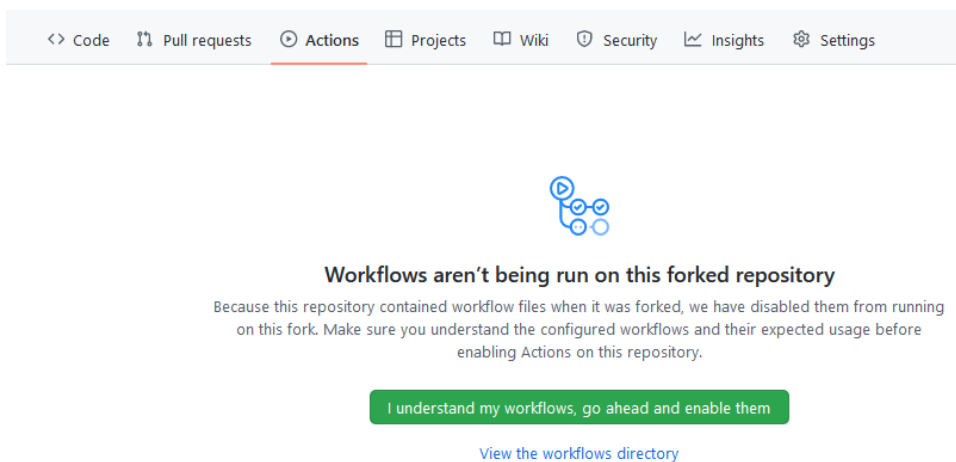
Attach files by dragging & dropping, selecting or pasting them.

## END OF LAB

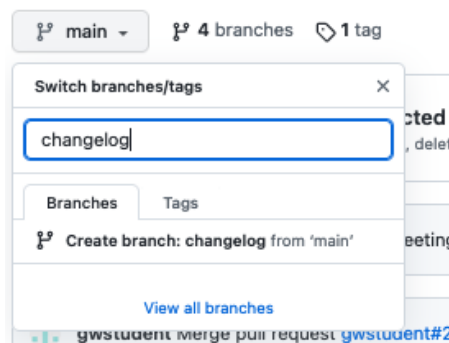
### Lab 4 – Merge requests and conventional commits

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

1. 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.
2. 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..."



3. 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.

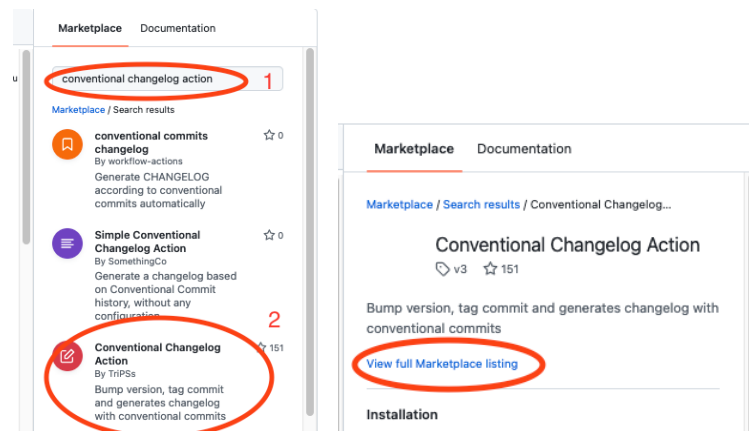


4. Now you should be on the "changelog" branch. Open the .github/workflows/pipeline.yaml file (hint: click on .githubs/workflow row). and edit it by clicking on the pencil icon again. Refer back to Lab 2, step 2 if you need a reminder of how to do this. Then change the reference in the "on:" clause to 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
```

- Let's see if there's an action that can help us with the changelog functionality and also use conventional commits. On the editor screen, 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.



- On the marketplace page for the action, you can read more about it if you want. Now we want to use version 3.14.0. For simplicity, let's grab the code to use that version clicking on the arrow to the right of the big green “Use latest version” button in the top right of the page, selecting the row for v3.14.0, and then clicking on the copy icon next to the right of the text in the dialog that pops up.

GitHub Action

## Conventional Changelog Action

v3.17.0 Latest version

**Conventional Changelog action**

This action will bump version, tag commit and generate a changelog with conventional commits.

**Inputs**

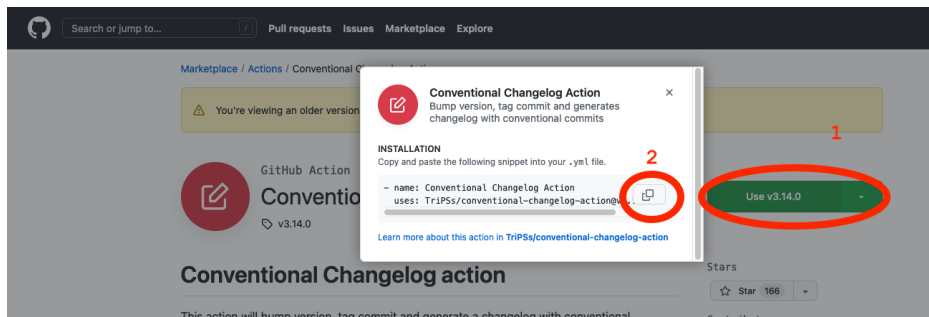
- Optional** `github-token`: GitHub token, if different permissions required than from checkout.
- Optional** `git-message`: Commit message that is used when committing the changelog.
- Optional** `git-user-name`: The git user.name to use for the commit. Default `Conventional Changelog Action`
- Optional** `git-user-email`: The git user.email to use for the commit. Default `conventional.changelog.action@github.com`
- Optional** `git-pull-method`: The git pull method used when pulling all changes from remote. Default `rebase`

Choose a version

- v3.17.0
- v3.17.0
- v3
- v3.16.0
- v3.16.0
- v3.15.0
- v3.15.0
- v3.14.1
- v3.14.1
- v3.14.0**
- v3.14.0
- v3.13.0
- v3.13.0
- v3.12.0

[Open Issues](#) [Pull requests](#)

Then, click on the "Use v3.14.0" green button to get the code and click on the copy icon to use it.



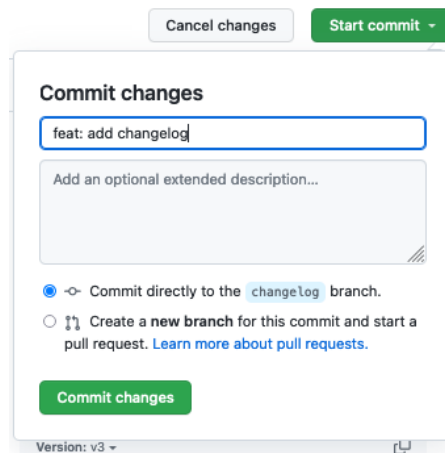
- Go back to the tab where you are editing the pipeline.yaml file under your secondary GitHub userid. Paste the usage info you just copied from the marketplace action as a new step in the build job **AFTER** the *checkout* step and **BEFORE** the step that *sets up JDK 11*. Pay attention to get the indentation right as shown below.

```
19 jobs:
20   build:
21
22     runs-on: ubuntu-latest
23
24     steps:
25       - uses: actions/checkout@v3
26
27       - name: Conventional Changelog Action
28         uses: TriPSs/conventional-changelog-action@v3.14.0
29
30       - name: Set up JDK 11
31         uses: actions/setup-java@v3
32         with:
33           java-version: '11'
```

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

```

- 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.

Cancel changes Start commit

**Commit changes**

fix: update branch name

Add an optional extended description...

☒ Commit directly to the `changelog` branch.

☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

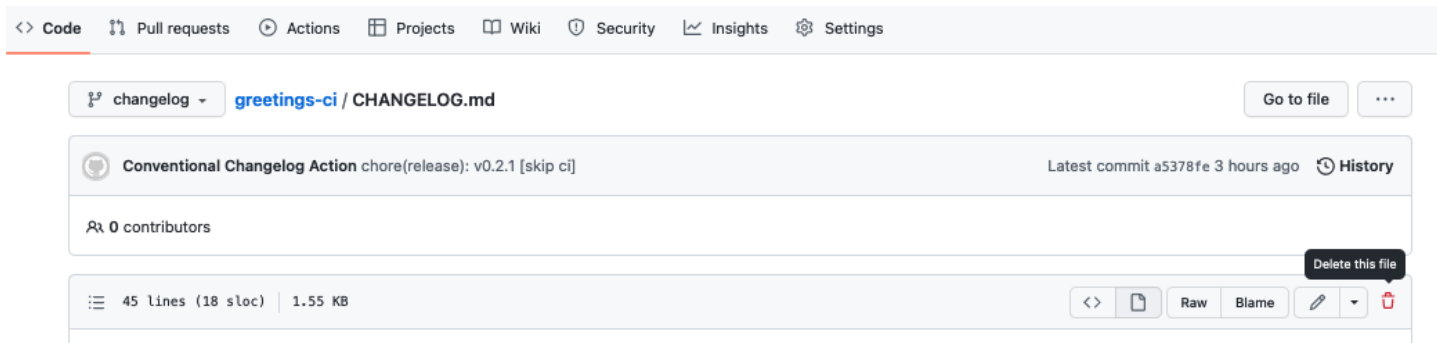
Commit changes

END OF LAB

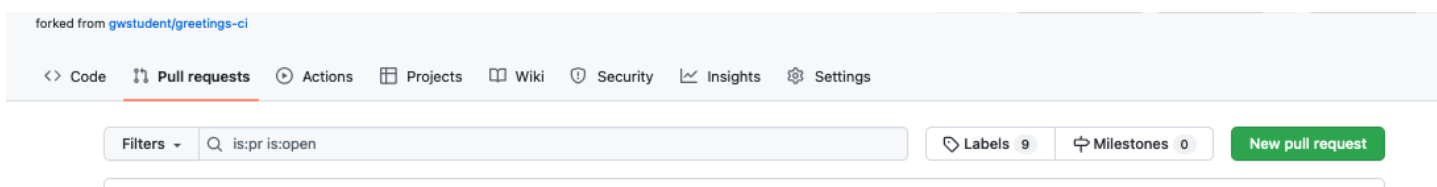
## Lab 5 – Pull requests from other users

**Purpose:** In this lab, we'll see how to take our new code and execute a pull request from another user.

- Now that we have our code changes for the changelog in our forked repository (under the secondary userid), let's see how to get them merged back into the original repository (primary userid) via another pull request – this time between two separate repositories.
- Still in the forked repository under your secondary GitHub userid and in the "changelog" branch, in preparation for the pull request, let's delete the CHANGELOG.md file so the target repo can create its own new one. Go back to the "Code" tab at the top of the repository and select the "CHANGELOG.md" file. Click on the trashcan icon on the far right in the gray bar above the text of the file. Then go ahead and commit those changes – just leave the commit message as-is.



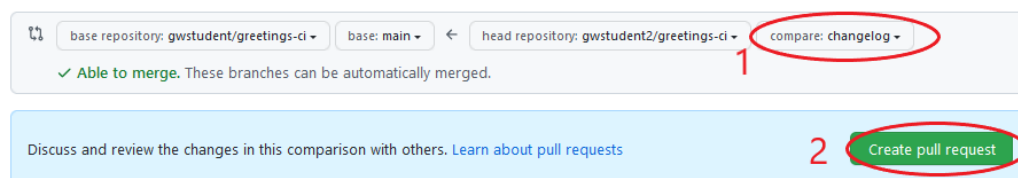
3. Also delete the package.json file via the same process.
4. In the forked repository for your secondary GitHub userid, select the “Pull requests” tab and then click on the “New pull request” button.



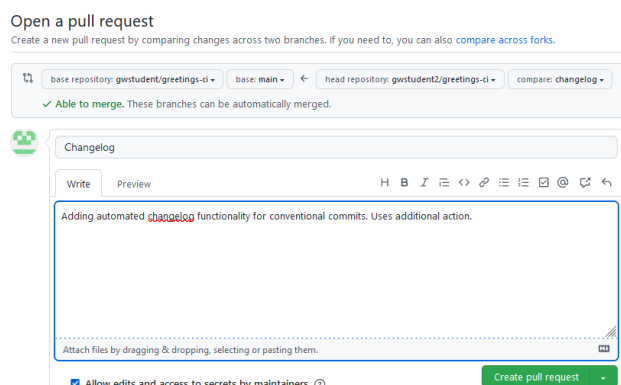
5. Note that this automatically drops you into a screen where you are comparing the main branch of your forked project from the secondary GitHub userid (referred to as the “head repository”) to the original project’s (primary userid) main branch (referred to as the “base repository”). In this case, since we made our changes in the “changelog” branch, we want to switch the branch in our forked project for the secondary GitHub user id (“head repository”) to be the “changelog” branch. You can click on the dropdown that says “compare: main” and select “changelog” in there. Then click on the “Create pull request” button.

### Comparing changes

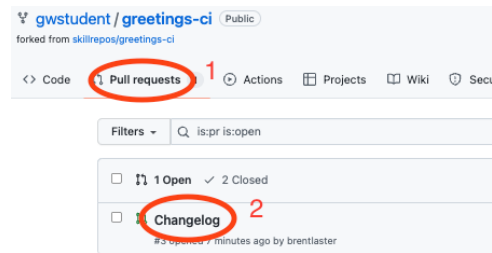
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).



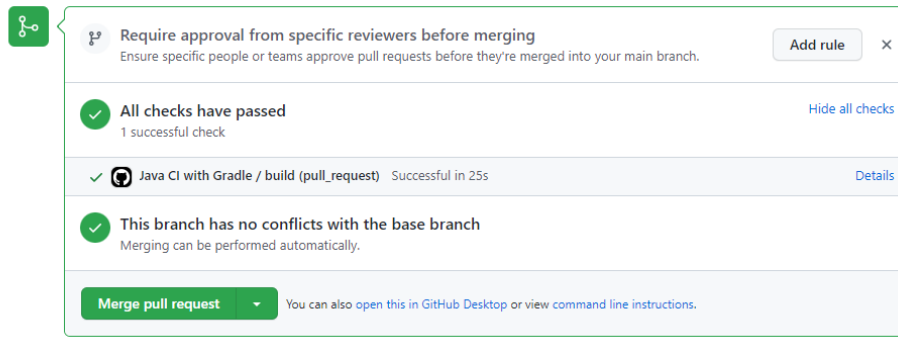
6. On the next screen, you can just enter an appropriate comment and then click the next “Create pull request” button. After this, you’ll see a screen that summarizes the pull request with tabs across the top to look at the commits, checks, and files that were changed.



- Now, go back to a session under your primary GitHub userid (log out and log back in if needed). Then go to the original repository for greetings-ci under that userid. Click on the “Pull requests” tab at the top and you should see 1 open pull request from your other userid. Click on the link for that one.



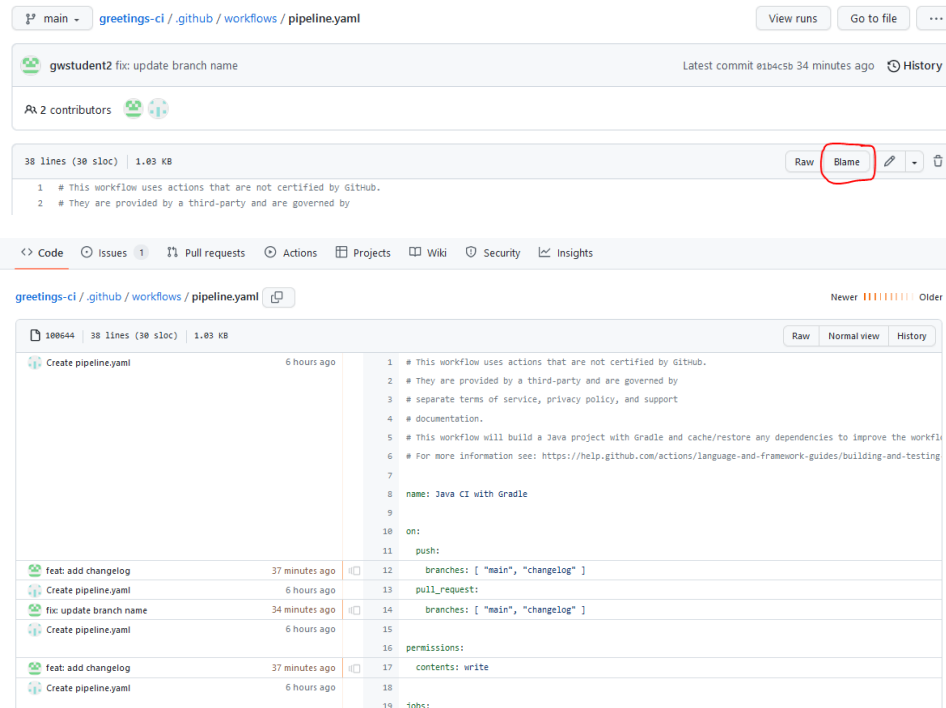
- Now you can scroll down, and eventually you should see a message that “All checks have passed”. You can click on the “See all checks” link to the right and see that the “check” here was a run of our workflow. Then you can click on the “Merge pull request” button and then click the “Confirm merge” button and the merge request should be completed.



- After the operation finishes, the workflow should have run and you should now see a CHANGELOG.md file and a package.json file in the list of files on the main branch of your original repo.

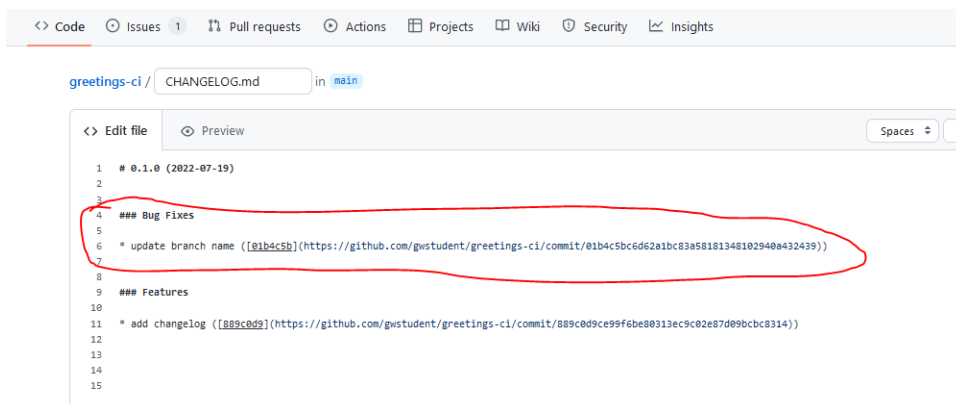
Conventional Changelog Action chore(release): v0.4.0	
.github/workflows	Update pipeline
extra	add extra dir
gradle/wrapper	Initial add
src/main/java	Initial add
CHANGELOG.md	chore(release):
build.gradle	Initial add
gradlew	Initial add
gradlew.bat	Initial add
package.json	chore(release):

10. Now, let's take a look at who made what changes in the workflow file. Go to the workflow file (pipeline.yaml) page. Instead of clicking the pencil icon to edit, click the "Blame" button. You should see a screen like the second screenshot below showing who made what changes.



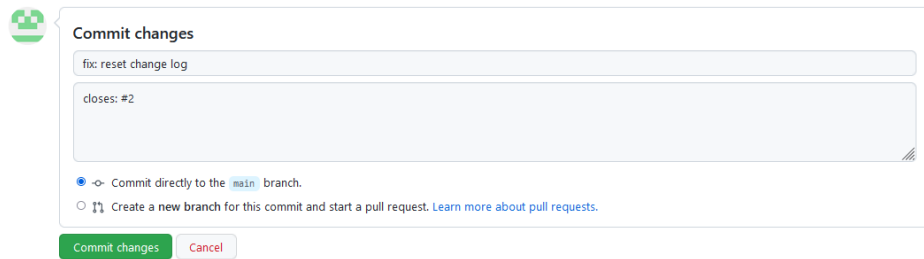
The screenshot shows the GitHub Blame page for the file `pipeline.yaml` in the repository `greetings-ci / .github / workflows`. The page displays the commit history for the file, with the 'Blame' button highlighted in a red circle. The commit history shows that the file was created by 'gwestudent2' 6 hours ago, and it includes a list of contributors and a table of changes.

11. Let's do one more fix for the repo. Open the CHANGELOG.md file and edit it. To make it a bit cleaner for this repository, let's remove the reference to the bug fix we made in the other repository. Delete the lines shown in the red circle below.



The screenshot shows the GitHub Edit file page for the file `CHANGELOG.md` in the repository `greetings-ci /`. The page displays the file content with a red circle highlighting the lines to be deleted. The file content includes a header, a section for Bug Fixes, and a section for Features.

12. At the bottom of the page, in the commit message area for the "Commit changes" box, enter "fix: reset change log". And in the "Add an optional extended description..." box, add the text "closes: #x" where "x" is the number of the issue that we opened earlier in the labs.

A screenshot of a "Commit changes" dialog box. It features a green and white icon on the left. The dialog has a title bar "Commit changes". Below the title bar, there are two text input fields. The first field contains the text "fix: reset change log". The second field contains the text "closes: #2". Below the input fields, there are two radio button options. The first option is selected and is labeled "Commit directly to the main branch.". The second option is labeled "Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)". At the bottom of the dialog, there are two buttons: "Commit changes" (green) and "Cancel" (red).

13. After that, click on the “Commit changes” button. This should run the action again and close the issue.
14. After this is done, look at the CHANGELOG.md file to see the fix increment and the issue (under the “Issues” tab) to see that it is closed.

END OF LAB