

# GitHub Actions Deep Dive

Revision 2.5 – 02/17/24

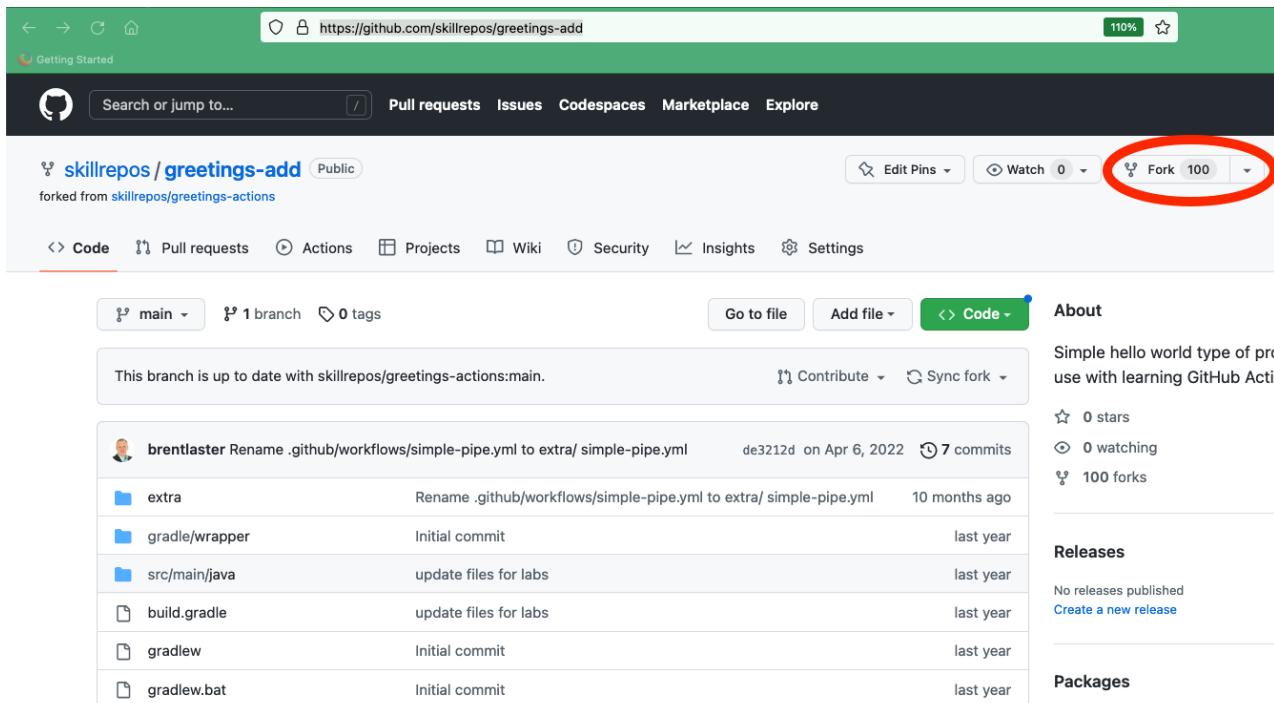
Tech Skills Transformations LLC / Brent Laster

*Important Prerequisite: You will need a GitHub account for this. (Free tier is fine.)*

## 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-add> and fork that project into your own GitHub space. You can accept the default options for the fork and click the "Create fork" button.



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.

skillrepos / greetings-add Public  
forked from skillrepos/greetings-actions

**Code** Pull request Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags Go to file Add file <> Code

This branch is up to date with skillrepos/greetings-actions:main.

Commit	Message	Date
brentlaster Rename .github/workflows/simple-pipe.yml to extra/ simple-pipe.yml	de3212d on Apr 6, 2022	7 commits
extra	Rename .github/workflows/simple-pipe.yml to extra/ simple-pipe.yml	10 months ago
gradle/wrapper	Initial commit	last year
src/main/java	update files for labs	last year
build.gradle	update files for labs	last year

- 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".

Automation

- Greetings By GitHub Actions Greets users who are first time contributors to the repo Configure Automation
- Stale By GitHub Actions Checks for stale issues and pull requests Configure Automation
- Manual workflow By GitHub Actions Simple workflow that is manually triggered. Configure Automation
- Labeler By GitHub Actions Labels pull requests based on the files changed Configure Automation

Browse all categories

- Automation
- Continuous integration**
- Deployment
- Security

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

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

**Code** Pull requests Actions Projects Wiki Security Insights Settings

### Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and set up a workflow yourself →

Categories

- Automation
- Continuous integration**
- Deployment
- Security

Q Gradle

Found 52 workflows

Android CI	Java
Java with Ant	Java
Clojure	Clojure
Publish Java Package	
Java with Gradle	
Publish Java Package	

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

The screenshot shows the GitHub Actions 'Get started with GitHub Actions' page. The search bar at the top contains 'Gradle'. On the left, there's a sidebar with categories: Automation, Deployment, Security, and Pages. Under 'Continuous integration', there are four workflow cards: 'Android CI' (By GitHub Actions), 'SLSA Generic generator' (By Open Source Security Foundation (OpenSSF)), 'Publish Java Package with Gradle' (By GitHub Actions), and 'Java with Gradle' (By GitHub Actions). The 'Java with Gradle' card is circled in red, and its 'Configure' button is also circled.

7. This will bring up a page with a starter workflow for CI that we can edit as needed. There are three edits we need to make here. The first is to change the name of the file. 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 image contains two screenshots of a GitHub code editor. The top screenshot shows the file path 'greetings-actions/.github/workflows/gradle.yml'. The bottom screenshot shows the file path 'greetings-add/.github/workflows/pipeline.yml'. Both screenshots show the same workflow content, which includes a header indicating it uses uncertified actions and provides third-party governed terms of service, privacy policy, and support.

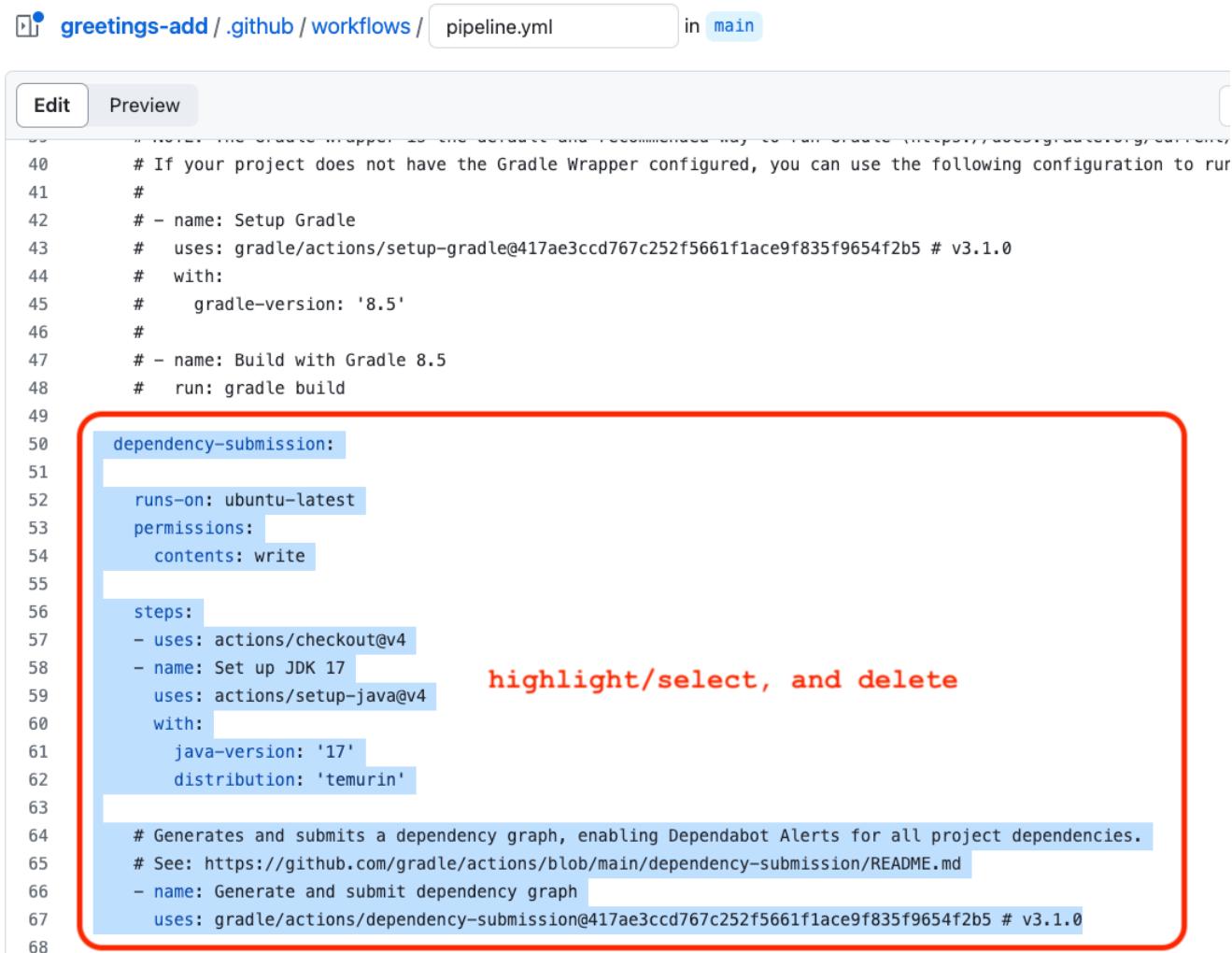
8. Now, edit the name of the workflow - change line 8 from "name: Java CI with Gradle" to "name: Simple Pipe".

```

5 # This workflow will build a Java project 5 # This workflow will build a Java project
6 # For more information see: https://docs. 6 # For more information see: https://docs.
7 7
8 name: Java CI with Gradle 8 name: Simple Pipe
9 9
10 on: 10 on:
11   push:

```

9. The third 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.



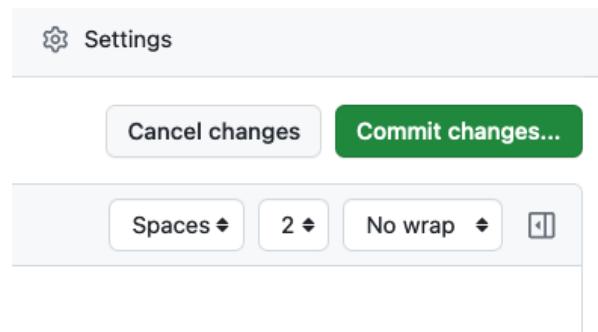
The screenshot shows the GitHub Actions pipeline configuration file (`pipeline.yml`) in the `main` branch of the `greetings-add` repository. The `dependency-submission:` section is highlighted with a red border. Inside this section, the `steps:` block is also highlighted. A red annotation **highlight/select, and delete** is placed over the `steps:` block. The code in the `dependency-submission:` section is as follows:

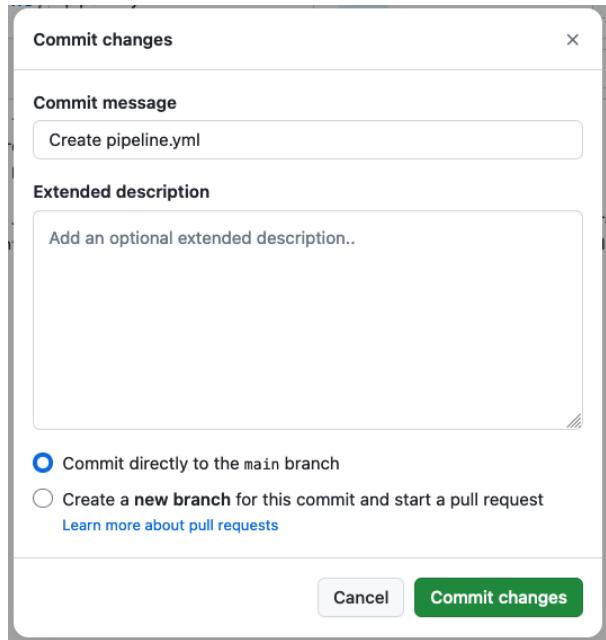
```

40     # If your project does not have the Gradle Wrapper configured, you can use the following configuration to run
41     #
42     # - name: Setup Gradle
43     #   uses: gradle/actions/setup-gradle@417ae3ccd767c252f5661f1ace9f835f9654f2b5 # v3.1.0
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@417ae3ccd767c252f5661f1ace9f835f9654f2b5 # v3.1.0
68

```

10. 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.





11. 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.

All workflows	
<b>1 workflow run</b>	Event ▾ Status ▾ Branch ▾ Actor ▾
<b>Create pipeline.yml</b> Simple pipe #1: Commit 2cac6a7 pushed by gwstudent	main 1 minute ago ... 35s

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 (next to the green check) for the run to get to the details for that run.

All workflows	
<b>1 workflow run</b>	Event ▾ Status ▾ Branch ▾ Actor ▾
<b>Create pipeline.yml</b> Simple pipe #1: Commit 2cac6a7 pushed by gwstudent	main 2 minutes ago ... 35s

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 interface for a repository named "greetings-add". The "Actions" tab is selected. A workflow named "Create pipeline.yml #1" is shown with a single job named "build" that has succeeded. The job log shows the steps: "Set up job" (2s) and "Run actions/checkout@v3" (1s). The log output includes commands like "Run actions/checkout@v3", "Syncing repository: gwstudent/greetings-add", "Getting Git version info", and "Temporarily overriding HOME='/home/runner/work/\_temp/9ff6fe06-369f-42ba-9e36-9564f8e973de'". A red circle highlights the "Run actions/checkout@v3" step.

\* END OF LAB \*

## Lab 2 – Learning more about Actions

**Purpose:** In this lab, we'll see how to find and use additional actions as well as persist artifacts.

1. We're going to explore one way in GitHub to update a workflow and add additional actions into it. Start out by opening up the workflow file pipeline.yml. There are multiple ways to get to it but let's open it via the Actions screen.

In your GitHub repository, click the Actions button at the top if not already on the Actions screen.

Under "All workflows", select the "Simple Pipe" workflow.

After that, select the "pipeline.yml" link near the middle top.

The screenshot shows the GitHub Actions main screen for the "greetings-add" repository. The "Actions" tab is selected. On the left, under "Actions", the "Simple pipe" workflow is selected. A red circle highlights the "pipeline.yml" link in the "Simple pipe" section. On the right, a summary of the workflow shows "1 workflow run" for the "Create pipeline.yml" step, which was triggered by a commit from the "main" branch 4 hours ago. The status is "Success" with 35s duration.

- Once the file opens up, click on the pencil icon (1) in the top right to edit it.

```

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: Simple Pipe
9
10 on:
11   push:
12     branches: [ "main" ]
13   pull_request:
14     types: [ "pull_request" ]

```

- You'll now see the file open up in the editor, but also to the right, you should see a new pane with references to the GitHub Actions Marketplace and Documentation. 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.

Next, click on the "Upload a Build Artifact" item. 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".

**Marketplace** Documentation

[Upload build](#)

Marketplace / Search results

Your search has returned 10 results.

- Upload a Build Artifact** By actions 2.7k

Upload a build artifact that can be used by subsequent workflow steps
- Upload build to Autify for Mobile** By autifyhq 3

Upload a build file to Autify for Mobile
- Tar and Upload a Build Artifact** By eviden-actions 1

Pack files in a tar archive and upload a build artifact that can be used by subsequent workflow steps
- Jira Upload Build Info** By HighwayThree 8

Github Action to upload build info associated with a Jira issue key to Jira Software REST API

**Marketplace** Documentation

Marketplace / Search results / Upload a Build Artifact

### Upload a Build Artifact

By actions v3.0.0 1.5k

Upload a build artifact that can be used by subsequent workflow steps

[View full Marketplace listing](#)

---

**Installation**

Copy and paste the following snippet into your .yml file.

Version: v3.0.0

```

- name: Upload a Build Artifact
  uses: actions/upload-artifact@v3.0.0
  with:
    # Artifact name
    name: # optional, default is artifact
    # A file, directory or wildcard pattern
    path:

```

- This should open up the full GitHub Actions Marketplace listing for this action. Notice the URL at the top - <https://github.com/marketplace/actions/upload-a-build-artifact>. You can use this same relative URL to see

other actions that are in the marketplace. For example, let's look at the checkout one we're already using. Go to <https://github.com/marketplace/actions/checkout>

Then click on the "actions/checkout" link under "Links" in the lower right.

The screenshot shows the GitHub Marketplace page for the "Checkout" GitHub Action. The action icon is a blue circle with a white play button. The title is "GitHub Action Checkout". Below it, it says "v3.0.0" and "Latest version". To the right, there is a "Use latest version" button. On the left, there is a "test-local" badge with "passing". The main content area is titled "Checkout V3" and contains the following text:  
This action checks-out your repository under `$GITHUB_WORKSPACE`, so your workflow can access it.  
Only a single commit is fetched by default, for the ref/SHA that triggered the workflow. Set `fetch-depth: 0` to fetch all history for all branches and tags. Refer [here](#) to learn which commit `$GITHUB_SHA` points to for different events.  
The auth token is persisted in the local git config. This enables your scripts to run authenticated git commands. The token is removed during post-job cleanup. Set `persist-credentials: false` to opt-out.  
When Git 2.18 or higher is not in your PATH, falls back to the REST API to download the files.

## What's new

- Updated to the node16 runtime by default
  - This requires a minimum [Actions Runner](#) version of v2.205.0 or later, which is by default

Links

- [actions/checkout](#) (highlighted with a red oval)
- [Open issues](#)
- 209

5. This will put you on the screen for the source code for this GitHub Action. Notice there is also an Actions button here. GitHub Actions use workflows that can use other GitHub Actions. Click on the Actions button to see the workflows that are in use/available.

The screenshot shows the GitHub Actions page for the "actions/checkout" action. At the top, there is a navigation bar with "Pull requests", "Issues", "Marketplace", "Explore", and an "Actions" button, which is highlighted with a red oval. Below the navigation bar, there is a section titled "Use this GitHub Action with your project" with a "View on Marketplace" button. The main content area shows the repository structure:  
main · 35 branches · 16 tags  
brcrista · Create check-dist.yml (#566) ... · afe4af0 · 14 days ago · 85 commits  
.github/workflows · Create check-dist.yml (#566) · 14 days ago  
.licenses/npm · Add Licensed To Help Verify Prod Licenses (#326) · 12 months ago  
\_\_test\_\_ · Swap to Environment Files (#360) · 11 months ago  
adrs · update default branch (#305) · 14 months ago

The screenshot shows the GitHub Actions page for a repository. On the left, there's a sidebar with 'Workflows' and a 'All workflows' tab selected. Below it are categories like 'Azure Static Web Apps CI/CD', 'Build and Test', 'Check dist', and 'Licensed'. On the right, under 'All workflows', it says 'Showing runs from all workflows' and '584 workflow runs'. There are two recent runs listed:

- Create check-dist.yml (#566)**: Build and Test #532: Commit afe4af0 pushed by thboop. Status: main. 14 days ago, 3m 39s.
- Create check-dist.yml (#566)**: Licensed #62: Commit afe4af0 pushed by thboop. Status: main. 14 days ago, 45s.

- 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 after, and in line with, the build job steps (after the "Build with Gradle Wrapper" step). Pay attention to the indenting. If you see red wavy lines under your code, that likely means the indenting is off. See the screenshot (lines 39-43) for how this should look afterwards. (Your line numbers may be different.)

```

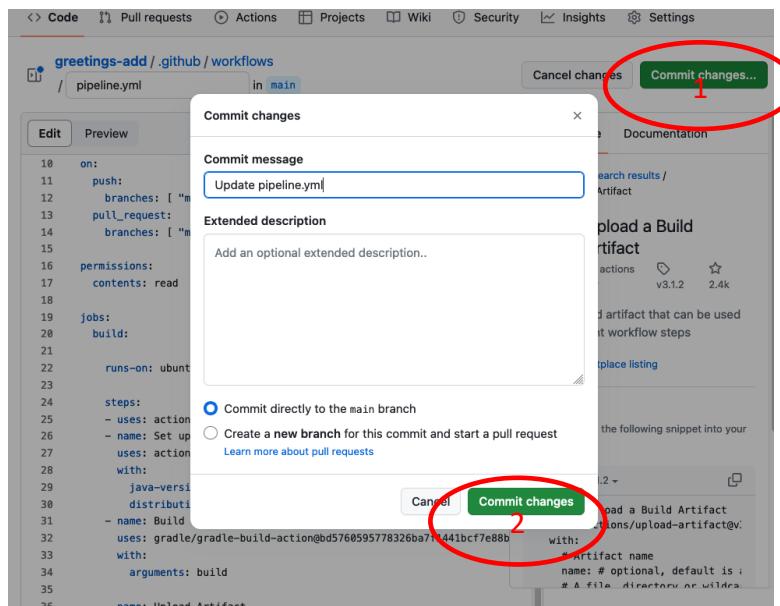
- name: Upload Artifact
  uses: actions/upload-artifact@v4.3.0
  with:
    name: greetings-jar
    path: build/libs

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
48

```

7. Click on the green "Commit changes" button in the upper right. In the dialog that comes up, add a different commit message if you want, then click the green "Commit changes" button to make the commit.



8. 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 "Update pipeline.yml" (or whatever your commit message was). On the next screen, in addition to the graph, there will be a new section called "Artifacts" near the bottom. You can download the artifact from there. Click on the name of the artifact to try this.

Root Project	Requested Tasks	Gradle Version	Build Outcome	Build Scan™
greetings-add	build	4.10	<span style="color: green;">✓</span>	<span style="color: grey;">Build Scan™ NOT PUBLISHED</span>

\* END OF LAB \*

## Lab 3 – Alternative ways to invoke workflows

Purpose: In this lab, we'll see how to add a different kind of event trigger that allows us to invoke the workflow manually

1. Let's make a change to make it easier to run our workflow manually to try things out, start runs, etc. We are going to add two input values - one for the version of the artifact we want to create and use and one for input values to pass to a test. Edit the pipeline.yaml file again. 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:  
    myVersion:  
      description: 'Input Version'  
    myValues:  
      description: 'Input Values'
```

The screenshot shows the GitHub code editor interface for a file named 'pipeline.yaml'. The file content is as follows:

```
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://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-  
7  
8 name: Simple Pipe  
9  
10 on:  
11   push:  
12     branches: [ "main" ]  
13   pull_request:  
14     branches: [ "main" ]  
15   workflow_dispatch:  
16     inputs:  
17       myVersion:  
18         description: 'Input Version'  
19       myValues:  
20         description: 'Input Values'  
21
```

The 'Commit changes...' button is highlighted in green at the top right of the editor.

2. Now let's add a step to our build job to get the timestamp to use to version the artifact. Add the step below AFTER the build step and BEFORE the upload step.

```
- name: Set timestamp  
  run: echo TDS=$(date +'%Y-%m-%dT%H-%M-%S') >> $GITHUB_ENV
```

3. Next add another step to "tag" the artifact with the input version (if there is one) and also the timestamp. Add this step right after the previous one and BEFORE the upload step. (Note that the second and third lines here are meant to be a single line when you put them in the yaml file.)

```

- name: Tag artifact
  run: mv build/libs/greetings-add.jar build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}.jar

```

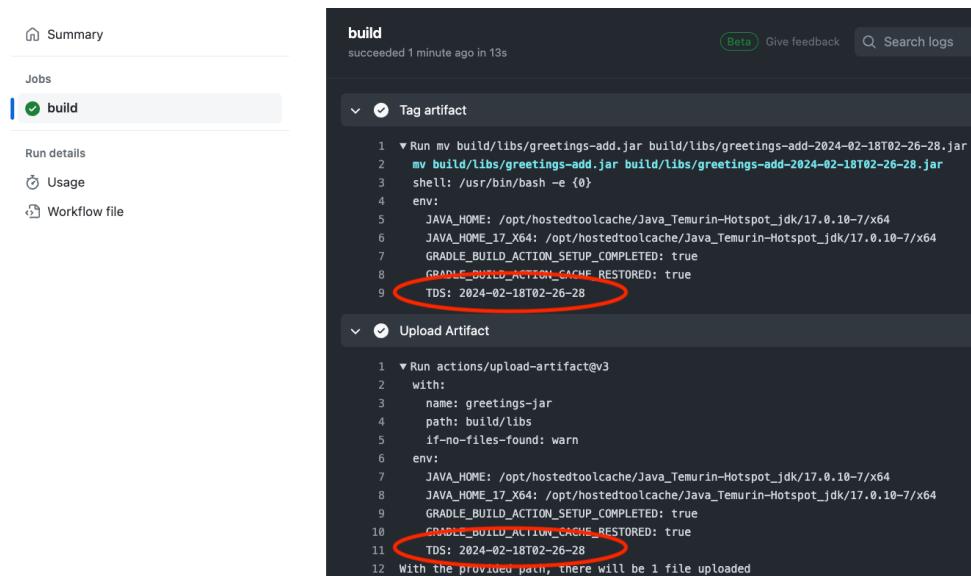
The figure below shows the steps added in the workflow. (Your line numbers may be different.)

```

46   - name: Build with Gradle Wrapper
47     run: ./gradlew build
48
49   - name: Set timestamp
50     run: echo TDS=$(date +'%Y-%m-%dT%H-%M-%S') >> $GITHUB_ENV
51
52   - name: Tag artifact
53     run: mv build/libs/greetings-add.jar build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}.jar
54
55   - name: Upload Artifact
56     uses: actions/upload-artifact@v4.3.0
57     with:

```

4. Go ahead and commit the changes to the main branch. After this runs, you can look at the logs by clicking on the Actions menu, then in the workflow runs list, click on the commit message for the particular run and then on the job itself. On the right-hand side, click on the downward pointing arrows next to the "Tag artifact" and "Upload Artifact" and expand them to see the individual steps. Notice the *TDS* variable we defined as part of the environment.



5. The workflow\_dispatch code we added to the event trigger sections created a way to manually run the workflow and also pass in values for the parameters we defined. To see how to run the workflow manually, click on the main Actions menu (if not already there), then select the "Simple pipe" workflow on the lefthand side. At that point you should see a "Run workflow" button on the far right at the top of the runs list. Click on that button and enter any numeric value you want for the Input version and then any Input values you want to supply. Then click on "Run workflow".

Actions

Simple pipe

pipeline.yml

20 workflow runs

This workflow has a `workflow_dispatch` event trigger.

**Simple pipe** Simple pipe #20: Manually run by gwstudent

**Update pipeline.yml** Simple pipe #19: Commit 2e3c0c1 pushed by gwstudent

**Update pipeline.yml** Simple pipe #18: Commit ac22d5e pushed by gwstudent

**Update pipeline.yml** Failed: Commit 2e3c0c1 pushed by gwstudent

Run workflow

Use workflow from Branch: main

Input Version 1.0.0

Input Values val1 val2

Run workflow

26 minutes ago

- After this run, you can select the run from the runs list, scroll down and find the greetings.jar artifact and click on it to download it. Once you have it downloaded, you can uncompress the artifact and you should see a jar file with the version you entered for "Input Version" and the time-date stamp.

Name	Size	Kind	Date Added
greetings-add-1.0.0-2023-01-30T01-17-11.jar	1 KB	Java JAR file	Today at 8:20 PM
greetings-add-1.0.02023-01-30T01-13-37.jar	1 KB	Java JAR file	Today at 8:14 PM
greetings-add-2023-01-30T00-52-59.jar	1 KB	Java JAR file	Today at 7:53 PM
greetings-add.jar	1 KB	Java JAR file	Today at 6:37 PM
greetings-ci-0.3.0.jar	1 KB	Java JAR file	Nov 5, 2022 at 6:24 AM
greetings-ci-0.3.0.jar	1 KB	Java JAR file	Aug 14, 2022 at 8:41 AM
greetings-ci-0.4.0.jar	1 KB	Java JAR file	Aug 23, 2022 at 9:46 PM
greetings-ci-0.6.0.2.jar	1 KB	Java JAR file	Aug 8, 2022 at 8:32 PM
greetings-ci-0.6.0.3.jar	1 KB	Java JAR file	Aug 8, 2022 at 8:32 PM
greetings-ci-0.6.0.jar	1 KB	Java JAR file	Aug 4, 2022 at 8:52 PM
greetings-jar(4).zip	1 KB	ZIP archive	Today at 7:44 PM
greetings-jar (1).zip	1 KB	ZIP archive	Nov 5, 2022 at 6:24 AM
greetings-jar (2).zip	1 KB	ZIP archive	Nov 5, 2022 at 6:24 AM
greetings-jar(1).zip	1 KB	ZIP archive	Nov 6, 2022 at 9:44 PM
greetings-jar(5).zip	1 KB	ZIP archive	Today at 7:53 PM
greetings-jar(6).zip	1 KB	ZIP archive	Today at 8:14 PM
greetings-jar(7).zip	1 KB	ZIP archive	Today at 8:20 PM

greetings-add-1.0.0-2023-01-30T01-17-11.jar  
Java JAR file - 1 KB  
Information  
Created Today at 8:20 PM  
More...

## Lab 4 – Sharing output between jobs

Purpose: In this lab, we'll see how to capture output from one job and share it with another one

- Let's add one more piece to this job so we can have the path of the jar file available for other jobs. To do this, edit the workflow file and add a step at the end of the job to set the output value.

```
- name: Set output
  id: setoutput
  run: echo jarpath=build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}.jar >> $GITHUB_OUTPUT
```

```

50   run: echo $TDS=$(date + "%Y-%m-%d-%H-%M-%S") >> $GITHUB_ENV
51
52   - name: Tag artifact
53     run: mv build/libs/greetings-add.jar build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}.jar
54
55   - name: Upload Artifact
56     uses: actions/upload-artifact@v4.3.0
57     with:
58       name: greetings-jar
59       path: |
60         build/libs
61         test-script.sh
62
63   - name: Set output
64     id: setoutput
65     run: echo jarpath=build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}.jar >> $GITHUB_OUTPUT
66
67

```

- Now we need to add an "outputs" section BETWEEN the "runs-on:" and the "steps:" section near the top of the "build" job. This will map the variable "artifact-path" to the outputs of the previous step.

```

# Map a step output to a job output
outputs:
  artifact-path: ${{ steps.setoutput.outputs.jarpath }}

```

The screenshot shows the GitHub Copilot interface with the pipeline.yml file open. On the left, the file structure is shown with .github/workflows/pipeline.yml selected. On the right, the code editor shows the workflow definition. A blue box highlights the new 'outputs' section added between the 'jobs:' and 'steps:' sections.

```

21
22
23   jobs:
24     build:
25
26       runs-on: ubuntu-latest
27       permissions:
28         contents: read
29
30       # Map a step output to a job output
31       outputs:
32         artifact-path: ${{ steps.setoutput.outputs.jarpath }}
33

```

- In order to verify that we can see the output from the build job, add a new, second job in the workflow file. (You can just add it at the bottom.). Copy and paste the simple job below that echoes out the output value from the build job. Note that "print-build-output" should line up with the "build" title of the first job.

```

print-build-output:
  runs-on: ubuntu-latest
  needs: build
  steps:
    - run: echo ${{needs.build.outputs.artifact-path}}

```

```

0.1
62   - name: Set output
63     id: setoutput
64     run: echo jarpath=build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}.
65
66
67   # NOTE: The Gradle Wrapper is the default and recommended way to run Gradle (https://docs.gradle.org)
68   # If your project does not have the Gradle Wrapper configured, you can use the following configu
69   #
70   # - name: Setup Gradle
71   #   uses: gradle/actions/setup-gradle@417ae3ccd767c252f5661f1ace9f835f9654f2b5 # v3.1.0
72   #   with:
73   #     gradle-version: '8.5'
74   #
75   # - name: Build with Gradle 8.5
76   #   run: gradle build
77
78   print-build-output:
79     runs-on: ubuntu-latest
80     needs: build
81     steps:
82       - run: echo ${needs.build.outputs.artifact-path}

```

4. Commit the changes. After the workflow runs, you should see two jobs in the graph for the workflow run - one for "build" and one for "print-build-output". Click on the "print-build-output" one and you can see from the logs that it was able to print the output value created from the "build" job.

**Workflow Summary:**

- Re-run triggered 1 minute ago
- Status: Success
- Total duration: 37s
- Artifacts: 1

**print-build-output Job Log:**

```

print-build-output
succeeded now in 2s

> ⚡ Set up job
> ⚡ Run echo build/libs/greetings-add-2023-01-30T02-37-36.jar
1 ▶ Run echo build/libs/greetings-add-2023-01-30T02-37-36.jar
2 echo build/libs/greetings-add-2023-01-30T02-37-36.jar
3 shell: /usr/bin/bash -e {0}
4 build/libs/greetings-add-2023-01-30T02-37-36.jar

> ⚡ Complete job

```

## Lab 5: Adding in a test case

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

1. First, let's create a script to test our code. The code for the test script we'll use is already in the "extra/test-script.sh" file. Go back to the "Code" tab, open up that file (in the "extra" subdirectory) and click the pencil icon.

The screenshot shows the GitHub Code editor interface. The repository is 'greetings-add'. The 'extra' directory contains several files: .github, simple-pipe.yml, count-args.txt, create-failure-issue.yml, create-issue-on-failure.txt, info.txt, test-run-old.txt, test-run.txt, and test-script.sh. The 'test-script.sh' file is selected and shown in the main editor area. The code is:

```
1 # Simple test script for greetings jar
2
3 set -e
4
5 java -jar $1 ${@:2} > output.bench
6 IFS=' ' read -ra ARR <<< "${@:2}"
7 for i in "${ARR[@]}"; do
8     grep "^$i$" output.bench
9 done
```

2. In the editor, all you need to do is change the path of the file. Click in the text entry area for "test-script.sh" and backspace over the "extra" path so that the file is directly in the "greetings-add" directory (root repo directory).

The screenshot shows the GitHub Code editor interface. The repository is 'greetings-add'. The 'test-script.sh' file is selected and shown in the main editor area. The code is identical to the previous screenshot, but the file path in the navigation bar is now 'greetings-add / test-script.sh'.

```
1 # Simple test script for greetings jar
2
3 set -e
4
5 java -jar $1 ${@:2} > output.bench
6 IFS=' ' read -ra ARR <<< "${@:2}"
7 for i in "${ARR[@]}"; do
8     grep "^$i$" output.bench
9 done
10
```

3. This script takes the path to 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.
4. Go ahead and commit this file into the repository for the path change.
5. Now let's add a third job to our workflow (in pipeline.yml) to do a simple "test". As you've done before, edit the *pipeline.yml* file.

6. Add the job definition for a job called "test-run" that runs on ubuntu-latest. You can copy and paste this code from **extra/test-run.txt** or grab it from the next page.

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.)

```
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 ${{ needs.build.outputs.artifact-path }} ${{ github.event.inputs.myValues }}
```

(This code is also available at <https://gist.github.com/gwstudent/9555f301f1888f888a8eb86fed4ffc8d>)

Edit Preview Code 55% faster with GitHub Copilot Space

```

78
79   print-build-output:
80     runs-on: ubuntu-latest
81     needs: build
82     steps:
83       - run: echo ${{needs.build.outputs.artifact-path}}
84
85   test-run:
86
87     runs-on: ubuntu-latest
88     needs: build
89
90     steps:
91       - name: Download candidate artifacts
92         uses: actions/download-artifact@v4
93         with:
94           name: greetings-jar
95
96       - name: Set up JDK 17
97         uses: actions/setup-java@v4
98         with:
99           java-version: '17'
100          distribution: 'temurin'
101
102      - name: Execute test
103        shell: bash
104        run: |
105          chmod +x ./test-script.sh
106          ./test-script.sh ${{ needs.build.outputs.artifact-path }} ${{ github.event.inputs.myValues }}
107

```

7. 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. Scroll back up, find the "Upload Artifact" step in the "build" job. Modify the **path** section of the "Upload Artifact" step to change from "path: build/libs" to look like below.

```

path: |
  build/libs
  test-script.sh

```

```

51
52   - name: Tag artifact
53     run: mv build/libs/greetings-add.jar build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}.jar
54
55   - name: Upload Artifact
56     uses: actions/upload-artifact@v4.3.0
57     with:
58       name: greetings-jar
59       path: |
60         build/libs
61         test-script.sh
62
63   - name: Set output
64     id: setoutput
65     run: echo jarpath=build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}.jar >> $GITHUB_OUTPUT

```

8. Now, you can just commit the pipeline changes with a simple message like "Add testing to pipeline".

9. Afterwards, you should see a new run of the action showing multiple jobs in the action run detail. Notice that we can select and drill into each job separately.

The screenshot shows the GitHub Actions pipeline summary for a run triggered via push. The run details are as follows:

- Status:** Success
- Total duration:** 33s
- Artifacts:** 1

The pipeline consists of three jobs connected sequentially:

```

graph LR
    build[build] -- "14s" --> printBuildOutput[print-build-output]
    printBuildOutput -- "0s" --> testRun[test-run]
    
```

**Jobs:**

- build
- print-build-output
- test-run

**Run details:**

- Usage: 0s
- Workflow file: pipeline.yml

## Lab 6: Adding your own action

**Purpose:** in this lab, we'll see how to create and use a custom GitHub Action.

- First, we'll fork the repo for a simple action that displays a count of the arguments passed into a function. Go to <https://github.com/skillrepos/arg-count-action> and then Fork that repository into your own GitHub space. (You can just accept the default selections on the page.)

The screenshot shows the GitHub repository page for `skillrepos/arg-count-action`. The repository has been forked from `gwstudent/arg-count-action`. The main branch is `main`.

**Actions:**

- Publish this Action to Marketplace**: Make your Action discoverable on GitHub Marketplace and in GitHub search.
- Draft a release**: Create a new draft release.

**About:** Simple GitHub Action demo

**Releases:** 8 tags

**Packages:** No packages published

**Commits:**

- techupskills Delete iterate-args.sh ... (1 hour ago)
- action.yml Update action.yml (2 days ago)
- count-args.sh Initial commit (3 days ago)

2. In your fork of the repository, look at the files here. We have a one-line shell script (for illustration) to return the count of the arguments - "count-args.sh." And we have the primary logic for the action in the "action.yml" file.

Take a look at the action.yml file and see if you can understand what its doing. The syntax is similar to what we've seen in our workflow up to this point.

3. Switch back to the file for your original workflow (go back to the greetings-add project and edit the *pipeline.yaml* file in *.github/workflows*. Let's add the code to use this custom action to report the number of arguments passed in. Edit the file and add the code shown below at the end (again indenting the first line to align with the other job names). (For convenience, this code is also in "greetings-add/extracount-args.txt".) **For now, just leave the text exactly as is so we can see what errors look like.**

**count-args:**

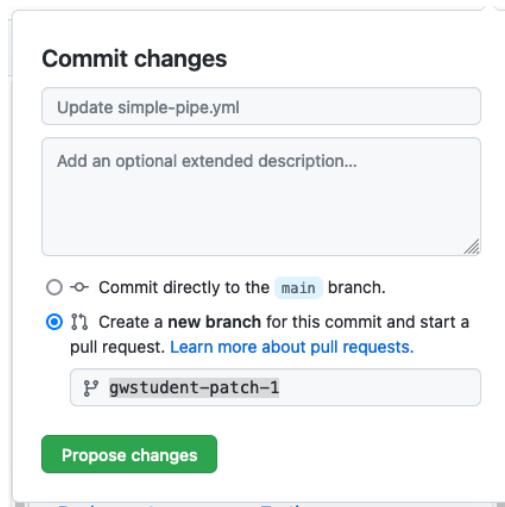
```
runs-on: ubuntu-latest

steps:
- id: report-count
  uses: <your github userid>/arg-count-action@main
  with:
    arguments-to-count: ${{ github.event.inputs.myValues }}
- run: echo
- shell: bash
  run: |
    echo argument count is ${{ steps.report-count.outputs.arg-count }}
```

```
102      - name: Execute test
103        shell: bash
104        run: |
105          chmod +x ./test-script.sh
106          ./test-script.sh ${{ needs.build.outputs.artifact-path }} ${{ github.event.inputs.myValues }}
107
108      count-args:
109
110      runs-on: ubuntu-latest
111
112      steps:
113      - id: report-count
114        uses: <your github userid>/arg-count-action@main
115        with:
116          arguments-to-count: ${{ github.event.inputs.myValues }}
117        - run: echo
118        - shell: bash
119        run: |
120          echo argument count is ${{ steps.report-count.outputs.arg-count }}
```

In this case, we call our custom action (<your github repo/arg-count-action>), using the latest from the main branch.

- Let's use a pull request to merge this change. Click on the green "Commit changes..." button, but in the "Commit changes" dialog, click on the bottom option to "Create a new branch for this commit and start a pull request." Change the proposed branch name if you want and then click on "Propose changes".



- In the next screen, click on the "Create pull request" button. In the following screen, update the comment if you want and then click on the "Create pull request" button. You'll then see it run through the jobs in our workflow as prechecks for merging.

The first screenshot shows a pull request comparison between 'base: main' and 'compare: patch-1'. It indicates that the branches are 'Able to merge'. Below the comparison are summary statistics: 1 commit, 1 file changed, and 1 contributor. A large green 'Create pull request' button is prominent.

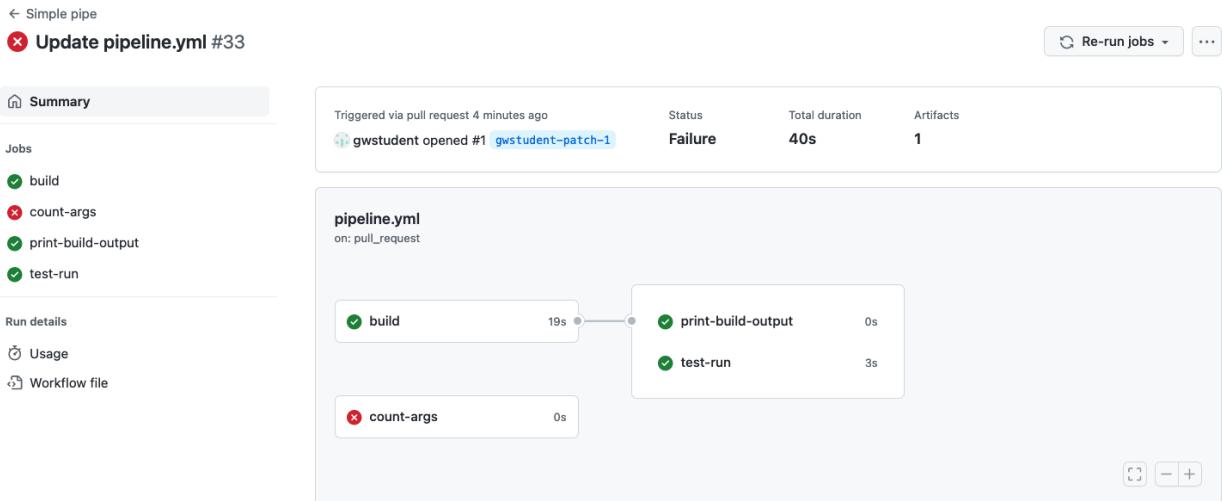
The second screenshot shows the 'Create pull request' dialog. It includes fields for updating files (e.g., 'simple-pipe.yml'), writing a comment, and previewing the changes. A rich text editor toolbar is visible above the comment area. At the bottom is a large green 'Create pull request' button.

6. When the checks are done running, you'll see one with a failure. Click on the link for "Details" on the right of the line with the failure to see the logs that are available. You can then see the error at the bottom of the log.

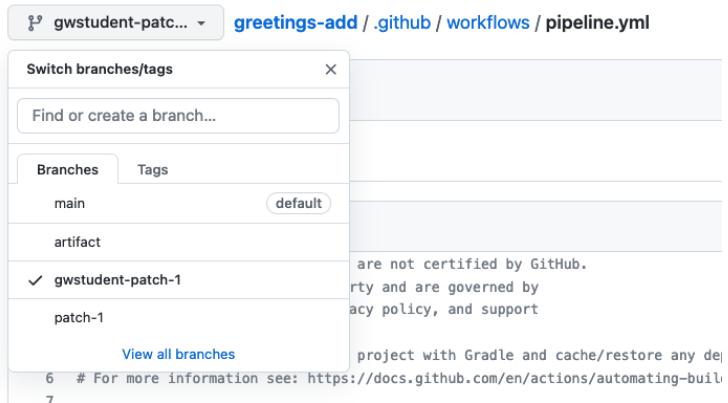
The screenshot shows a GitHub Actions pipeline status page for a pull request. At the top, it says "Update pipeline.yml #1" and "gwstudent wants to merge 1 commit into main from gwstudent-patch-1". A green "Open" button is visible. Below this, a summary box indicates "Some checks were not successful" with 3 successful and 1 failing check. The failing check is "Simple pipe / count-args (pull\_request)" with a red X icon. To the right of this line, there is a blue "Details" link with a circled arrow. Other successful checks listed are "Simple pipe / build (pull\_request)", "Simple pipe / print-build-output (pull\_request)", and "Simple pipe / test-run (pull\_request)". Below the summary, a message states "This branch has no conflicts with the base branch" and "Merging can be performed automatically". At the bottom, there is a "Merge pull request" button and a note about opening the pull request in GitHub Desktop or viewing command line instructions.

The screenshot shows the "count-args" job details page for the pull request. The left sidebar lists other jobs: "build" (green checkmark), "count-args" (red X), "print-build-output" (green checkmark), and "test-run" (green checkmark). The "Summary" link is highlighted with a red circle. The main pane shows the "Set up job" step with a log output. The log shows the runner version, operating system, runner image, runner image provisioner, GitHub token permissions, secret source, workflow preparation, action preparation, and finally an error at step 20: "Error: Unable to resolve action `<your github userid>/arg-count-action@main` , repository not found".

7. In the left column, click on the "Summary" link. This will take you back to the main graph page where you can also see the error.



8. So, before we can merge the PR, we need to fix the code. Go back to the "Actions" tab at the top, select the "Simple Pipe" workflow on the left, and then select the **pipeline.yaml** file (link above the list of workflow runs - if not already there) and **switch to the patch branch that you created for the pull request**. (Alternatively, you can select the file via the Code tab.)



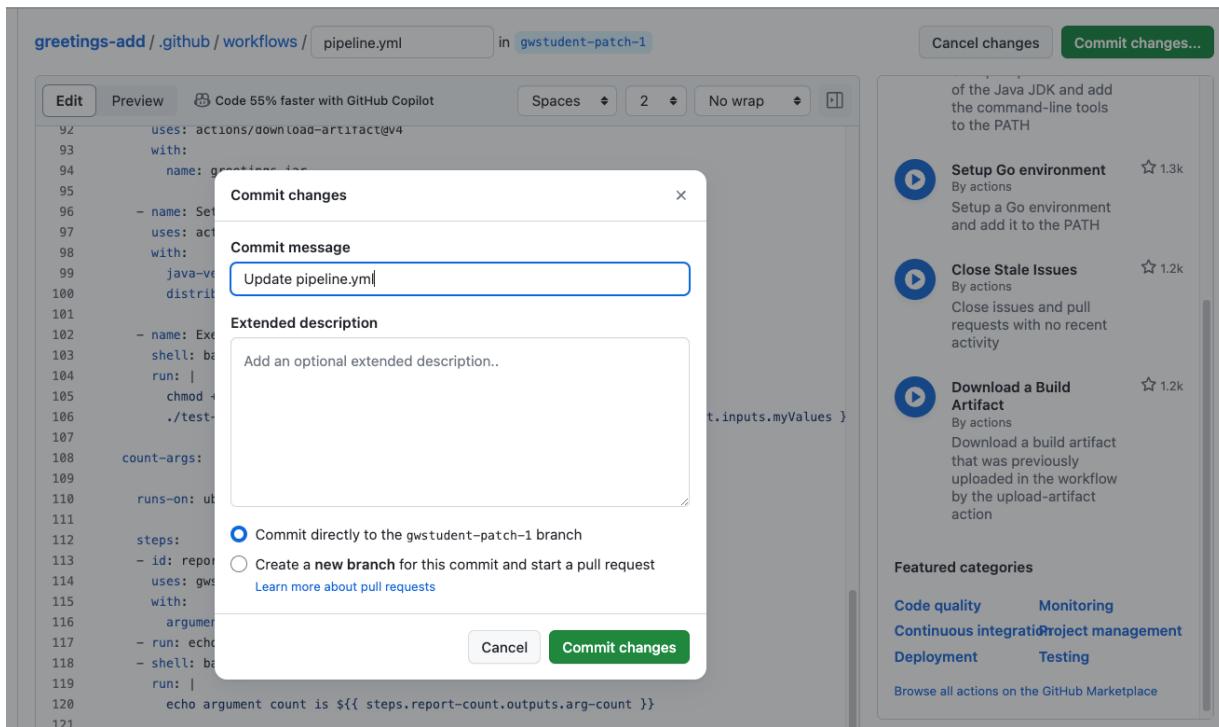
9. Edit the pipeline.yaml file (use the pencil icon). Then update the line that has "uses : <your github userid>/arg-count-action@main" to actually have your GitHub userid in it.

```

90
91   count-args:
92
93     runs-on: ubuntu-latest
94
95     steps:
96       - id: report-count
97         uses: gwstudent/arg-count-action@main
98         with:
99           arguments-to-count: ${{ github.event.inputs.myValues }}
100      - run: echo
101      - shell: bash
102        run: |
103          echo argument count is ${{ steps.report-count.outputs.arg-count }}
104

```

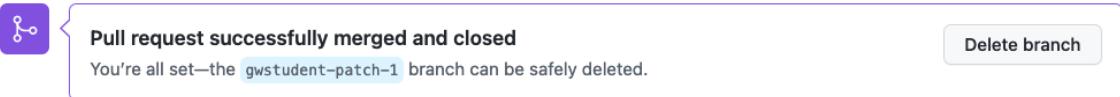
10. When you're done, click on the green "Commit changes..." button, add in a comment if you want, leave the selection set to "Commit directory to the ... branch" so it will go into the same patch branch as before. Then select "Commit changes".



11. Now click on the "Pull requests" link at the top of the page and select the Pull Request again. Eventually all the checks should complete. You can now choose to "Merge pull request", confirm the merge and delete the branch.

Add more commits by pushing to the **gwstudent-patch-1** branch on **gwstudent/greetings-add**.

The screenshot shows the 'Branch protection rules' section for a pull request. It includes a 'Require approval from specific reviewers before merging' section with a 'Branch protection rules' link and an 'Add rule' button. Below are two green checkmarks: 'All checks have passed' (4 successful checks) and 'This branch has no conflicts with the base branch' (Merging can be performed automatically). At the bottom is a large green 'Merge pull request' button with a dropdown arrow, and a note: 'You can also open this in GitHub Desktop or view command line instructions.'



12. Afterwards, you should see that a new run of the workflows in main has been kicked off and will eventually complete.

Workflow Run Details			
Event	Status	Branch	Actor
10 minutes ago	Success	main	gwstudent
36s			

## Lab 7: Exploring logs

**Purpose:** In this lab, we'll take a closer look at the different options for getting information from logs.

1. If not already there, switch back to the Actions tab. To the right of the list of workflows is a search box. Let's execute a simple search - note that only certain keywords are provided and not a complete search. Let's search for the workflow runs that were done for the branch that you used for the Pull Request in the last lab. Enter "**branch:<patch-branch-name>**" (no spaces) in the search box and hit enter.

Workflow Run Details			
Event	Status	Branch	Actor
15 minutes ago	Success	gwstudent-patch-1	gwstudent
39s			
32 minutes ago	Success	gwstudent-patch-1	gwstudent
40s			

- Click on the "X" at the right end of the search box to clear the entry. You can also accomplish the same thing by clicking on the items in the "workflow run results" bar. Clicking on one of the arrows next to them will bring up a list of values to select from that will also filter the list. Try clicking on some of them. Click on the "X" again when done.

All workflows  
Showing runs from all workflows

Filter workflow runs

35 workflow run results

		Event	Status	Branch	Actor
<input checked="" type="checkbox"/>	Merge pull request #1 from gwstudent/gwstudent-patch-1	main	Success	gwstudent-patch-1	... 36s ago
<input checked="" type="checkbox"/>	Update pipeline.yml	gwstudent-patch-1	Success	main	... 36s ago

Filter by branch

Find a branch

gwstudent-patch-1

main

- Make sure you are on the branch *main*. (Switch back if you need to.) Select the "Simple Pipe" workflow. You'll now have a box with "..." beside the search box that has some additional options. Click on the "..." beside the search box to see some of them. They include disabling the workflow and setting up a "badge" for the workflow. Let's go ahead and set up a badge now to show success/failure for running the workflow. Click on the entry for "Create status badge".

Code Pull requests Actions Projects Wiki Security Insights Settings

Actions New workflow

All workflows

Simple pipe

Management Caches

Simple pipe

pipeline.yml

35 workflow runs

Event Status Branch Actor

Create status badge

Disable workflow

Run workflow

This workflow has a `workflow_dispatch` event trigger.

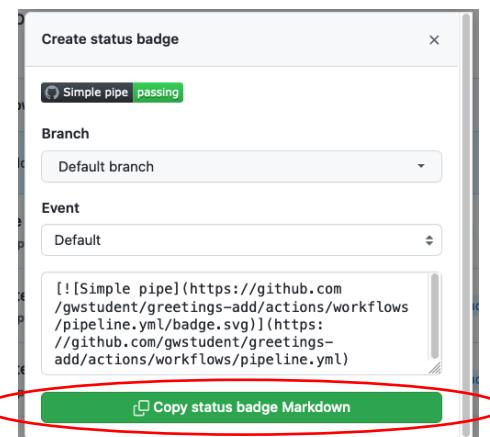
Merge pull request #1 from gwstudent/gwstudent-patch-1

Simple pipe #35: Commit 04ab81b pushed by gwstudent

main

17 minutes ago 36s ...

- In the dialog that pops up, click on the entry for "Copy status badge Markdown". Then close the dialog.



- Click on the "<> Code" tab at the top of the project. At the bottom of the file list, click on the green button to "Add a README" (or edit the README if you already have one). Paste the code you copied in the previous step into the README.md text edit window.

```

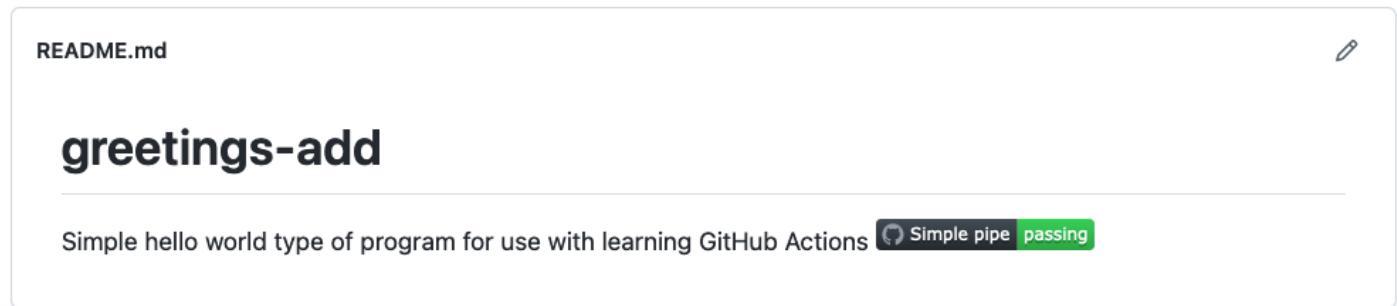
greetings-add / README.md in main
Cancel changes

<> Edit new file Preview Spaces 2 No wrap

1 # greetings-add
2 Simple hello world type of program for use with learning GitHub Actions
3 [![Simple pipe](https://github.com/gwstudent/greetings-add/actions/workflows/pipeline.yml/badge.svg)](https://github.com/gwstudent/greetings-add/actions/workflows/pipeline.yml)

```

- Scroll down and commit your changes. Then you should see the badge showing up as part of the README.md content.



- Click back on the Actions tab. Click on the name of the top run in the Workflow runs list. Notice that we have information in the large bar at the top about who initiated the change, the SHA1, the status, duration, and number of artifacts.

Actions

All workflows

36 workflow runs

	Event	Status	Branch	Actor
Create README.md	main	passed	main	3 minutes ago 40s
Merge pull request #1 from gwstudent/gwstudent-patch-1	main	passed	main	29 minutes ago

- In the main part of the window, we have the job graph, showing the status and relationships between jobs. **Click on the "test-run" job**. In the screen that pops up, we can get more information about what occurred on the runner for that job.

First, let's turn on timestamps. Click on the "gear" icon and select the "Show timestamps" entry.

In the list of steps click on the fourth item "Execute test" to expand it. Then, in line 1 of that part, click on the arrowhead after the timestamp to expand the list and see all the steps executed in between.

The screenshot shows the GitHub Actions pipeline interface. On the left, there's a sidebar with a 'Summary' section and a list of jobs: 'build', 'count-args', 'print-build-output', and 'test-run'. The 'test-run' job is selected. The main area shows the 'test-run' job details, which succeeded 1 minute ago in 6s. It includes a log viewer with several log entries. One entry, 'Execute test', is expanded, showing its internal steps. A red circle highlights the gear icon in the top right corner of the log viewer, and another red circle highlights the arrowhead in the expanded 'Execute test' log entry.

9. We can get links to share to any line. Hover over any of the line numbers and then right-click to Copy Link, Open in a New Tab, or whatever you would like to do.

10. Click on the gear icon again. Notice there is an option to "Download log archive" if we want to get a copy of the logs locally. Or we can get a full view of the raw logs by clicking on the last entry.

Click on "View raw logs". When you are done looking at them, switch back to the workflow screen.

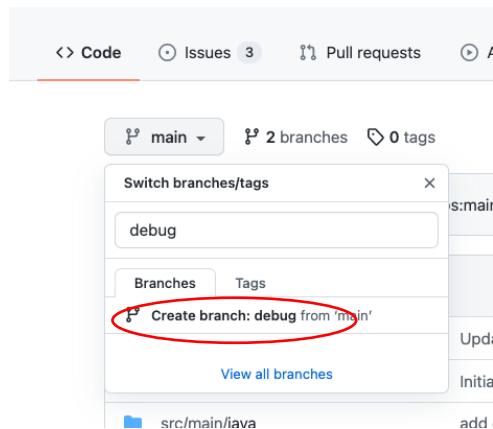
The screenshot shows a browser window displaying the raw log output for a GitHub Actions pipeline. The URL is https://pipelines.actions.githubusercontent.com/serviceHosts/1e0ea379-fff2-4162-91e7-7225d42edb94/\_apis/pipelines/1/run. The log output is a long list of timestamped messages. Notable entries include requests for labels, a job definition, waiting for a runner, starting on a hosted agent, and various runner configurations. The log ends with preparation for actions. A red box highlights the first few lines of the log.

```
2023-01-31T02:14:14.4216423Z Requested labels: ubuntu-latest
2023-01-31T02:14:14.4216457Z Job defined at: gwstudent/greetings-add/.github/workflows/pipeline.yml@refs/heads/main
2023-01-31T02:14:14.4216482Z Waiting for a runner to pick up this job...
2023-01-31T02:14:14.5892733Z Job is waiting for a hosted runner to come online.
2023-01-31T02:14:17.8971699Z Job is about to start running on the hosted runner: Hosted Agent (hosted)
2023-01-31T02:14:22.3710156Z Current runner version: '2.301.1'
2023-01-31T02:14:22.3738482Z ##[group]Operating System
2023-01-31T02:14:22.3739041Z Ubuntu
2023-01-31T02:14:22.3739303Z 22.04.1
2023-01-31T02:14:22.3739591Z LTS
2023-01-31T02:14:22.3739981Z ##[endgroup]
2023-01-31T02:14:22.3740262Z ##[group]Runner Image
2023-01-31T02:14:22.3740633Z Image: ubuntu-22.04
2023-01-31T02:14:22.3740980Z Version: 20230122.1
2023-01-31T02:14:22.3741446Z Included Software: https://github.com/actions/runner-images/blob/ubuntu22/20230122.1/images/linux/Ubuntu2204-Readme.md
2023-01-31T02:14:22.3742125Z Image Release: https://github.com/actions/runner-images/releases/tag/ubuntu22%2F20230122.1
2023-01-31T02:14:22.3742575Z ##[endgroup]
2023-01-31T02:14:22.3742887Z ##[group]Runner Image Provisioner
2023-01-31T02:14:22.3743227Z 2.0.98.1
2023-01-31T02:14:22.3743523Z ##[endgroup]
2023-01-31T02:14:22.3744159Z ##[group]GITHUB_TOKEN Permissions
2023-01-31T02:14:22.3744732Z Contents: read
2023-01-31T02:14:22.3745058Z Metadata: read
2023-01-31T02:14:22.3745627Z ##[endgroup]
2023-01-31T02:14:22.3749547Z Secret source: Actions
2023-01-31T02:14:22.3750061Z Prepare workflow directory
2023-01-31T02:14:22.4575259Z Prepare all required actions
```

## Lab 8: Looking at debug info

Purpose: In this lab, we'll look at some ways to get more debugging info from our workflows.

- First, let's create a new branch in GitHub for the debug instances of our workflows. On the repository's Code page, click on the drop-down under "main", and enter "debug" in the "Find or create a branch..." field. Then click on the "Create branch: debug from 'main'" link in the dialog.



- At this point you should be in the new branch - the "debug" branch. Go to the workflow file in .github/workflows and edit the pipeline.yaml file. **Change the references from "main" in the "on" section at the top to "debug"**. Also, add a new job to surface some debug context. Add in the lines below after the "jobs:" line. Pay attention to indenting again. A screenshot of how everything should look and lines up is further down. (For convenience, the text for the info job is also in a file in extra/info.txt.)

```
info:  
  runs-on: ubuntu-latest  
  
steps:  
  - name: Print warning message  
    run:  
      echo "::warning::This version is for debugging only."  
  - name: Dump context for runner  
    env:  
      RUNNER_CONTEXT: ${{ toJSON(runner) }}  
    run:  
      echo "::debug::Runner context is above."
```

Edit Preview Code 55% faster with GitHub Copilot

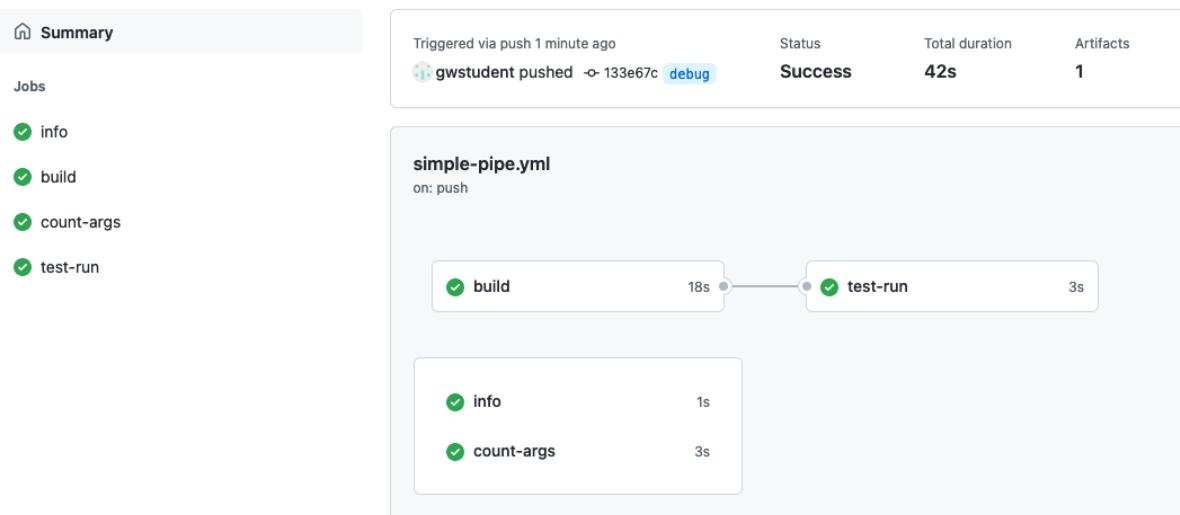
```

11   push:
12     branches: [ "debug" ]
13   pull_request:
14     branches: [ "debug" ]
15   workflow_dispatch:
16
17   inputs:
18     myVersion:
19       description: 'Input Version'
20     myValues:
21       description: 'Input Values'
22
23   jobs:
24     info:
25       runs-on: ubuntu-latest
26
27     steps:
28       - name: Print warning message
29         run: |
30           echo "::warning::This version is for debugging only."
31       - name: Dump context for runner
32         env:
33           RUNNER_CONTEXT: ${{ toJSON(runner) }}
34         run:
35           echo "::debug::Runner context is above."
36
37     build:
38       runs-on: ubuntu-latest

```

- When you are done making the changes, commit directly to the *debug* branch. Switch back to the Actions tab and click on the currently running workflow. Then click on the "info" job in the graph and look at the logs.

#### Update simple-pipe.yml Simple Pipe #11



- Expand the entries for "Print warning message" and "Dump context for runner" to see the outputs for those.

The screenshot shows the GitHub Actions logs for a job named "info". The job status is "Succeeded" and it took 1 second. The logs show the following steps:

- Set up job**: 0s
- Print warning message**: 1s
 

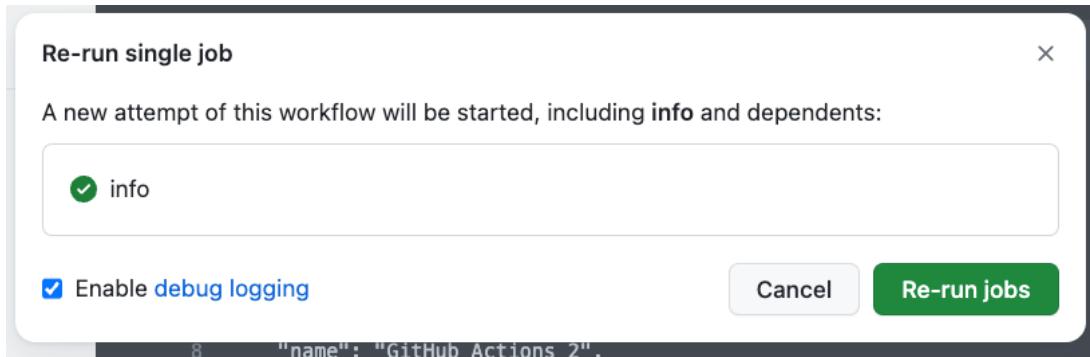
```
1 ► Run echo "::warning::This version is for debugging only."
2 Warning: This version is for debugging only.
```
- Dump context for runner**: 0s
 

```
1 ► Run echo "::debug::Runner context is above."
2 echo "::debug::Runner context is above."
3 shell: /usr/bin/bash -e {0}
4 env:
5   RUNNER_CONTEXT: {
6     "os": "Linux",
7     "tool_cache": "/opt/hostedtoolcache",
8     "temp": "/home/runner/work/_temp",
9     "workspace": "/home/runner/work/greetings-actions"
10 }
```
- Complete job**: 0s

- Notice that while we can see both commands that echo our custom "warning" and "debug" messages - only the output of the warning message actually is displayed, not the output of the debug message. There are a couple of ways to get the debugging info.
- The first way is to simply rerun the job. If you hover over the job name under the Summary section, you can see two curved arrows appear. Click on those. (The same arrows are also available in the upper right of the logs window - next to the gear icon.)

The screenshot shows the GitHub Actions summary page. The "info" job is selected, indicated by a blue border around its row. To the right of the job list is a "Search logs" bar and a set of three icons: a refresh arrow, a gear, and a magnifying glass.

- This will bring up a dialog to re-run that job (and any dependent jobs) - with a checkbox to click to *Enable debug logging*. Click that box and then click the "Re-run jobs" button.



- After the job is re-run, if you look at the latest output, and expand the "Dump context for runner" step, you'll see the actual debug output.

The screenshot shows a GitHub Actions run log for a workflow named "greetings-actions". The left sidebar lists the workflow's summary and various jobs: "info" (selected), "build", "count-args", and "test-run". The main area displays the "info" job's logs. The first log entry is a success message: "succeeded 3 minutes ago in 1s". Below it, the "Print warning message" step is shown as completed with a green checkmark and a duration of "0s". The second log entry is the "Dump context for runner" step, also completed with a green checkmark and "0s" duration. This step outputs a multi-line debug dump of the runner context, starting with "##[debug]Evaluating: toJSON(runner)" and listing various properties like "debug", "os", "arch", "name", and "tool\_cache".

```
1 ##[debug]Evaluating: toJSON(runner)
2 ##[debug]Evaluating toJSON:
3 ##[debug]..Evaluating runner:
4 ##[debug]..=> Object
5 ##[debug]=> '{'
6 ##[debug]  "debug": "1",
7 ##[debug]  "os": "Linux",
8 ##[debug]  "arch": "X64",
9 ##[debug]  "name": "GitHub Actions 2",
10 ##[debug]  "tool_cache": "/opt/hostedtoolcache",
```

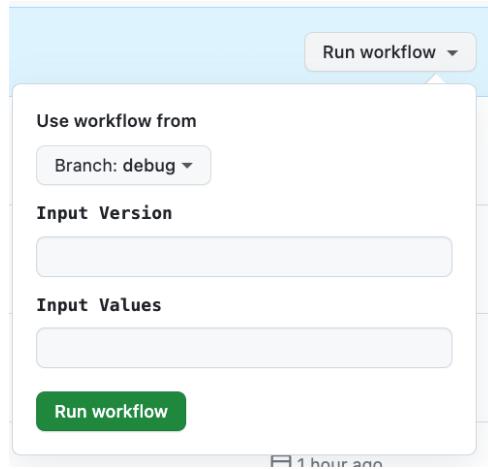
9. However, that doesn't cause debug info to show up for commits. We do that by enabling a secret or a variable for `ACTIONS_STEP_DEBUG`. Since this setting is not sensitive information, we'll use a variable.
  10. To do this, go to the repository's top menu and select "Settings". Then on the left-hand side, under "Security", select "Secrets and variables", and then "Actions" under that. Select the "Variables" tab and "New repository variable".

The screenshot shows the GitHub repository settings page for a specific repository. The top navigation bar includes links for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The Settings link is highlighted with a red circle labeled '1'. On the left, a sidebar lists General, Access, Collaborators, Moderation options, Code and automation (Branches, Tags, Actions, Webhooks, Environments, Codespaces, Pages), and Security (Code security and analysis, Deploy keys, Secrets and variables). The 'Secrets and variables' item is circled with a red line and labeled '2'. The main content area is titled 'Actions secrets and variables'. It explains that secrets and variables allow managing reusable configuration data. It distinguishes between encrypted secrets for sensitive data and plain text variables for non-sensitive data. A note states that anyone with collaborator access can use these for actions. Below this, there are two tabs: 'Secrets' (disabled) and 'Variables' (highlighted with a red circle labeled '3'). A green button labeled 'New repository variable' is circled with a red line and labeled '4'. Below the tabs, sections for 'Environment variables' and 'Repository variables' show that there are no variables defined.

11. On the next screen, enter **ACTIONS\_STEP\_DEBUG** for the name, set the value to *true* and click on the **Add variable** button.

The screenshot shows the 'Actions variables / New variable' creation form. The sidebar on the left is identical to the previous screenshot, with the 'Secrets and variables' item circled and labeled '2'. The main form has a title 'Actions variables / New variable'. A note states that variable values are exposed as plain text. The 'Name \*' field contains 'ACTIONS\_STEP\_DEBUG'. Below it, a list of validation rules is shown: Alphanumeric characters ([a-z], [A-Z], [0-9]) or underscores (\_) only, Spaces are not allowed, Cannot start with a number, and Cannot start with GITHUB\_ prefix. The 'Value \*' field contains 'true'. At the bottom right, a green button labeled 'Add variable' is circled with a red line and labeled '4'.

12. Now, switch back to the "Actions" tab, select the "Simple Pipe" workflow, and click on the "Run workflow" button. **Select "debug" from the list for the branch.** Enter in any desired arguments. Then click the green "Run workflow" button to execute the workflow.



13. A new run will be started. Go into it and select the "info" job. In the output now, if you expand the sections, you should be able to see a lot of "##[debug]" messages including the one you added in the "Dump context for runner" section.

A screenshot of the GitHub Actions job log for the 'info' job. On the left, there's a sidebar with a 'Summary' icon and a 'Jobs' list containing 'info', 'build', 'count-args', and 'test-run', where 'info' is currently selected. The main area shows the 'info' job details: 'succeeded 32 seconds ago in 1s'. Below this, the logs are displayed. An expanded section titled 'Dump context for runner' shows a series of numbered log entries. Entry 39 is circled in red and contains the text '##[debug]Runner context is above.', which corresponds to the step added in the previous step.

14. Note that the debug log info will be turned on for all runs from here on - as long as the repository variable exists and is set to "true".

## Lab 9 – Securing inputs

Purpose: In this lab, we'll look at how to plug a potential security hole with our inputs.

1. Make sure you're in the "main" branch. Switch to the pipeline.yml file in the .github/workflows directory and look at the "test-run" job and in particular, this line in the "Execute test" step:

```
./test-script.sh ${{ needs.build.outputs.artifact-path }} ${{ github.event.inputs.myValues }}}
```

```
84
85     - name: Execute test
86       shell: bash
87       run: |
88         chmod +x ./test-script.sh
89         ./test-script.sh ${{ needs.build.outputs.artifact-path }} ${{ github.event.inputs.myValues }}
90
91   count-args:
--
```

2. When we create our pipelines that execute code based on generic inputs, we must be cognizant of potential security vulnerabilities such as injection attacks. This code is subject to such an attack. To demonstrate this, use the `workflow_dispatch` event for the workflow in the *Actions* menu, put in a version and pass in the following as the arguments in the arguments field (NOTE: That is two backquotes around `ls -la`) ``ls -la`` Then hit "Run workflow".

The screenshot shows the GitHub Actions interface. On the left, the 'Actions' sidebar is visible with options like 'All workflows', 'create-failure-issue', and 'Simple pipe' (which is selected). The main area displays the 'Simple pipe' workflow with four runs listed. The fourth run, which is the most recent, has a modal dialog open over it. The modal contains fields for 'Use workflow from' (set to 'Branch: main'), 'Input Version' (set to '1.0.2'), and 'Input Values' (containing the value 'ls -la'). Below these fields is a green 'Run workflow' button. The run details in the modal show the commit information: 'Simple pipe #37: Commit 4c29dcb pushed by gwstudent' and the status 'debug'. The run was completed '1 hour ago' with a duration of '41s'.

3. After the run completes, look at the output of the "test-run" job. Select the "Execute test" step and expand the logs. Notice that the step itself ran successfully, but it has actually run the `'ls -la'` command directly on the runner system. (Scroll down past the initial debug info to see it - around line 60.) The command was innocuous in this case, but this could have been a more destructive command.

The screenshot shows the GitHub Actions interface. On the left, there's a sidebar with 'Summary', 'Jobs' (listing 'build', 'count-args', 'print-build-output', 'test-run', and 'create-issue-on-failure'), 'Run details' (with 'Usage' and 'Workflow file' links), and a 'test-run' section which is expanded. The main area shows the log for the 'test-run' job, specifically the 'Execute test' step. The log output is as follows:

```

test-run
succeeded 15 minutes ago in 3s
Search logs
0s

Execute test
48 ##[debug]./test-script.sh build/libs/greetings-add-1.0.2-2023-01-31T04-50-08.jar `ls -la`
49 ##[debug]!
50 ##[debug]Loading env
51 ▼ Run chmod +x ./test-script.sh
52 chmod +x ./test-script.sh
53 ./test-script.sh build/libs/greetings-add-1.0.2-2023-01-31T04-50-08.jar `ls -la`
54 shell: /usr/bin/bash --noprofile --norc -e -o pipefail {0}
55 ##[debug]/usr/bin/bash --noprofile --norc -e -o pipefail /home/runner/work/_temp/e5605cf6-7472-4743-9336-6c8e59b25863.sh
56 total
57 16
58 drwxr-xr-x
59 drwxr-xr-x
60 drwxr-xr-x
61 3
62 3
63 3
64 runner
65 runner
66 runner
67 runner
68 docker
69 docker

```

- Let's fix the command to not be able to execute the code in this way. We can do that by placing the output into an environment variable first and then passing that to the step. Edit the `pipeline.yaml` file and change the code for the "Execute test" step to look like the following. This code replaces the code after the "shell: bash" line - pay attention to how things line up:

```

env:
  ARGS: ${{ github.event.inputs.myValues }}
run: |
  chmod +x ./test-script.sh
  ./test-script.sh ${needs.build.outputs.artifact-path} "$ARGS"

```

The screenshot shows the GitHub Actions pipeline editor. At the top, it says 'greetings-add / .github / workflows / pipeline.yml' and 'in main'. Below is the YAML configuration:

```

  ...
  68   print-build-output:
  69     runs-on: ubuntu-latest
  70     needs: build
  71     steps:
  72       - run: echo ${needs.build.outputs.artifact-path}
  73
  74
  75   test-run:
  76     runs-on: ubuntu-latest
  77     needs: build
  78
  79     steps:
  80       - name: Download candidate artifacts
  81         uses: actions/download-artifact@v3
  82         with:
  83           name: greetings-jar
  84
  85       - name: Execute test
  86         shell: bash
  87         env:
  88           ARGS: ${{ github.event.inputs.myValues }}
  89         run: |
  90           chmod +x ./test-script.sh
  91           ./test-script.sh ${needs.build.outputs.artifact-path} "$ARGS"
  92
  93
  94   count-args:
  95
  96     runs-on: ubuntu-latest

```

5. Commit back the changes and wait till the action run for the push completes.
6. Now, you can execute the code again with the same arguments as before.

The screenshot shows the GitHub Actions interface for the 'Simple Pipe' workflow. The sidebar on the left has 'Actions' selected, with 'Simple Pipe' highlighted. The main area displays 16 workflow runs. One specific run is shown in detail: 'Update pipeline.yml' (main branch), which was committed by gwstudent2. The run echo'd the command 'ls -la'. On the right, there's a sidebar with options to 'Use workflow from Branch: main', set 'Input Version 1.0.2-', and enter 'Input Values 'ls -la''. A green 'Run workflow' button is also present.

7. Notice that this time, the output did not run the commands, but just echoed them back out as desired.

END OF LAB

## Lab 10: (Bonus/Optional) Chaining workflows, using conditionals, and working with REST APIs in workflows.

**Purpose:** Learning one way to drive one workflow from another.

1. 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. But first, we need to ensure that the "Issues" functionality is turned on for this repository. Go to the project's Settings main page, scroll down and under "Features", make sure the "Issues" selection is checked.

The screenshot shows the GitHub repository settings for 'greetings-actions'. In the 'Features' section, the 'issues' checkbox is checked and circled in red. A tooltip below it states: 'Issues integrate lightweight task tracking into your repository. Keep projects on track with issue labels and milestones, and reference them in commit messages.' At the bottom of the features section, there's a 'Get organized with issue templates' section with a 'Set up templates' button.

2. The workflow to create the issue using a REST API call is already written to save time. It is in the main project under "extra/create-failure-issue.yml". You need to get this file in the .github/workflows directory. You can just move it via GitHub with the following steps.

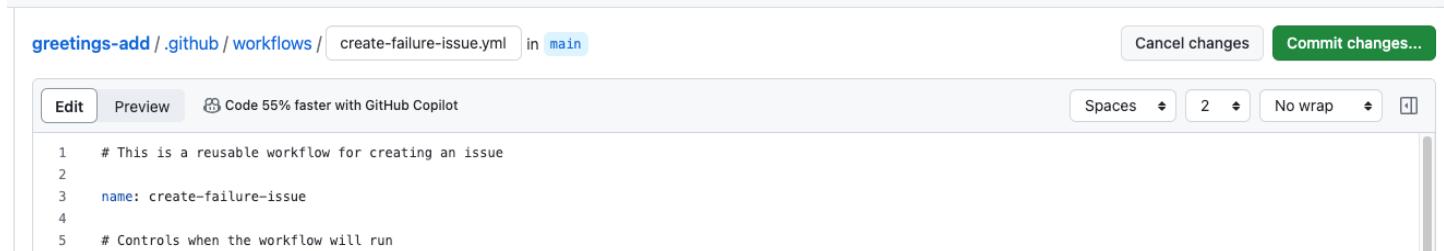
- a. In the repository, browse to the "extra" folder and to the "create-failure-issue.yml" file.
- b. Take a few moments to look over the file and see what it does. Notice that:
  - i. It has a workflow\_call section in the "on" area, which means it can be run from another workflow.
  - ii. It has a workflow\_dispatch section in the "on" area, which means it can be run manually.
  - iii. It has two inputs - a title and body for the issue.
  - iv. The primary part of the body is simply a REST call (using the GITHUB\_TOKEN) to create a new issue.
- c. Click the pencil icon to edit it.



A screenshot of a GitHub repository page. The path 'greetings-add / extra / create-failure-issue.yml' is shown at the top. Below it, a commit by 'brentlaster' titled 'Update create-failure-issue.yml' is listed, made 8713637 · 10 months ago. A 'History' link is next to the commit. Below the commit, there's a 'Code' button, a 'Blame' link, and a note 'Code 55% faster with GitHub Copilot'. On the right, there are 'Raw', 'Edit this file' (which is highlighted), 'Download', and other file operations buttons. The code editor shows the following content:

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

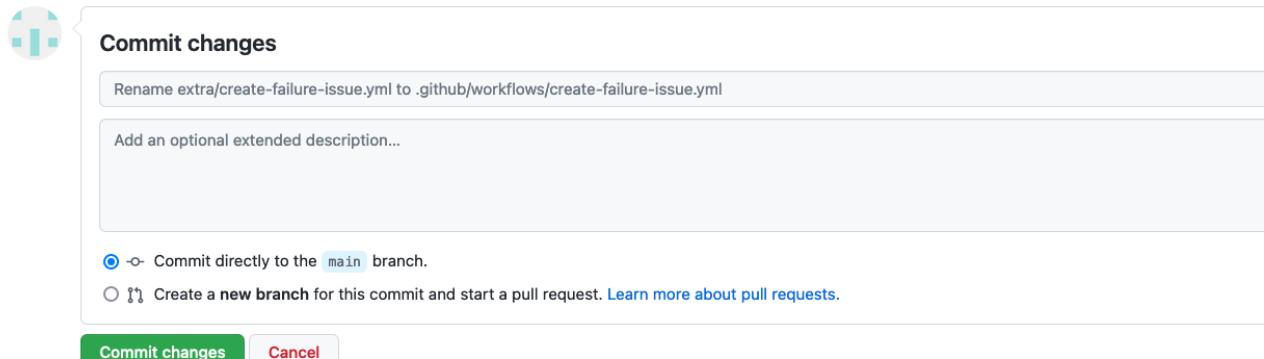
- d. In the filename field at the top, change the name of file. Use the backspace key to backspace over "extra/" making sure to backspace over the word. Then type in the path to put it in the workflows ".github/workflows/create-failure-issue.yml".



A screenshot of a GitHub repository page. The path 'greetings-add / .github / workflows / create-failure-issue.yml' is shown at the top. Below it, a 'main' branch indicator is shown. On the right, there are 'Cancel changes' and a green 'Commit changes...' button. The code editor shows the same content as the previous screenshot:

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

- e. To complete the change, commit as usual using the "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".

The screenshot shows the GitHub Workflows interface. On the left, there's a sidebar with 'Workflows' and 'All workflows'. Under 'All workflows', 'Simple Pipe' is listed, and 'create-failure-issue' is highlighted with a blue background. The main area shows the 'create-failure-issue' workflow with its YAML file, 'create-failure-issue.yml'. It indicates '1 workflow run'. Below that, it says 'This workflow has a workflow\_dispatch event trigger.' A 'Run workflow' button is visible. A modal window is overlaid on the page, prompting for 'Issue title' and 'Issue body' fields, both of which are currently set to placeholder text.

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

The screenshot shows the GitHub Issues tab. At the top, there are navigation links: Code, Issues (with a count of 1), Pull requests, Actions, Projects, Wiki, Security, Insights, Settings. Below that is a search bar with 'is:issue is:open' and buttons for Labels (9) and Milestones (0). A 'New issue' button is also present. The main list shows one issue: '#2 opened now by github-actions bot'. The issue title is 'FAILURE: This is a title' and it is marked as 'Open'.

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

**create-issue-on-failure:**

```
permissions:
  issues: write
needs: [test-run, count-args]
if: always() && failure()
uses: ./github/workflows/create-failure.yml
```

```

with:
  title: "Automated workflow failure issue for commit ${{ github.sha }}"
  body: "This issue was automatically created by the GitHub Action workflow ** ${{ github.workflow }} **"

```

- In order to have this executed via the "if" statement, we need to force a failure. We can do that by simply adding an "exit 1" line at the end of the "count-args" job (right above the job you just added).

Make that change too. (A screenshot is below showing what the changes should look like. The "exit 1" is line 65 in the figure.)



The screenshot shows a GitHub code editor interface. The file being edited is `simple-pipe.yml` in the `gwstudent2:main` branch. The code is as follows:

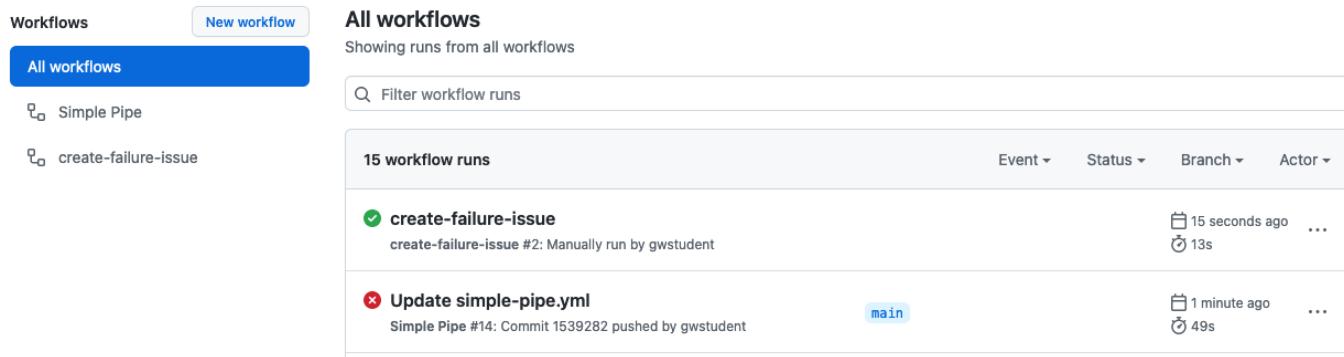
```

71   - shell: bash
72     run: |
73       echo argument count is ${{ steps.report-count.outputs.arg-count }}
74       exit 1
75
76   create-issue-on-failure:
77
78     permissions:
79       issues: write
80     needs: [test-run, count-args]
81     if: always() && failure()
82     uses: ./github/workflows/create-failure-issue.yml
83     with:
84       title: "Automated workflow failure issue for commit ${{ github.sha }}"
85       body: "This issue was automatically created by the GitHub Action workflow ** ${{ github.workflow }} **"
86

```

The line `exit 1` is highlighted in blue, indicating it is selected or being edited.

- After you've made the changes, commit them. At that point, you should get a run of the workflow. Click back to the Actions tab to watch it. After a few minutes, it will complete, and the "count-args" job will fail. This is expected because of the "exit 1" we added. But in a few moments, the create-issue-on-failure job should kick in and invoke the other workflow and produce a new ticket.



The screenshot shows the GitHub Actions tab with the following details:

- Workflows:** All workflows
- All workflows:** Showing runs from all workflows
- Filter:** Filter workflow runs
- 15 workflow runs:**
  - create-failure-issue:** Status: Success, Run #2, Last run 15 seconds ago, 13s duration. Description: create-failure-issue #2: Manually run by gwstudent.
  - Update simple-pipe.yml:** Status: Failed, Run #14, Last run 1 minute ago, 49s duration. Description: Simple Pipe #14: Commit 1539282 pushed by gwstudent.

8. You can look at the graphs from the runs of the two workflows if you want.

✖ **Update simple-pipe.yml** Simple Pipe #14

Triggered via push 3 minutes ago  
gwstudent pushed → 1539282 main

Status: **Failure** Total duration: **49s** Artifacts: **1**

Jobs:

- ✓ build
- ✗ count-args
- ✓ test-run
- ✓ create-issue-on-failure

**simple-pipe.yml**  
on: push

```
graph LR; A[count-args] -- "3s" --> B[test-run]; B -- "2s" --> C[create-issue-on-failure]; C -- "1s" --> D[build]; D -- "16s" --> B;
```

Annotations: 1 error

✗ **count-args**  
Process completed with exit code 1.

Artifacts: Produced during runtime

Name	Size
📦 greetings-jar	1006 Bytes

✓ **create-failure-issue** create-failure-issue #2

Manually triggered 3 minutes ago  
gwstudent → 1539282

Status: **Success** Total duration: **13s** Artifacts: **-**

Jobs:

- ✓ create\_issue\_on\_failure

**create-failure-issue.yml**  
on: workflow\_dispatch

```
graph LR; A[create_issue_on_failure] -- "1s" --> B;
```

9. Under "Issues", you can also see the new ticket that was opened with the text sent to it.

Code Issues 2 Pull requests Actions Projects Wiki Security Insights Settings

Automated workflow failure issue for commit  
153928267f2fc38a4e91b5bd00d61f033abd59d6 #4

Open github-actions bot opened this issue 3 minutes ago · 0 comments

github-actions bot commented 3 minutes ago

This issue was automatically created by the GitHub Action workflow \*\* Simple Pipe \*\*

THE END - THANKS!