

GitHub Actions Deep Dive

Revision 2.2 – 07/13/23

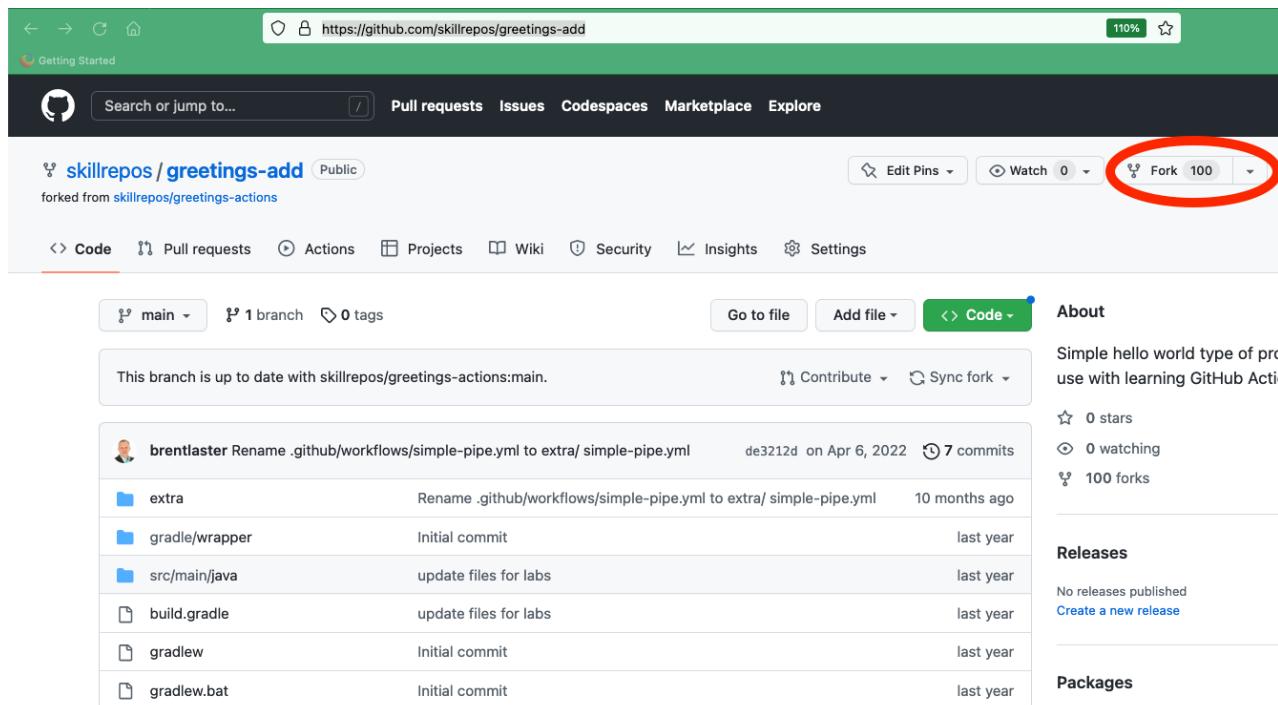
Tech Skills Transformations LLC / Brent Laster

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.



The screenshot shows the GitHub repository page for `skillrepos/greetings-add`. The URL in the address bar is <https://github.com/skillrepos/greetings-add>. The repository is public and was forked from `skillrepos/greetings-actions`. The top navigation bar includes links for Pull requests, Issues, Codespaces, Marketplace, and Explore. On the right side of the header, there is a 'Fork' button with a dropdown menu showing '100' forks. This button is highlighted with a red circle. Below the header, there are tabs for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Code' tab is selected. The main content area shows the repository's history. It lists a single commit by `brentlaster` that renames `.github/workflows/simple-pipe.yml` to `extra/simple-pipe.yml`. The commit was made on April 6, 2022, and has 7 commits. To the right of the commit list, there is an 'About' section with details: Simple hello world type of pro use with learning GitHub Action, 0 stars, 0 watching, and 100 forks. There are also sections for Releases (no releases published) and Packages (Create a new release).

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

This screenshot shows the GitHub repository page for `skillrepos/greetings-add`. The `Actions` tab is highlighted with a red circle. Below the tabs, there are buttons for `main`, `1 branch`, and `0 tags`, along with links to `Go to file`, `Add file`, and `Code`. A message indicates the branch is up to date with the main branch. Below this, a list of commits by `brentlaster` is shown, with the most recent commit being a rename of `.github/workflows/simple-pipe.yml` to `extra/simple-pipe.yml`.

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".

This screenshot shows the "Browse all categories" section of the GitHub Actions page. It lists several categories: Automation, Greetings, Stale, Manual workflow, Labeler, and a "Continuous integration" category which is highlighted with a red circle. Below this, there are links for Automation, Deployment, and Security.

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.

This screenshot shows the "Get started with GitHub Actions" page for the "Continuous integration" category. The search bar at the top contains the word "Gradle", which is highlighted with a red circle. Below the search bar, it says "Found 52 workflows". A list of workflows is displayed, including "Android CI", "Java with Ant", "Clojure", "Publish Java Package", "Java with Gradle", and "Publish Java Package".

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

- This will bring up a page with a starter workflow for CI that we can edit as needed. There are two edits we'll make - changing the name of the file and the name of the workflow. 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.yaml. (You can just backspace over or delete the name and type the new name.)

TO

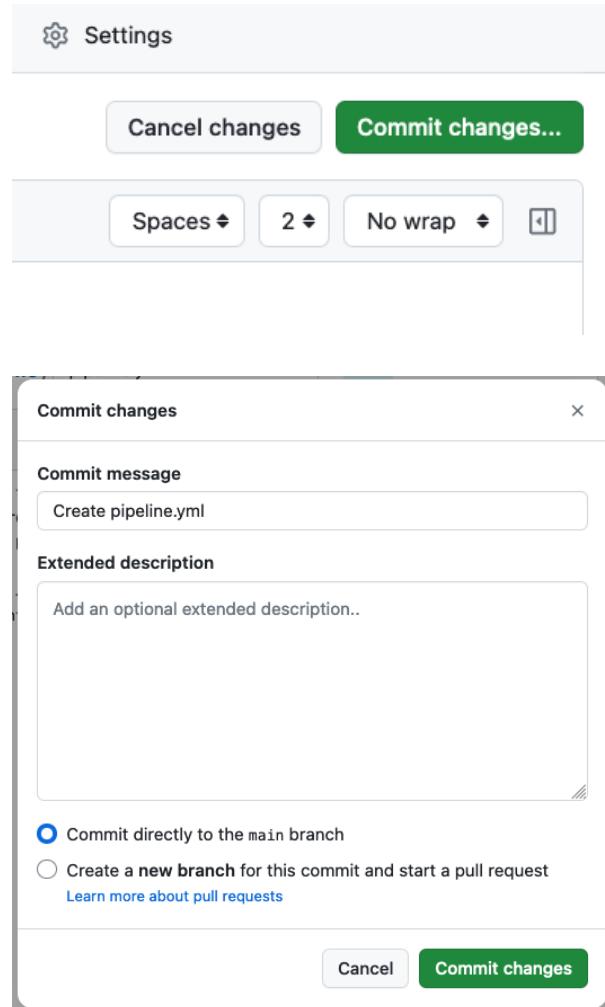
- Now, edit the name of the workflow - change line 8 from "name: Java CI with Gradle" to "name: Simple Pipe".

```

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



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

Actions		New workflow
All workflows	<input checked="" type="radio"/> Create pipeline.yml	1 workflow run
Simple pipe	<input type="radio"/>	Event ▾
Management	<input type="radio"/>	Status ▾
Caches	<input type="radio"/>	Branch ▾
	<input type="radio"/>	Actor ▾

All workflows
Showing runs from all workflows

1 workflow run

Create pipeline.yml
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.

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.

* 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.

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

The screenshot shows the GitHub Actions pipeline.yml file in the editor. The file content is:

```

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     branches: [ "main" ]

```

The top right corner of the editor has a toolbar with several icons. One icon, specifically the edit icon (pencil), is circled in red.

- 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" 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 / Search results

- Veracode Upload And Scan** By veracode (13)
Upload files to veracode and start a static scan
- Cloud Storage Uploader** By google-github-actions (51)
Upload files or folders to GCS buckets
- Upload a Build Artifact** By actions (1.1k)
Upload a build artifact that can be used by subsequent workflow steps
- Run tfsec with sarif upload** By aquasecurity (17)
Run tfsec against terraform code base and upload the sarif output to the github repo
- twine-upload** By yaananth (1)
Upload to twine

Marketplace / Search results / Upload a Build Artifact

Upload a Build Artifact

By actions (1.1k) 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 is version 3.0.0 and is labeled as 'Latest version'. It has a green status badge 'test-local passing'. To the right, there is a 'Use latest version' button. Below the action details, there is a 'Verified creator' badge, a 'Stars' counter (2.5k), a 'Contributors' section with several user icons, and a 'Categories' section with 'Utilities' selected. In the 'Links' section, the 'actions/checkout' link is highlighted with a red oval. Other links include 'Open issues' (209) and '209' releases.

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' repository. At the top, there is a navigation bar with 'Code' (highlighted with a red oval), 'Issues 149', 'Pull requests 35', 'Discussions', 'Actions' (highlighted with a red oval), 'Projects', 'Wiki', 'Security', and 'Insights'. Below the navigation, there is a section for 'Use this GitHub Action with your project' and a table of workflow files. On the right, there is an 'About' section with a description, a 'github.com/features/actions' link, a 'Readme' link, and an 'MIT License' link. There is also a 'Releases 16' section with a 'v2.3.4 Latest' link and a '15 releases' link.

The screenshot shows the GitHub Actions checkout page for a repository. On the left, there's a sidebar with 'Workflows' and a 'All workflows' button. The main area is titled 'All workflows' and shows 'Showing runs from all workflows'. A search bar at the top says 'Filter workflow runs'. Below it, a table lists two workflow runs:

	Event	Status	Branch	Actor
Create check-dist.yml (#566) Build and Test #532: Commit afe4af0 pushed by thboop	main	Success	14 days ago	...
Create check-dist.yml (#566) Licensed #62: Commit afe4af0 pushed by thboop	main	Success	14 days ago	...

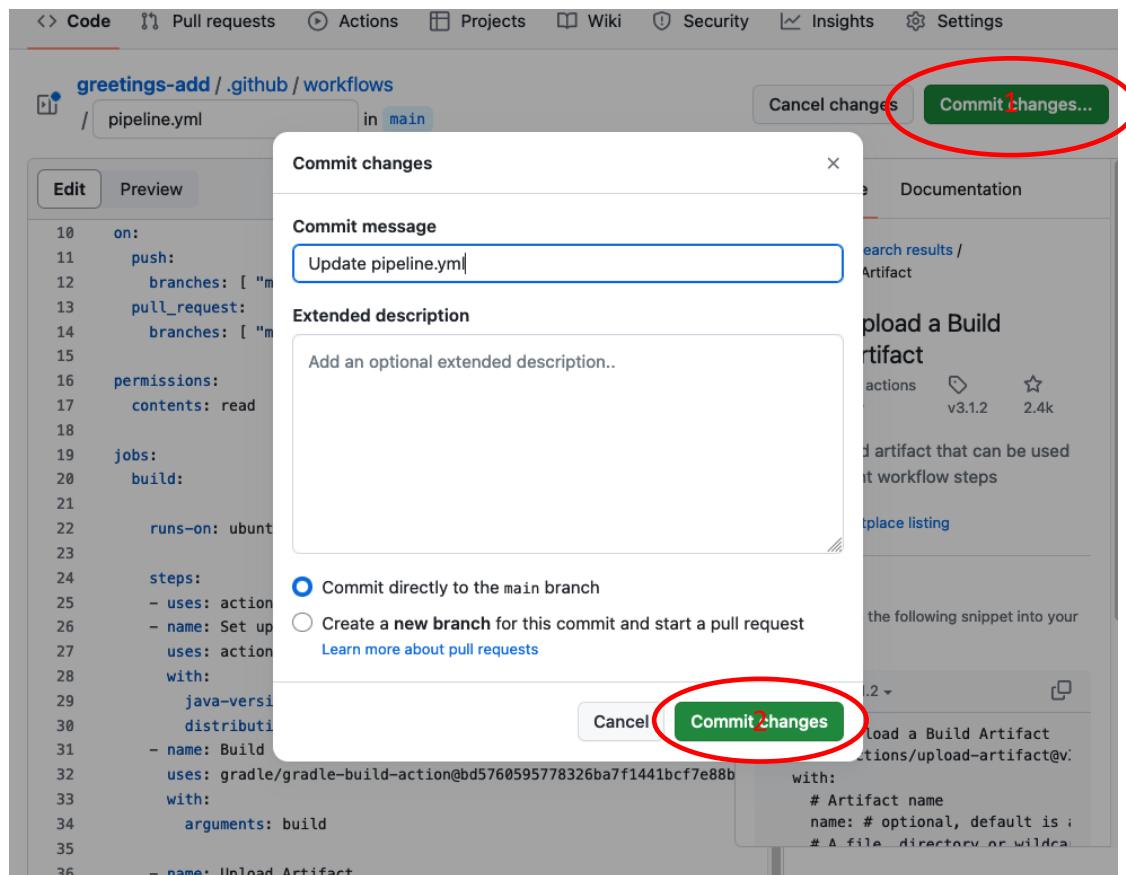
6. Switch back to the browser tab where you are editing the workflow for greetings-actions. Update the build job to include a new step to use the "upload-artifact" action to upload the jar the build job creates. To do this, add the following lines inline with the build job steps. Pay attention to the indenting. If you see red wavy lines under your code, that likely means the indenting is off. See the screenshot (lines 36-40) for how this should look afterwards. (Your line numbers may be different.)

```
- name: Upload Artifact
  uses: actions/upload-artifact@v3
  with:
    name: greetings-jar
    path: build/libs
```

The screenshot shows the GitHub workflow editor with the YAML code for the workflow. Line 36 highlights the 'Upload Artifact' step, which is the one we added. The code is as follows:

```
permissions:
contents: read
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 11
        uses: actions/setup-java@v3
        with:
          java-version: '11'
          distribution: 'temurin'
      - name: Build with Gradle
        uses: gradle/gradle-build-action@67421db6b
        with:
          arguments: build
      - name: Upload Artifact
        uses: actions/upload-artifact@v3
        with:
          name: greetings-jar
          path: build/libs
```

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.

The screenshot shows a pipeline summary for a job named "build". Under the "Artifacts" section, there is a table with one row: "greetings-jar" (Size: 1006 Bytes). The "greetings-jar" entry is circled in red. Below the artifacts, there is a "Gradle Builds" table with one row: "greetings-add" (Requested Tasks: build, Gradle Version: 4.10, Build Outcome: checked, Build Scan™: NOT PUBLISHED). A note below the table says "▶ Caching for gradle-build-action was enabled - expand for details". At the bottom, it says "Job summary generated at run-time".

* 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

- 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 pipeline editor for the "greetings-add" repository. The "pipeline.yaml" file is open. A red oval highlights the "workflow_dispatch" trigger added at the bottom of the "on:" section. The code snippet is as follows:

```
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]
  workflow_dispatch:
    inputs:
      myVersion:
        description: 'Input Version'
      myValues:
        description: 'Input Values'
```

- 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
```

- 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.

```
- 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.

```
40   - name: Build with Gradle
41     uses: gradle/gradle-build-action@67421db6bd0bf253fb4bd25b31ebb98943c375e1
42     with:
43       arguments: build
44
45   - name: Set timestamp
46     run: echo TDS=$(date +'%Y-%m-%dT%H-%M-%S') >> $GITHUB_ENV
47
48   - name: Tag artifact
49     run: mv build/libs/greetings-add.jar build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.
50
51   - name: Upload Artifact
52     uses: actions/upload-artifact@v3
53     with:
54       name: greetings-jar
55       path: build/libs
```

- Go ahead and commit the changes. 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.

```

build
succeeded 7 minutes ago in 16s
Search logs
0s

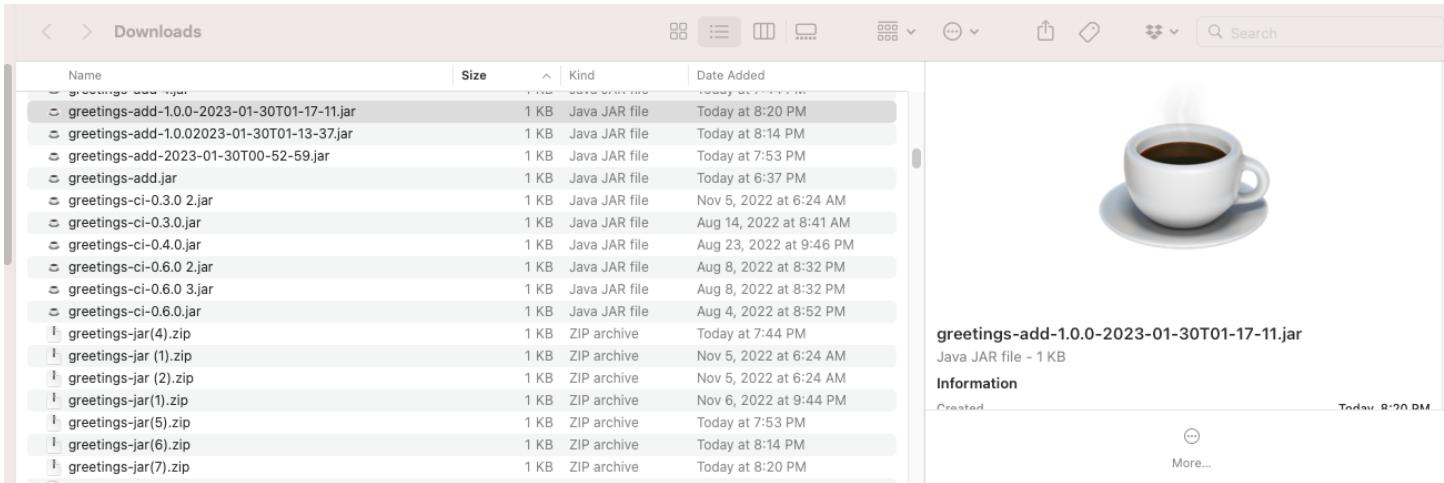
Tag artifact
1 ✓ Run mv build/libs/greetings-add.jar build/libs/greetings-add-2023-01-30T00-57-54.jar
2 mv build/libs/greetings-add.jar build/libs/greetings-add-2023-01-30T00-57-54.jar
3 shell: /usr/bin/bash -e {0}
4 env:
5 JAVA_HOME: /opt/hostedtoolcache/Java_Temurin-Hotspot_jdk/11.0.17-8/x64
6 JAVA_HOME_11_X64: /opt/hostedtoolcache/Java_Temurin-Hotspot_jdk/11.0.17-8/x64
7 GRADLE_BUILD_ACTION_SETUP_COMPLETED: true
8 GRADLE_BUILD_ACTION_CACHE_RESTORED: true
9 TDS: 2023-01-30T00-57-54
10
11
12 With the provided path, there will be 1 file uploaded
13 Starting artifact upload
14 For more detailed logs during the artifact upload process, enable step-debugging: https://docs.github.com/actions/monitoring-and-troubleshooting-workflows/enabling-debug-logging#enabling-step-debug-logging
15 Artifact name is valid!
16 Container for artifact "greetings-jar" successfully created. Starting upload of file(s)
17 Total size of all the files uploaded is 871 bytes

```

- 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".

Run	Status	Timestamp
Simple pipe #20: Manually run by gwstudent	Success	26 minutes ago
Update pipeline.yml #19: Commit 2e3c0c1 pushed by gwstudent	Failure	
Update pipeline.yml #18: Commit ac22d5e pushed by gwstudent	Failure	
Update pipeline.yml #17: Commit 3a2a2a2 pushed by gwstudent	Failure	

- 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.



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

1. 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
```

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

```

45   uses: gradle/gradle-build-action@67421db6bd0bf253fb4bd25b31ebb98943c375e1
46   with:
47     arguments: build
48
49 - name: Set tag
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 }}${{ env.TDS }}.jar
54
55 - name: Upload Artifact
56   uses: actions/upload-artifact@v3
57   with:
58     name: greetings-jar
59     path: build/libs
60
61 - name: Set output
62   id: setoutput
63   run: echo jarpath=build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}.jar >> $GITHUB_OUTPUT
64
65

```

2. 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 }}

26
27   jobs:
28     build:
29
30       runs-on: ubuntu-latest
31
32       # Map a step output to a job output
33       outputs:
34         artifact-path: ${{ steps.setoutput.outputs.jarpath }}
35
36     steps:
37       - uses: actions/checkout@v3
38       - name: Set up JDK 11
39         uses: actions/setup-java@v3
40           ...

```

- In order to verify that we can see the output from the build job, add a new, second job in the workflow file. 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}}

```

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

```

<> Edit file  ⏎ Preview changes  in main
      Spaces 2 No wrap
51
52   - name: Tag artifact
53     run: mv build/libs/greetings-add.jar build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.
54
55   - name: Upload Artifact
56     uses: actions/upload-artifact@v3
57     with:
58       name: greetings-jar
59       path: build/libs
60
61   - name: Set output
62     id: setoutput
63     run: echo jarpath=build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}.jar >> $GITH
64
65
66   print-build-output:
67     runs-on: ubuntu-latest
68     needs: build
69     steps:
70       - run: echo ${{needs.build.outputs.artifact-path}}
71

```

Use Control + Space or Option + Space to trigger autocomplete in most situations.

- 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.

The screenshot shows two views of a GitHub Actions pipeline named "Simple pipe".

Pipeline Summary:

- Re-run triggered 1 minute ago by gwstudent on main branch.
- Status: Success
- Total duration: 37s
- Artifacts: 1
- Jobs: build, print-build-output

Job Log (print-build-output):

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

- 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. Open up that file (in the "extra" subdirectory) and click the pencil icon to edit it.

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

- Give feedback
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).

```
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 second 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.

© 2023 Tech Skills Transformations, LLC & Brent Laster

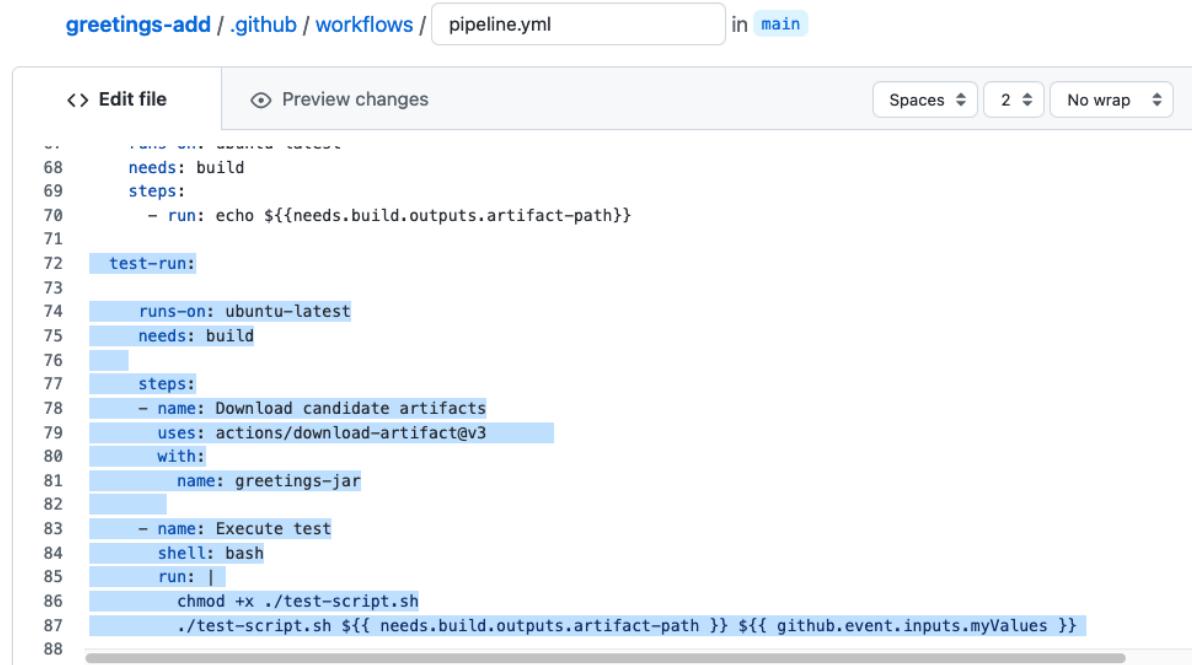
What this code does is wait for the build job to complete (the *needs: build* part), then run two steps. The first step downloads the artifacts we uploaded before to have them there for the testing script. And the second step runs the separate testing script against the downloaded artifacts, making it 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@v3
      with:
        name: greetings-jar

    - name: Execute test
      shell: bash
      run: |
        chmod +x ./test-script.sh
        ./test-script.sh ${needs.build.outputs.artifact-path} ${github.event.inputs.myValues}
```



A screenshot of a GitHub workflow editor showing the `pipeline.yml` file. The file content is identical to the one above, but the `test-run` section is highlighted with a blue background, and the `uses` line in the `steps` section is also highlighted, indicating where it should be placed. The GitHub interface includes a toolbar with "Edit file", "Preview changes", and settings for "Spaces", "2", and "No wrap".

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
```

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

<> Edit file Preview changes Spaces 2 No wrap

```
47   - name: Set timestamp
48     run: echo TDS=$(date +'%Y-%m-%dT%H-%M-%S') >> $GITHUB_ENV
49
50   - name: Tag artifact
51     run: mv build/libs/greetings-add.jar build/libs/greetings-add-${{ github.event.inputs.myVers
52
53
54
55   - name: Upload Artifact
56     uses: actions/upload-artifact@v3
57     with:
58       name: greetings-jar
59       path: |
60         build/libs
61         test-script.sh
62
63
64   - name: Set output
65     id: setoutput
66     run: echo jarpath=build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}>
67
68 print-build-output:
69   runs-on: ubuntu-latest
```

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

<> Code Pull requests Actions Projects Wiki Security Insights Settings

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

<> Edit file Preview changes Spaces 2 No wrap

```
47   - name: Set timestamp
48     run: echo TDS=$(date +'%Y-%m-%dT%H-%M-%S') >> $GITHUB_ENV
49
50   - name: Tag artifact
51     run: mv build/libs/greetings-add.jar build/libs/greetings-add-${{ github.event.inputs.myVers
52
53
54
55   - name: Upload Artifact
56     uses: actions/upload-artifact@v3
57     with:
58       name: greetings-jar
59       path: |
60         build/libs
61         test-script.sh
62
63
64   - name: Set output
65     id: setoutput
66     run: echo jarpath=build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}>
67
68 print-build-output:
69   runs-on: ubuntu-latest
```

Commit changes

Add testing to pipeline

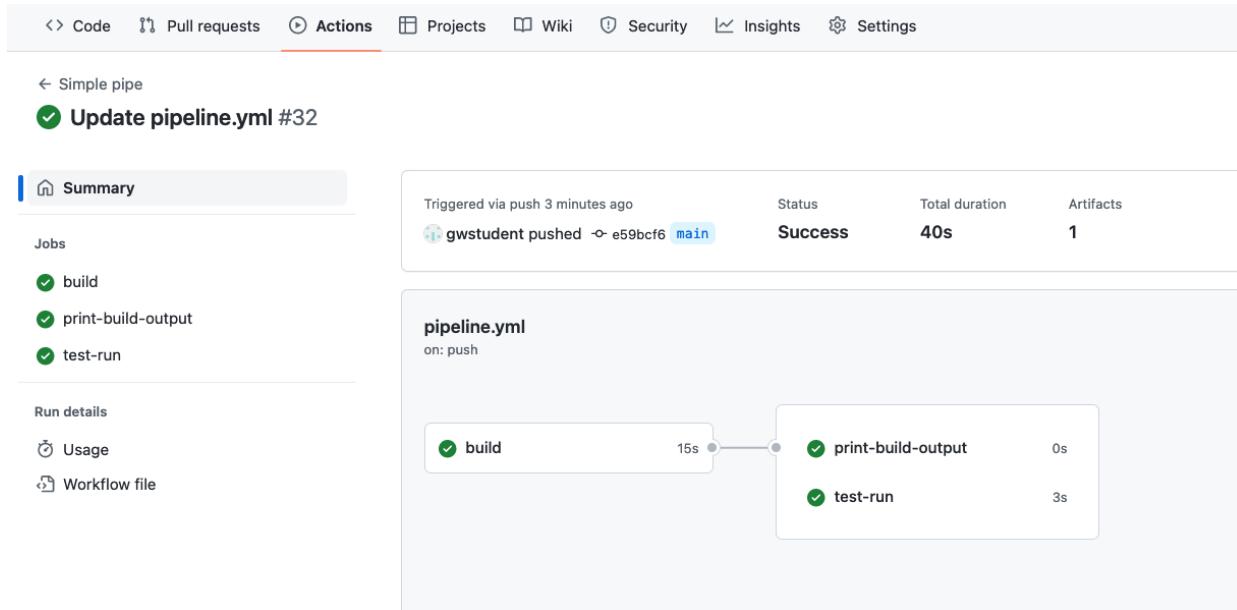
Add an optional extended description...

Commit directly to the main branch.

Create a new branch for this commit and start a pull request. Learn more about pull requests.

Commit changes

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.



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.)

skillrepos / arg-count-action

forked from gwstudent/arg-count-action

About

Simple GitHub Action demo

Releases

8 tags

Packages

No packages published

- 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.

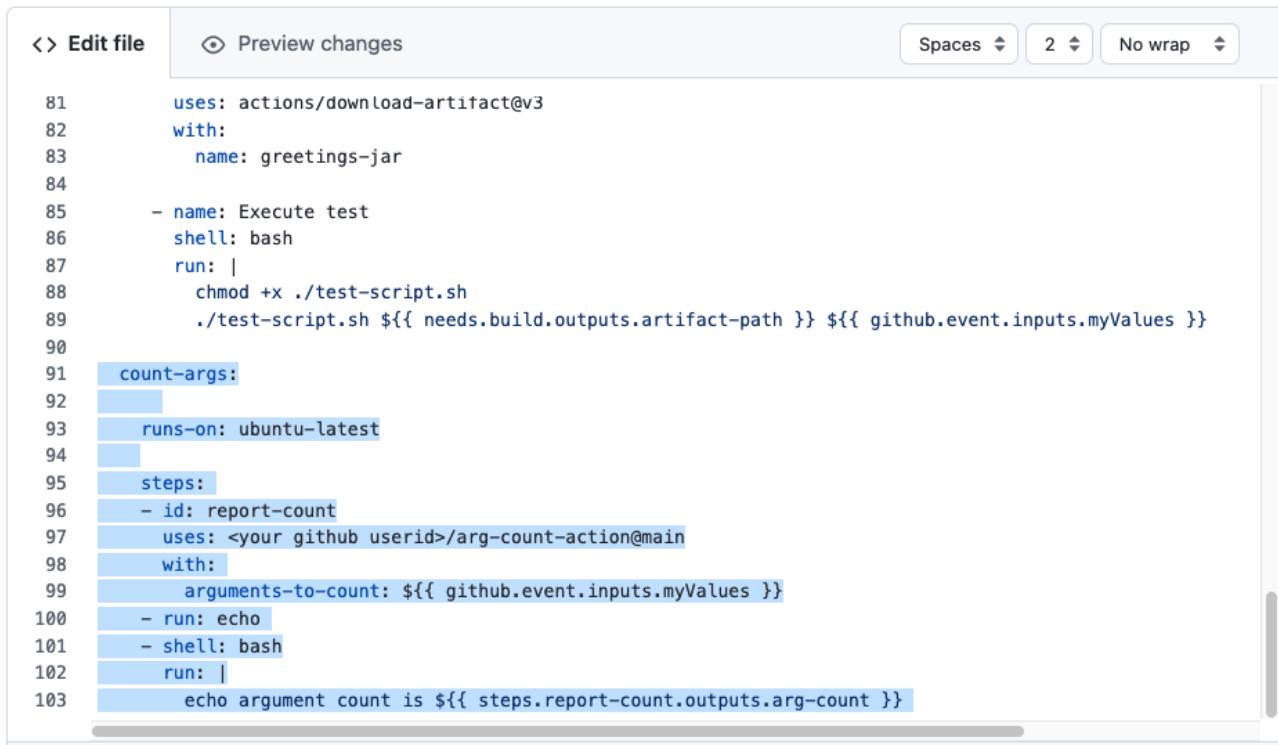
3. Switch back to the file for your original workflow (go back to the greetings-add project and edit the *pipeline.yaml* file in *.github/workflows*. Let's add the code to use this custom action to report the number of arguments passed in. Edit the file and add the code shown below (again indenting the 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 }}
```

[greetings-add / .github / workflows / pipeline.yml](#) in [main](#)



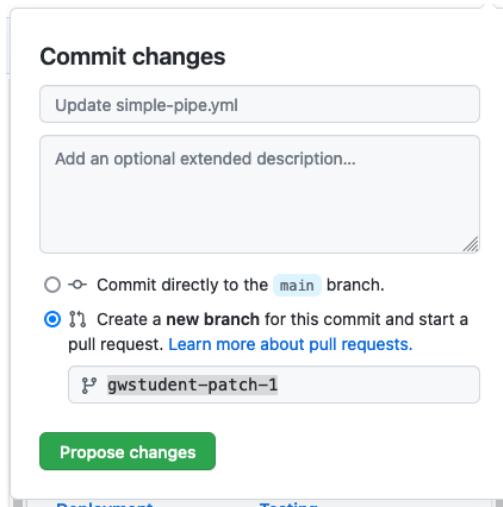
The screenshot shows a GitHub code editor interface with the following details:

- File Path:** greetings-add / .github / workflows / pipeline.yml
- Branch:** main
- Code Content (Line Numbers 91-103):**

```
91   count-args:
92
93     runs-on: ubuntu-latest
94
95     steps:
96       - id: report-count
97         uses: <your github userid>/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 }}
```

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".



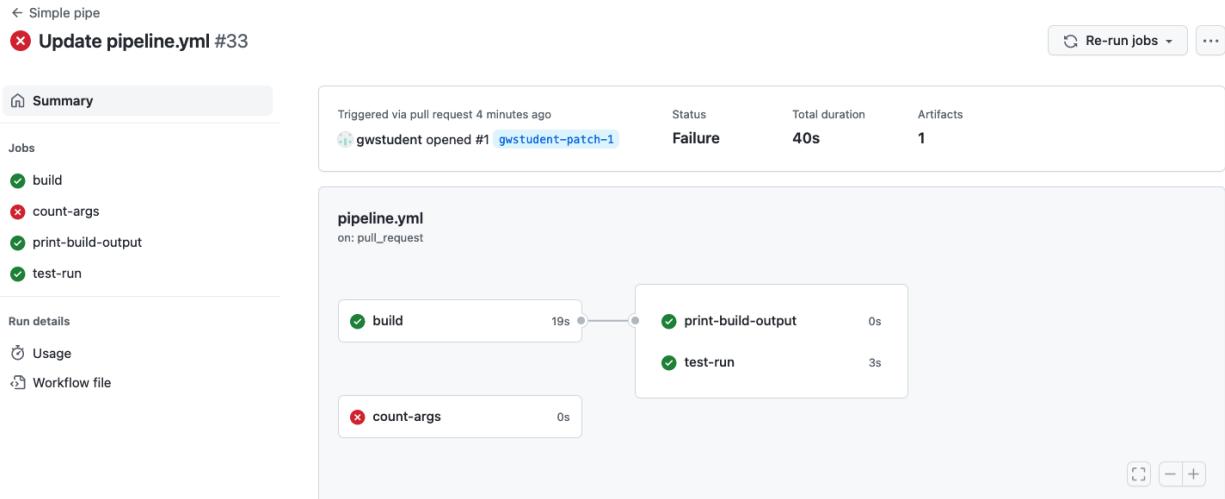
- In the next screen, click on the "Create pull request" button. In the following screen, update the comment if you want and then click on the "Create pull request" button. You'll then see it run through the jobs in our workflow as prechecks for merging.

- 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.

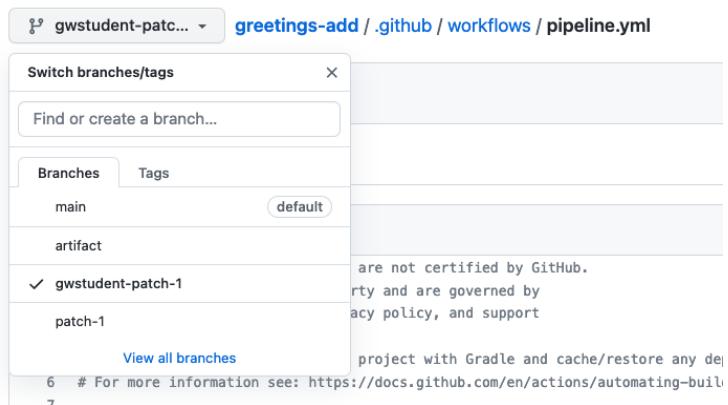
The screenshot shows a GitHub pull request page for 'Update pipeline.yml #1'. At the top, there's a green 'Open' button and a message from 'gwstudent' wanting to merge into 'main' from 'gwstudent-patch-1'. Below this, a summary box says 'Some checks were not successful' with 3 successful and 1 failing check. The failing check is 'Simple pipe / count-args (pull_request)'. A blue oval highlights the 'Details' link for this failing check. 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 checks, a green circle indicates 'This branch has no conflicts with the base branch' and merging can be performed automatically. At the bottom, there's a 'Merge pull request' button and a note about opening in GitHub Desktop or viewing command line instructions.

The screenshot shows the 'Actions' tab for 'Update pipeline.yml #33'. On the left, a sidebar lists jobs: 'build' (green checkmark), 'count-args' (red X), 'print-build-output' (green checkmark), and 'test-run' (green checkmark). The 'count-args' job is selected. The main area shows the 'count-args' job details. It failed 2 minutes ago in 0s. The log shows the setup process and then an error at step 20: 'Error: Unable to resolve action `<your github userid>/arg-count-action@main`, repository not found'. A red oval highlights the 'Summary' link in the sidebar.

- 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.)



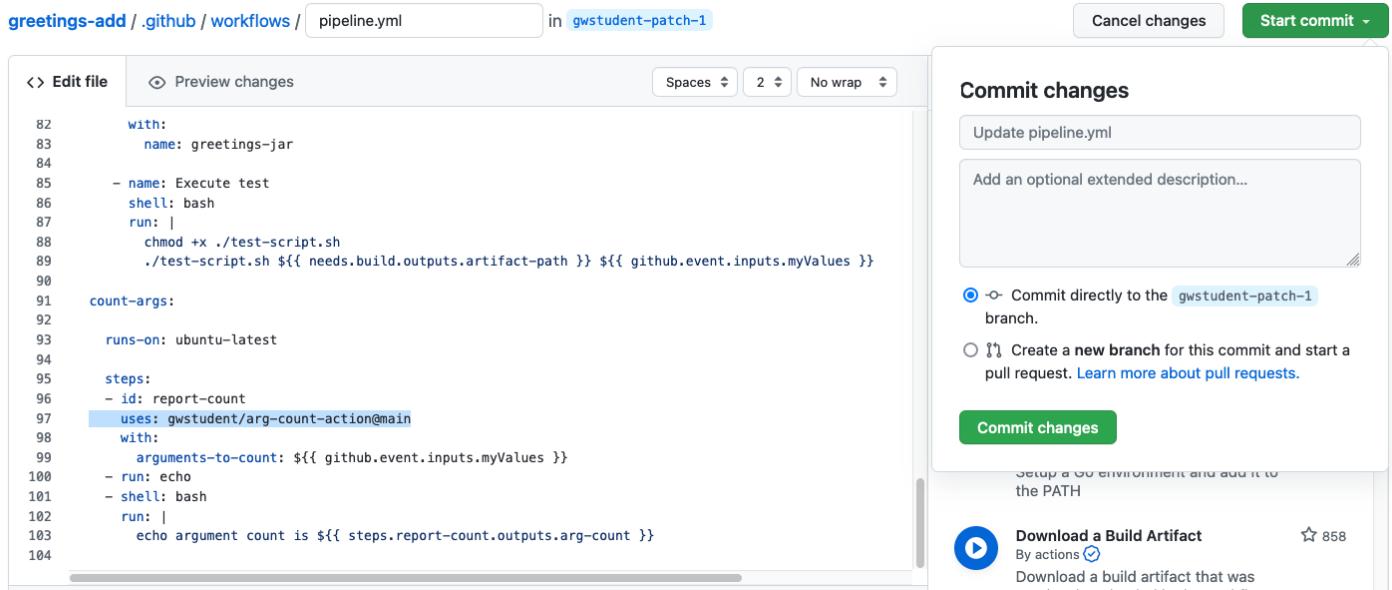
9. Edit the pipeline.yaml file (use the pencil icon). Then update the line that has "uses : <your github userid>/arg-count-action@main" to actually have your GitHub userid in it.

```

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".



The screenshot shows the GitHub commit dialog for a file named `pipeline.yml` in the `gwstudent-patch-1` branch. The code editor on the left contains YAML configuration for a workflow named `greetings-jar`. The commit message field on the right is titled "Commit changes" and contains the placeholder "Update pipeline.yml". Below the message field are two radio button options: one selected for "Commit directly to the gwstudent-patch-1 branch" and another for "Create a new branch for this commit and start a pull request". A large green "Commit changes" button is at the bottom of the dialog.

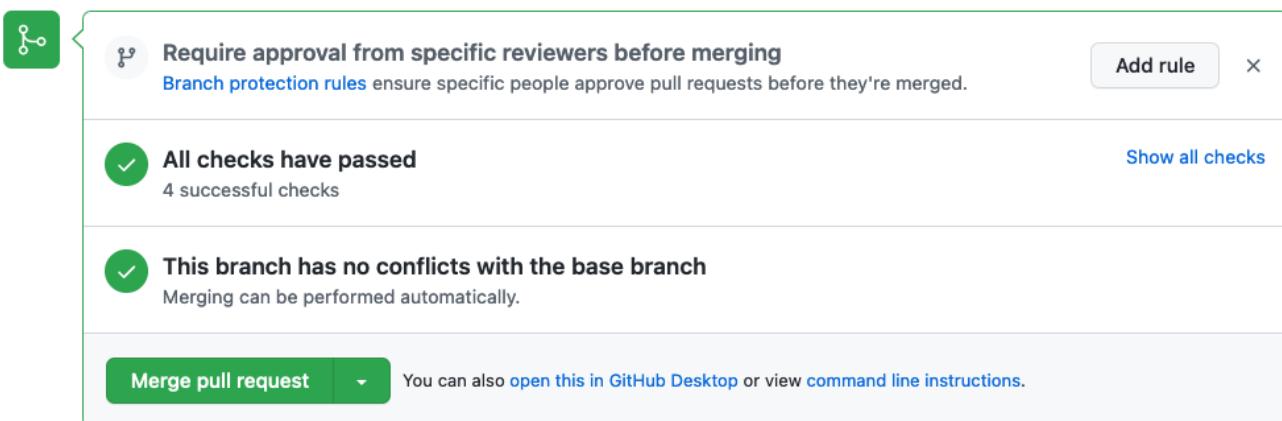
```

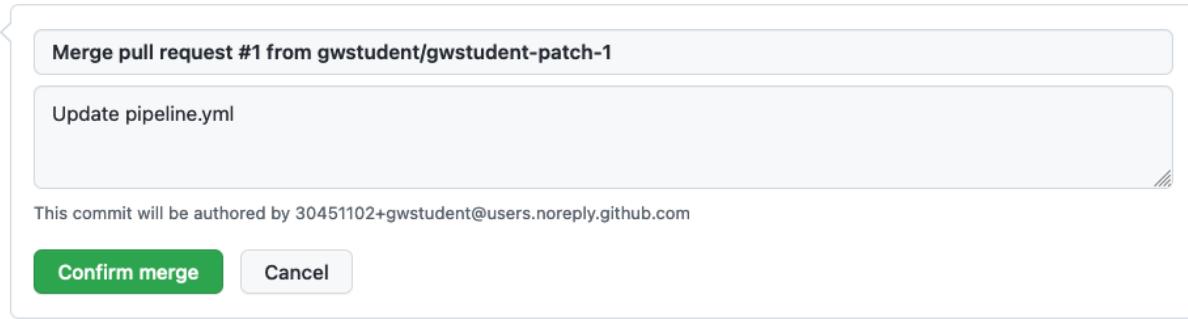
82   with:
83     name: greetings-jar
84
85   - name: Execute test
86     shell: bash
87     run: |
88       chmod +x ./test-script.sh
89       ./test-script.sh ${needs.build.outputs.artifact-path} ${github.event.inputs.myValues}
90
91   count-args:
92
93   runs-on: ubuntu-latest
94
95   steps:
96     - 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}

```

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`.





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

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.

- 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

35 workflow run results

		Event	Status	Branch	Actor
<input checked="" type="checkbox"/>	Merge pull request #1 from gwstudent/gwstudent-patch-1	Simple pipe #35: Commit 04ab81b pushed by gwstudent		main	
<input checked="" type="checkbox"/>	Update pipeline.yml	Simple pipe #34: Pull request #1 synchronize by gwstudent		gwstudent-pa...	

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

Simple pipe pipeline.yml

35 workflow runs

This workflow has a `workflow_dispatch` event trigger.

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

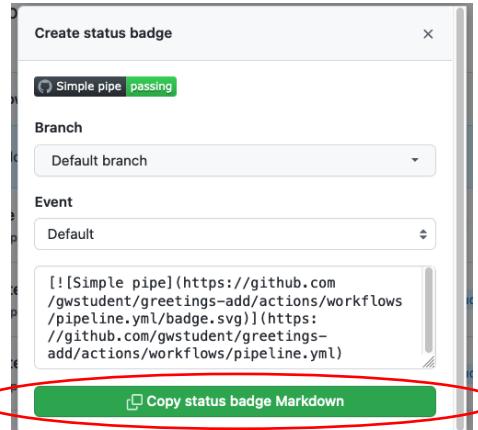
Simple pipe #35: Commit 04ab81b pushed by gwstudent

17 minutes ago 36s

Create status badge

Disable workflow Run workflow

- 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.

Help people interested in this repository understand your project by adding a README.

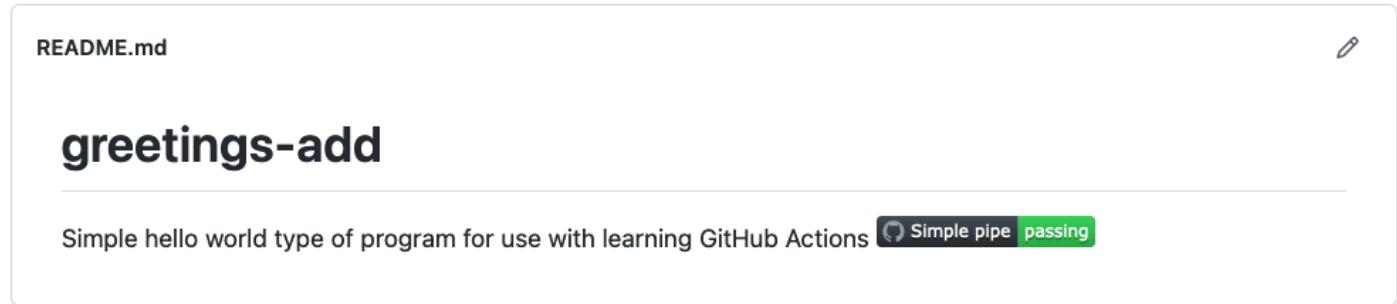
Add a README

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/wor
```

- 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.

Code Pull requests Actions Projects Wiki Security Insights Settings

All workflows Showing runs from all workflows Filter workflow runs

Actions New workflow All workflows Simple pipe Management Caches

36 workflow runs Event Status Branch Actor

Create README.md Simple pipe #36: Commit ff9d6ac pushed by gwstudent main 3 minutes ago 40s

Merge pull request #1 from gwstudent/gwstudent-patch-1 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 third 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' button, a 'Jobs' section containing 'build', 'count-args', 'print-build-output', and 'test-run' (which is selected and highlighted in blue), and 'Run details', 'Usage', and 'Workflow file' buttons. The main area is titled 'test-run' and shows a success message 'succeeded 4 minutes ago in 4s'. It includes a search bar, a gear icon (circled in red), and a 'Re-run all jobs' button. The pipeline steps are listed: 'Set up job' (1s), 'Download candidate artifacts' (1s), 'Execute test' (0s). The 'Execute test' step is expanded, showing four log lines: 'Run chmod +x ./test-script.sh', 'chmod +x ./test-script.sh', './test-script.sh build/libs/greetings-add-2023-01-31T02-14-08.jar', and 'shell: /usr/bin/bash --noprofile --norc -e -o pipefail {0}'. Below this is another step 'Complete job' (0s).

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.

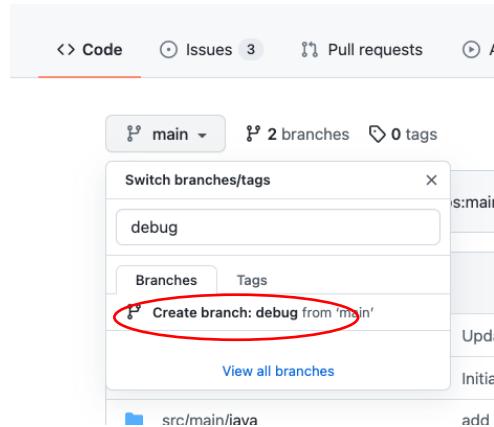
The screenshot shows a browser window with the URL https://pipelines.actions.githubusercontent.com/serviceHosts/1e0ea379-fff2-4162-91e7-7225d42edb94/_apis/pipelines/1/run. The page title is 'Getting Started'. The main content area displays the raw log output for the pipeline run, which is a long list of timestamped log entries. The log starts with '2023-01-31T02:14:14.421642Z Requested labels: ubuntu-latest' and continues through various stages of the pipeline execution, including waiting for a runner, starting the runner, and executing the workflow script.

```
2023-01-31T02:14:14.421642Z 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%F20230122.1
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]
2023-01-31T02:14:22.3749545Z 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.

1. 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.



2. 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 to "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."
```

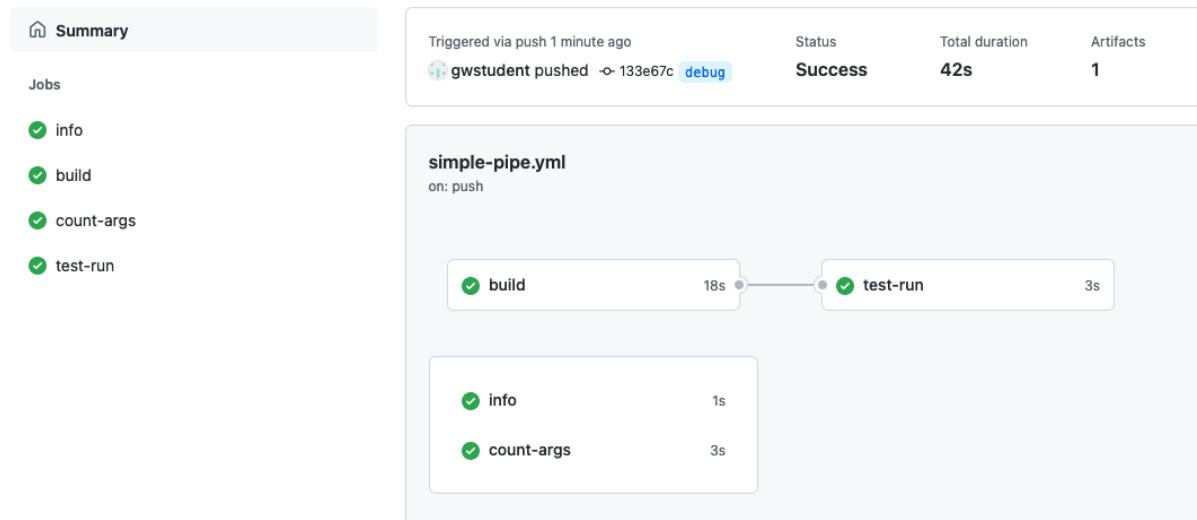
<> Edit file Preview changes

```

5
6   on:
7     push:
8       branches: [ debug ]
9     pull_request:
10    branches: [ debug ]
11   workflow_dispatch:
12   inputs:
13     myValues:
14       description: 'Input Values'
15
16 jobs:
17
18   info:
19     runs-on: ubuntu-latest
20
21   steps:
22     - name: Print warning message
23       run: |
24         echo "::warning::This version is for debugging only."
25     - name: Dump context for runner
26       env:
27         RUNNER_CONTEXT: ${{ toJSON(runner) }}
28       run:
29         echo "::debug::Runner context is above."
30
31
32
33
34
35
36
37
38
39
39
```

- When you are done making the changes, commit as usual. 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 summary indicates it succeeded 2 minutes ago in 1s. The logs pane displays the following output:

```

info
succeeded 2 minutes ago in 1s

Search logs

Set up job
Print warning message
Run echo "::warning::This version is for debugging only."
Warning: This version is for debugging only.
Dump context for runner
Run echo "::debug::Runner context is above."
echo "::debug::Runner context is above."
shell: /usr/bin/bash -e {0}
env:
RUNNER_CONTEXT: {
  "os": "Linux",
  "tool_cache": "/opt/hostedtoolcache",
  "temp": "/home/runner/work/_temp",
  "workspace": "/home/runner/work/greetings-actions"
}

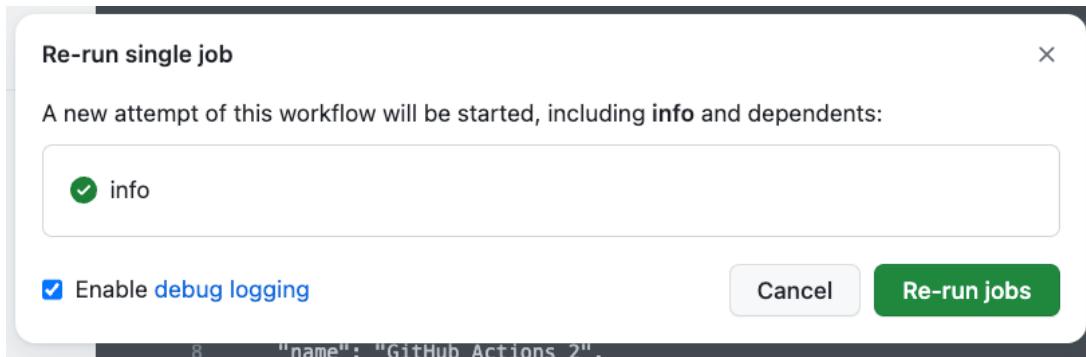
Complete job

```

- 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. Under the "Jobs" section, the "info" job is selected, indicated by a blue vertical bar on its left. To the right of the job list is a "Re-run" button. On the far right of the summary area are a "Search logs" bar, a refresh icon, and a gear icon.

- 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.



8. 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.

```
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', ...
...
```

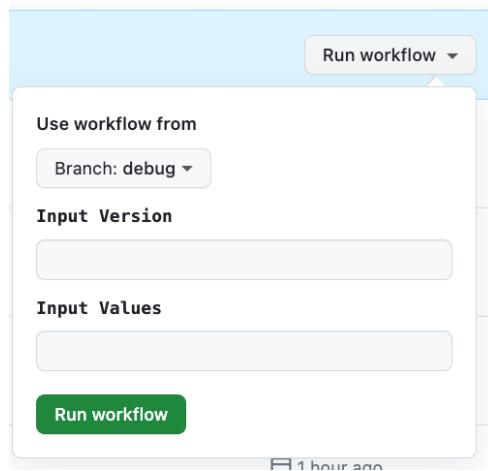
9. However, that doesn't cause debug info to show up for commits. We do that by enabling a secret or a variable for ACTIONS_STEP_DEBUG. Since this setting is not sensitive information, we'll use a variable.
10. To do this, go to the repository's top menu and select "Settings". Then on the left-hand side, 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 circled in red and has a red number '1' above it. On the left sidebar, under the 'Secrets and variables' section, the 'Variables' tab is selected and circled in red with a red number '3' above it. A green button labeled 'New repository variable' is also circled in red with a red number '4' above it. The main content area displays sections for Environment variables and Repository variables, both of which show no variables present.

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

This screenshot shows the 'Actions variables / New variable' creation form. The left sidebar shows the 'Secrets and variables' section is selected and circled in red with a red number '2' above it. The main form fields include a note about variable values being exposed as plain text, a 'Name *' field containing 'ACTIONS_STEP_DEBUG', and a 'Value *' field containing 'true'. Below the value field is a list of validation rules: 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. At the bottom of the form is a green 'Add variable' button, which is circled in red with a red number '5' above it.

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' section containing five entries: 'info' (selected), 'build', 'count-args', and 'test-run'. The main area shows the 'info' job details: 'info' succeeded 32 seconds ago in 1s. Below this, under the 'Dump context for runner' section, a code block shows several lines of log output. Lines 38 and 39 are circled in red, highlighting the message '##[debug]Runner context is above.'

```
info
succeeded 32 seconds ago in 1s
Search logs
0s

info
Dump context for runner
15 ##[debug] "tool_cache": "/opt/hostedtoolcache",
16 ##[debug] "temp": "/home/runner/work/_temp",
17 ##[debug] "workspace": "/home/runner/work/greetings-actions"
18 ##[debug]{
19 ##[debug]Evaluating condition for step: 'Dump context for runner'
20 ##[debug]Evaluating: success()
21 ##[debug]Evaluating success:
22 ##[debug]=> true
23 ##[debug]Result: true
24 ##[debug]Starting: Dump context for runner
25 ##[debug]Loading inputs
26 ##[debug]Loading env
27 ▶ Run echo ::debug::Runner context is above.
28 ##[debug]/usr/bin/bash e /home/runner/work/_temp/4282ca63-108c-4656-8c6a-4a05c1ff90b3.sh
29 ##[debug]Runner context is above.
40 ##[debug]Finishing: Dump context for runner
```

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, there's a sidebar with 'Actions' selected, showing 'All workflows' and a 'Simple pipe' workflow. The 'Simple pipe' workflow has 40 workflow runs. A modal window is open for one of the runs, specifically for commit #37. The modal shows the 'Use workflow from' dropdown set to 'Branch: main', the 'Input Version' field containing '1.0.2-', and the 'Input Values' field containing the value 'ls -la'. A green 'Run workflow' button is visible. The run itself has a status of 'In Progress'.

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 a 'Summary' section and a 'Jobs' section containing five items: 'build', 'count-args', 'print-build-output', 'test-run' (which is selected), and 'create-issue-on-failure'. Below that are 'Run details', 'Usage', and 'Workflow file'. The main area is titled 'test-run' and shows a log entry from 15 minutes ago. The log details the execution of the 'Execute test' step, which runs a shell command to chmod +x the test-script.sh file and then execute it with arguments derived from the pipeline environment.

```

test-run
succeeded 15 minutes ago in 3s
Execute test 0s
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 (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. It displays the 'greetings-add/.github/workflows/pipeline.yml' file. The 'test-run' step has been modified. The 'env:' block now contains the variable 'ARGS' set to the value of 'github.event.inputs.myValues'. The 'run:' block contains a single line of bash commands: 'chmod +x ./test-script.sh' followed by './test-script.sh \${{ needs.build.outputs.artifact-path }} "\$ARGS"'.

```

greetings-add/.github/workflows/pipeline.yml in main
<> Edit file Preview changes Spaces 2 N
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 a repository named 'Simple Pipe'. On the left, there's a sidebar with 'Actions', 'New workflow', 'All workflows', 'Simple Pipe' (which is selected and highlighted in blue), 'Management', and 'Caches'. The main area is titled 'Simple Pipe pipeline.yml' and shows '16 workflow runs'. A message says 'This workflow has a workflow_dispatch event trigger.' Below this, four workflow runs are listed:

- Update pipeline.yml** (main branch): Simple Pipe #16: Commit 047f1a5 pushed by gwstudent2
- Simple Pipe**: Simple Pipe #15: Manually run by gwstudent2
- Simple Pipe**: Simple Pipe #14: Manually run by gwstudent2
- Update pipeline.yml** (dahun branch): [status pending]

To the right of the runs, there are filters for 'Event', 'Status', 'Branch', and 'Actor'. A search bar at the top right says 'Filter workflow runs'. A large button labeled 'Run workflow' is visible. On the far right, there's a timestamp '24 minutes ago' and three dots for more options.

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.

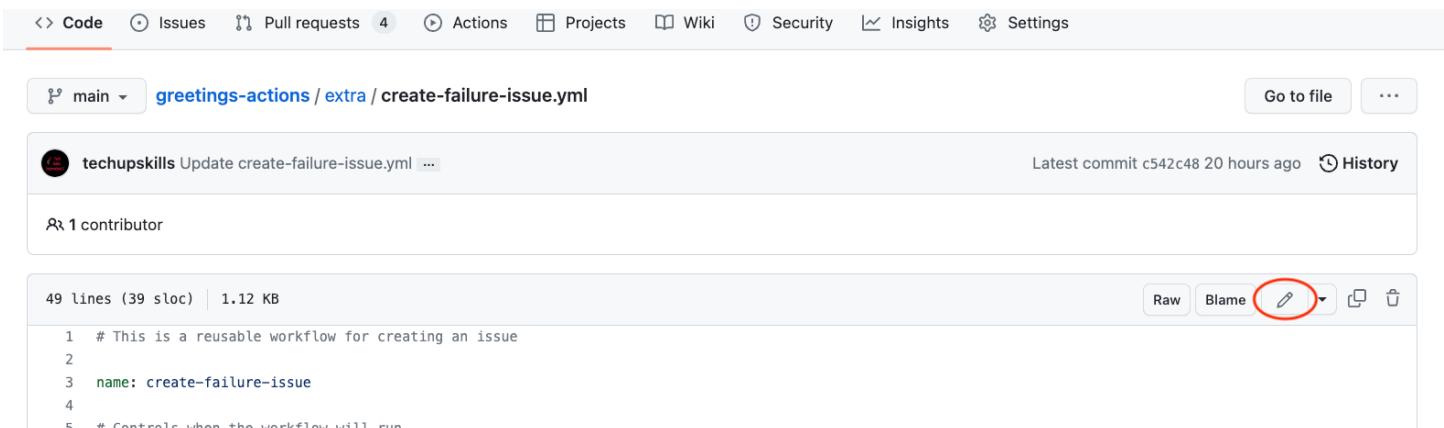
1. We're going to leverage a reusable workflow that will be able to automatically create a GitHub issue in our repository. And then we will invoke that workflow from our current workflow. But first, we need to ensure that the "Issues" functionality is turned on for this repository. Go to the project's Settings main page, scroll down and under "Features", make sure the "Issues" selection is checked.

The screenshot shows the 'Getting Started' page of a GitHub repository. On the left, there are links for 'Pages' and 'Moderation settings'. In the center, there's a 'Features' section with the following checkboxes:

- Wikis
- Restrict editing to collaborators only
- Issues

A note below the 'Issues' checkbox says: '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 call-to-action: 'Get organized with issue templates' and 'Set up templates'.

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.

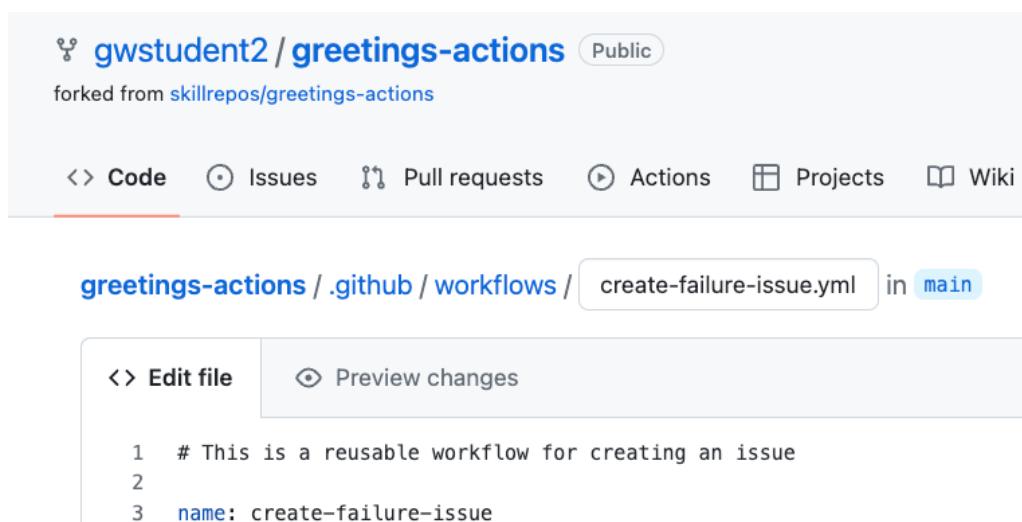


```

<> Code Issues Pull requests 4 Actions Projects Wiki Security Insights Settings
main greetings-actions / extra / create-failure-issue.yml Go to file ...
techupskills Update create-failure-issue.yml ... Latest commit c542c48 20 hours ago History
1 contributor
49 lines (39 sloc) | 1.12 KB Raw Blame ⌂ ⓘ
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".



gwstudent2 / greetings-actions Public
forked from skillrepos/greetings-actions

<> Code Issues Pull requests Actions Projects Wiki

greetings-actions / .github / workflows / create-failure-issue.yml in main

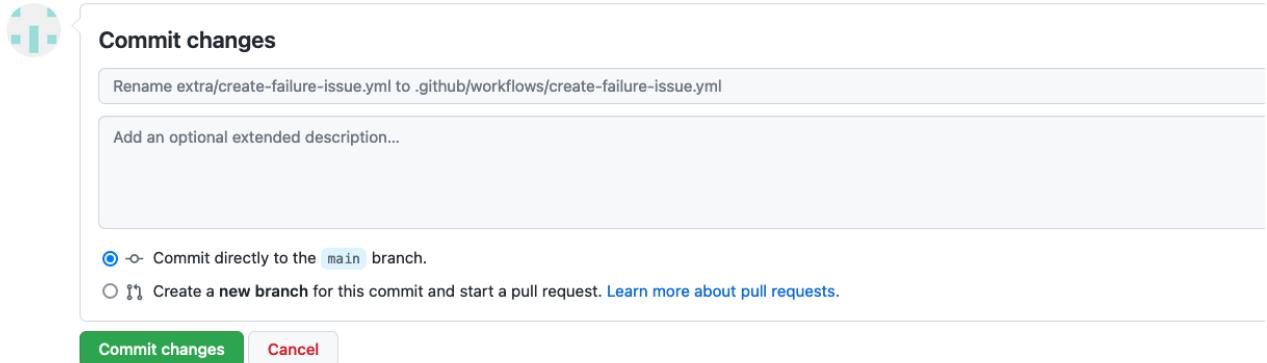
<> Edit file ⌂ Preview changes

```

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

```

- e. To complete the change, scroll to the bottom of the page, and click on the green "Commit changes" button.



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

The screenshot shows the GitHub Workflows page. On the left, under 'Workflows', the 'create-failure-issue' workflow is selected. It shows '1 workflow run'. A message says 'This workflow has a workflow_dispatch event trigger.' A 'Run workflow' button is visible. A modal window is open, titled 'Run workflow'. It contains fields for 'Issue title' (with placeholder 'This is a title') and 'Issue body' (with placeholder 'This is the body text.'). Buttons for 'Use workflow from' (branch: main), 'Run workflow', and 'Cancel' are also present.

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

The screenshot shows the GitHub Issues page. The top navigation bar includes 'Code', 'Issues 1', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', 'Settings', and a 'New issue' button. The 'Issues' tab is active. A search bar shows 'is:issue is:open'. Below it, filters show '1 Open' and '0 Closed'. A search result for '#2 opened now by github-actions bot' is shown, with a green circle icon and the text 'FAILURE: This is a title'.

- 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.)

create-issue-on-failure:

```
permissions:
  issues: write
needs: [test-run, count-args]
if: always() && failure()
uses: ./github/workflows/create-failure-issue.yml
with:
  title: "Automated workflow failure issue for commit ${{ github.sha }}"
  body: "This issue was automatically created by the 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.)

greetings-actions / .github / workflows / simple-pipe.yml in gwstudent2:main

```
1   - shell: bash
2     run: |
3       echo argument count is ${{ steps.report-count.outputs.arg-count }}
4       exit 1
5
6   create-issue-on-failure:
7
8     permissions:
9       issues: write
10    needs: [test-run, count-args]
11    if: always() && failure()
12    uses: ./github/workflows/create-failure-issue.yml
13    with:
14      title: "Automated workflow failure issue for commit ${{ github.sha }}"
15      body: "This issue was automatically created by the GitHub Action workflow ** ${{ github.workflow }} **"
```

- 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.

Workflows New workflow

All workflows

Simple Pipe

create-failure-issue

All workflow runs

15 workflow runs

Event ▾ Status ▾ Branch ▾ Actor ▾

✓ create-failure-issue
create-failure-issue #2: Manually run by gwstudent
15 seconds ago 13s

✗ Update simple-pipe.yml
Simple Pipe #14: Commit 1539282 pushed by gwstudent
main 1 minute ago 49s

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

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

Re-run jobs ▾ ...

Summary

Triggered via push 3 minutes ago
gwstudent pushed -> 1539282 main

Status Failure Total duration Artifacts

Jobs

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

simple-pipe.yml
on: push

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

Re-run jobs ▾ ...

Summary

Manually triggered 3 minutes ago
gwstudent -> 1539282

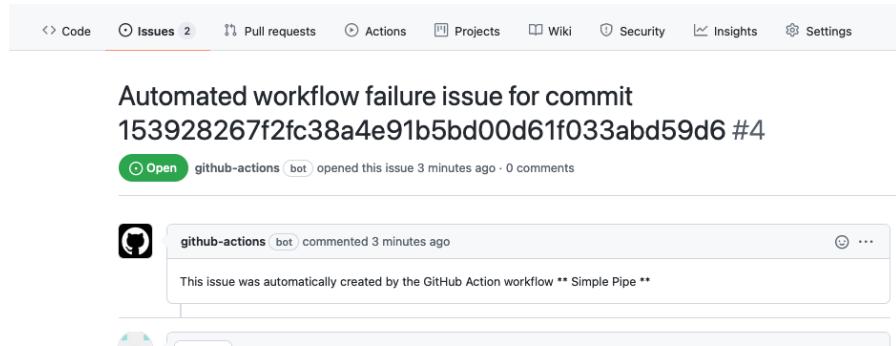
Status Success Total duration Artifacts

Jobs

create_issue_on_failure (✓)

create-failure-issue.yml
on: workflow_dispatch

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



THE END - THANKS!