

GitHub Actions Deep Dive

Revision 2.0 – 04/26/23

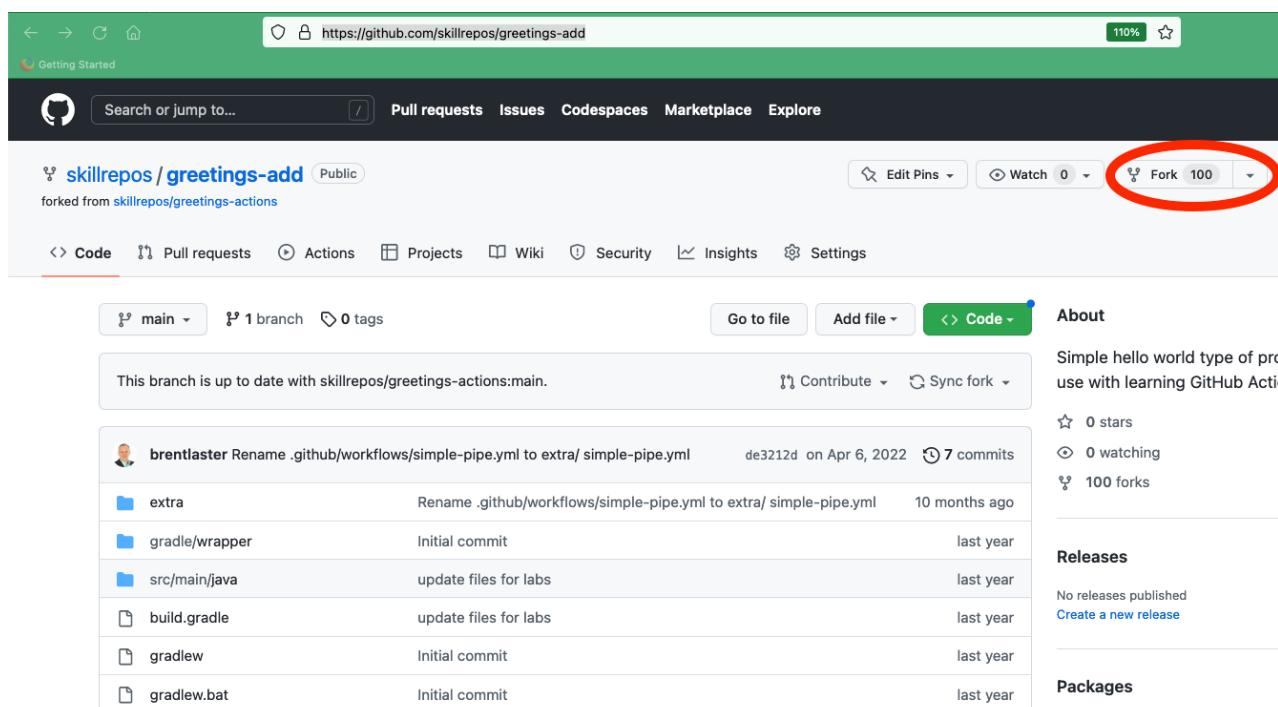
Tech Skills Transformations LLC / Brent Laster

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.



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.

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
- Stale By GitHub Actions Checks for stale issues and pull requests
- Manual workflow By GitHub Actions Simple workflow that is manually triggered.
- Labeler By GitHub Actions Labels pull requests based on the files changed

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
By GitHub Actions	Configure	By GitHub Actions	Configure	By GitHub Actions
Build an Android project with Gradle.	Java	Build and test a Java project with Apache Ant.	Java	Build and test a Clojure project with Leiningen.
Publish Java Package	Java	Java with Gradle	Java	Publish Java Package

- 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. A search bar at the top has 'Grade' typed into it. Below the search bar, there's a 'Categories' dropdown menu with 'Continuous Integration' selected. On the right, a search result for 'Grade' is displayed, showing four workflows: 'Android CI', 'SLSA Generic generator', 'Publish Java Package with Gradle', and 'Java with Gradle'. The 'Java with Gradle' workflow is circled in red, and its 'Configure' button is also circled.

- 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.)

The screenshot shows the GitHub code editor with the file path 'greetings-actions/.github/workflows/gradle.yml' highlighted in blue. The code editor interface includes tabs for 'Edit new file' and 'Preview', and settings for 'Spaces', '2', and 'No wrap'. The code itself is a simple YAML snippet:

```

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

```

TO

The screenshot shows the GitHub code editor with the file path 'greetings-add/.github/workflows/pipeline.yaml' highlighted in blue. The code editor interface includes tabs for 'Edit new file' and 'Preview', and settings for 'Spaces', '2', and 'No wrap'. The code is identical to the 'gradle.yml' file:

```

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.

```

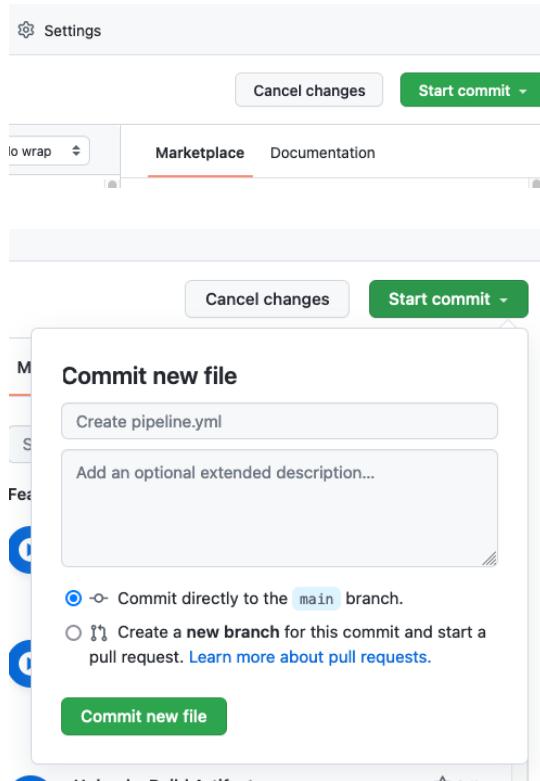
- Now, edit the name of the workflow - change line 8 from "name: Java CI with Gradle" to "name: Simple Pipe".

```

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



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.

All workflows	
1 workflow run	Event ▾ Status ▾ Branch ▾ Actor ▾
Create pipeline.yml	main
Simple pipe #1: Commit 2cac6a7 pushed by gwstudent	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	
1 workflow run	Event ▾ Status ▾ Branch ▾ Actor ▾
Create pipeline.yml	main
Simple pipe #1: Commit 2cac6a7 pushed by gwstudent	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. (You can ignore the "save-state" messages in the outputs.)

The screenshot shows the GitHub Actions interface for a repository named 'gwstudent / greetings-add'. The 'Actions' tab is selected. A workflow named 'Create pipeline.yml #1' is shown. The 'build' job is highlighted with a red circle. The job summary indicates it succeeded 2 minutes ago in 24s. The job details show two steps: 'Set up job' and 'Run actions/checkout@v3'. The 'Run actions/checkout@v3' step is expanded, showing sub-steps 1 through 17. The sub-step 17 is a temporary override for the HOME environment variable.

* 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 interface for the 'gwstudent / greetings-add' repository. The 'Actions' tab is selected. The 'Simple pipe' workflow is selected. The 'pipeline.yml' link is highlighted with a red circle. Below it, a table shows the 'Simple pipe' workflow run, which has 1 workflow run. The run was created by 'Create pipeline.yml' and pushed by 'gwstudent' on the 'main' branch 4 hours ago. It took 35s.

- Once the file opens up, click on the pencil icon 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:

```

- 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 GitHub Actions. We're going to add a job to our workflow to upload an artifact. Let's find actions related to uploading.

In the "Search Marketplace for Actions" box on the upper right, enter "Upload" 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

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 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.

GitHub Action
Checkout
v3.0.0 [Latest version](#)

Checkout V3

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 2.205.0 to run, which is by default

Links

- [actions/checkout](#) (circled in red)
- [Open issues](#)
- 209

- 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.

actions / checkout

Actions (circled in red)

About
Action for checking out a repo
[github.com/features/actions](#)

Releases 16
v2.3.4 (Latest)
on Nov 3, 2020
+ 15 releases

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	...

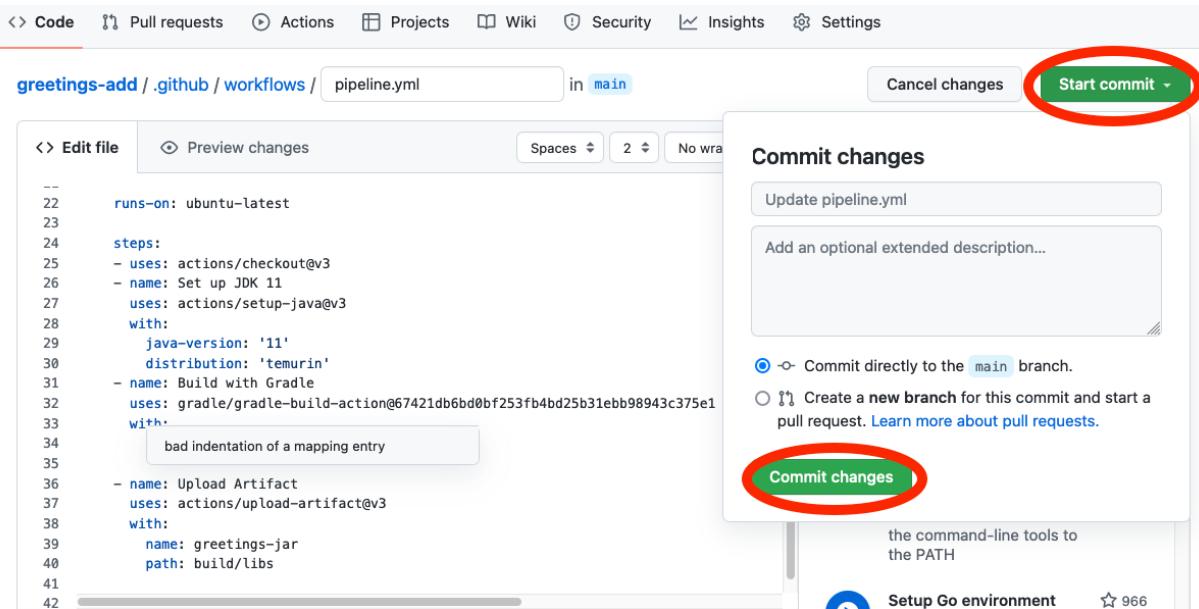
6. Switch back to the browser tab where you are editing the workflow for greetings-actions. Update the build job to include a new step to use the "upload-artifact" action to upload the jar the build job creates. To do this, add the following lines inline with the build job steps. Pay attention to the 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 "Start commit" 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	<input checked="" type="checkbox"/>	Build Scan™ NOT PUBLISHED

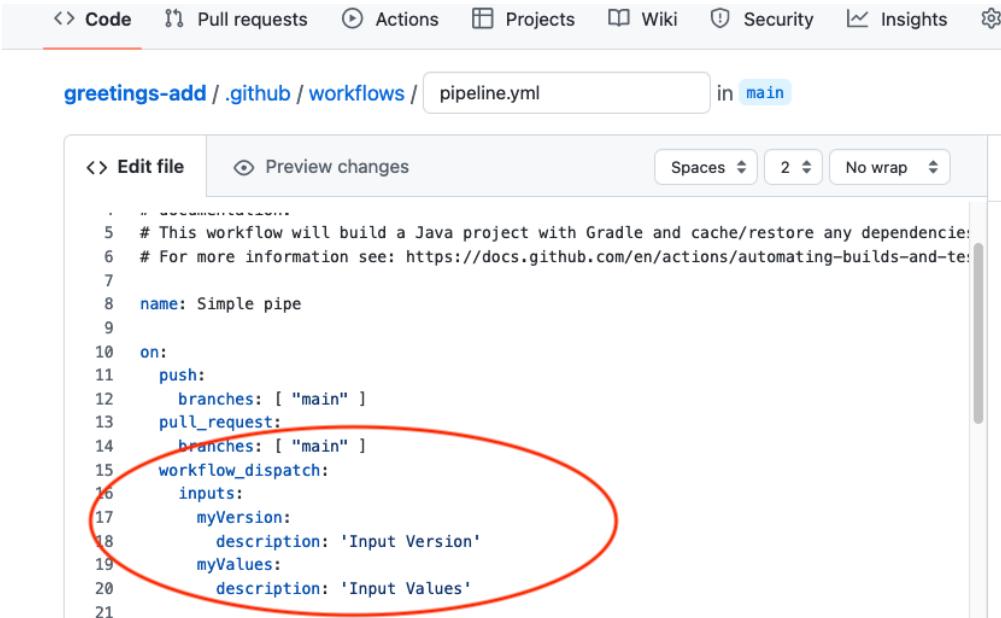
* END OF LAB *

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'
```



A screenshot of the GitHub Actions pipeline editor. The URL in the address bar is `greetings-add/.github/workflows/pipeline.yml`. The code editor shows a YAML configuration file. A red oval highlights the `workflow_dispatch` section, which contains the input definitions for `myVersion` and `myValues`.

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

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.

```

- 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
56

```

4. 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.

The screenshot shows a GitHub Actions workflow run named 'build'. The 'Summary' tab is selected. In the 'Jobs' section, the 'build' job is shown as succeeded 7 minutes ago. The 'Tag artifact' step is expanded, showing the command `mv build/libs/greetings-add.jar build/libs/greetings-add-2023-01-30T00-57-54.jar` and environment variables like JAVA_HOME, GRADLE_BUILD_ACTION_SETUP_COMPLETED, and TDS. The 'Upload Artifact' step is also expanded, showing the command `actions/upload-artifact@v3` and similar environment variables. The logs for both steps are visible at the bottom of their respective sections.

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".

6. 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.

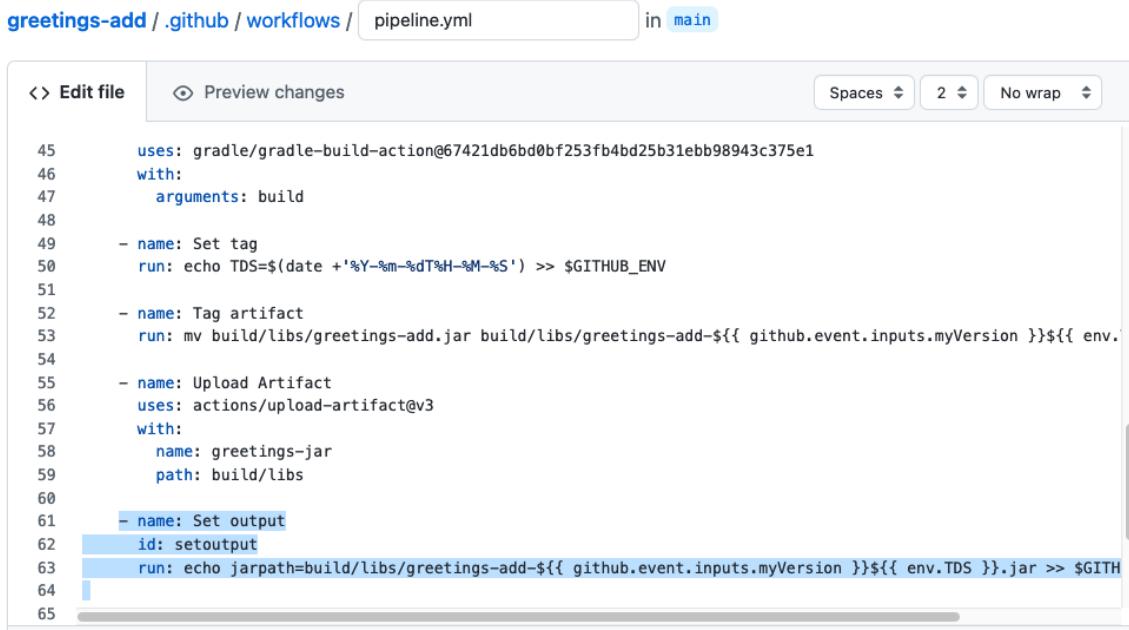
Name	Size	Kind	Date Added
greetings-add-1.0.0-2023-01-30T01-17-11.jar	1 KB	Java JAR file	Today at 8:20 PM
greetings-add-1.0.02023-01-30T01-13-37.jar	1 KB	Java JAR file	Today at 8:14 PM
greetings-add-2023-01-30T00-52-59.jar	1 KB	Java JAR file	Today at 7:53 PM
greetings-add.jar	1 KB	Java JAR file	Today 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

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
<> Edit file Preview changes Spaces 2 No wrap
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 >> $GITH
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](#)

```

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
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.

Simple pipe

Update pipeline.yml #28

Summary

Re-run triggered 1 minute ago
gwstudent -> 1167568 main

Status: Success | Total duration: 37s | Artifacts: 1

pipeline.yml
on: push

build 19s → print-build-output 0s

Simple pipe

Update pipeline.yml #28

Summary

Jobs: build, 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 {}
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.

Code Pull requests Actions Projects Wiki Security Insights Settings

main · greetings-add / extra / test-script.sh Go to file ⋮

techupskills Create test-script.sh Latest commit bc2bd9d 9 hours ago History

1 contributor

9 lines (7 sloc) | 204 Bytes Raw Blame Edit this file

```
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.

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}

```

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

```

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

```

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.myVersion }}.jar
52
53
54      - name: Upload Artifact
55        uses: actions/upload-artifact@v3
56        with:
57          name: greetings-jar
58          path: |
59            build/libs
60            test-script.sh
61
62
63
64      - name: Set output
65        id: setoutput
66        run: echo jarpath=build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}>> $GITHUB_OUTPUT
67
68      print-build-output:
69        runs-on: ubuntu-latest

```

- 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.myVersion }}.jar
52
53
54      - name: Upload Artifact
55        uses: actions/upload-artifact@v3
56        with:
57          name: greetings-jar
58          path: |
59            build/libs
60            test-script.sh
61
62
63
64      - name: Set output
65        id: setoutput
66        run: echo jarpath=build/libs/greetings-add-${{ github.event.inputs.myVersion }}${{ env.TDS }}>> $GITHUB_OUTPUT
67
68      print-build-output:
69        runs-on: ubuntu-latest

```

Commit changes

Add testing to pipeline

Add an optional extended description...

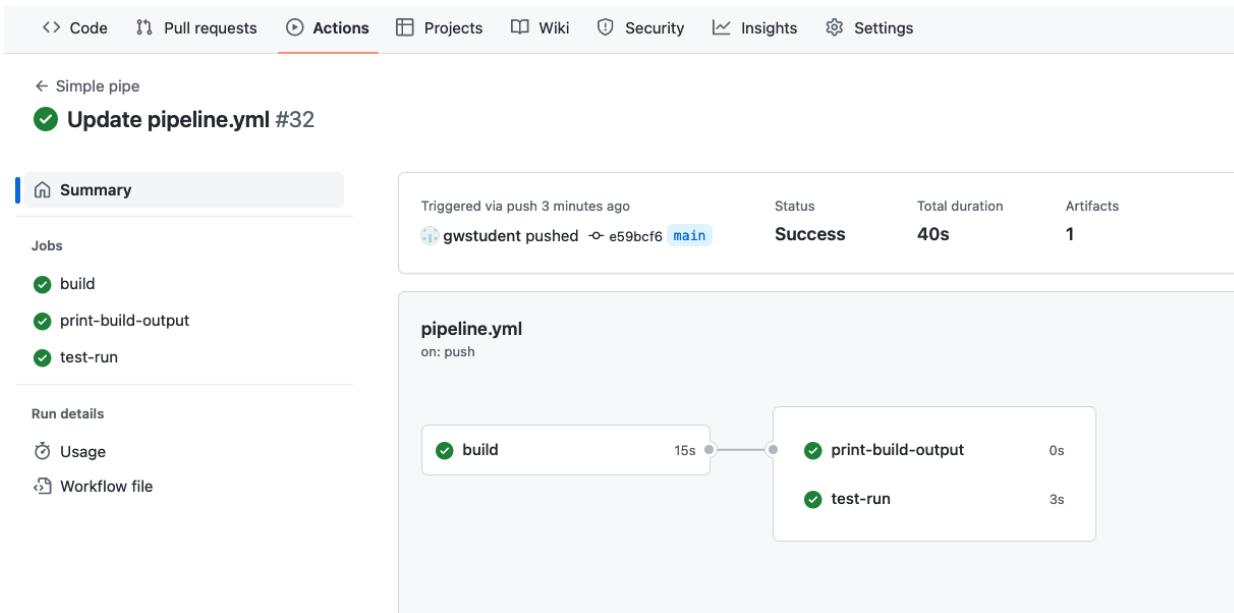
Commit directly to the `main` branch.

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

Commit changes

Cancel changes Start commit

- 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.)

The screenshot shows the GitHub repository for "skillrepos/arg-count-action". The "Fork" button is highlighted with a red oval. The repository description is "Simple GitHub Action demo". It shows a main branch that is 1 commit ahead of gwstudent:main. There are two files: "action.yml" and "count-args.sh". The "action.yml" file contains the primary logic for the action.

- 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 }}
```

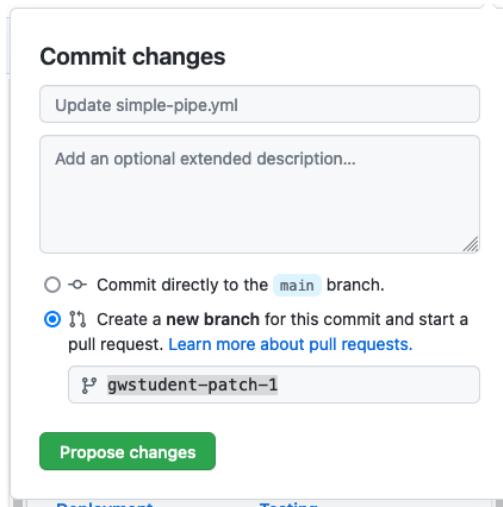
[greetings-add / .github / workflows / pipeline.yml](#) in [main](#)

The screenshot shows a GitHub code editor interface. At the top, there are tabs for 'Edit file', 'Preview changes', and file navigation buttons ('Spaces', '2', 'No wrap'). Below the tabs, the code is displayed with line numbers on the left. The 'count-args' step is highlighted with a blue selection bar. The code itself is as follows:

```
81   uses: actions/download-artifact@v3
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: <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 "Start commit" 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".



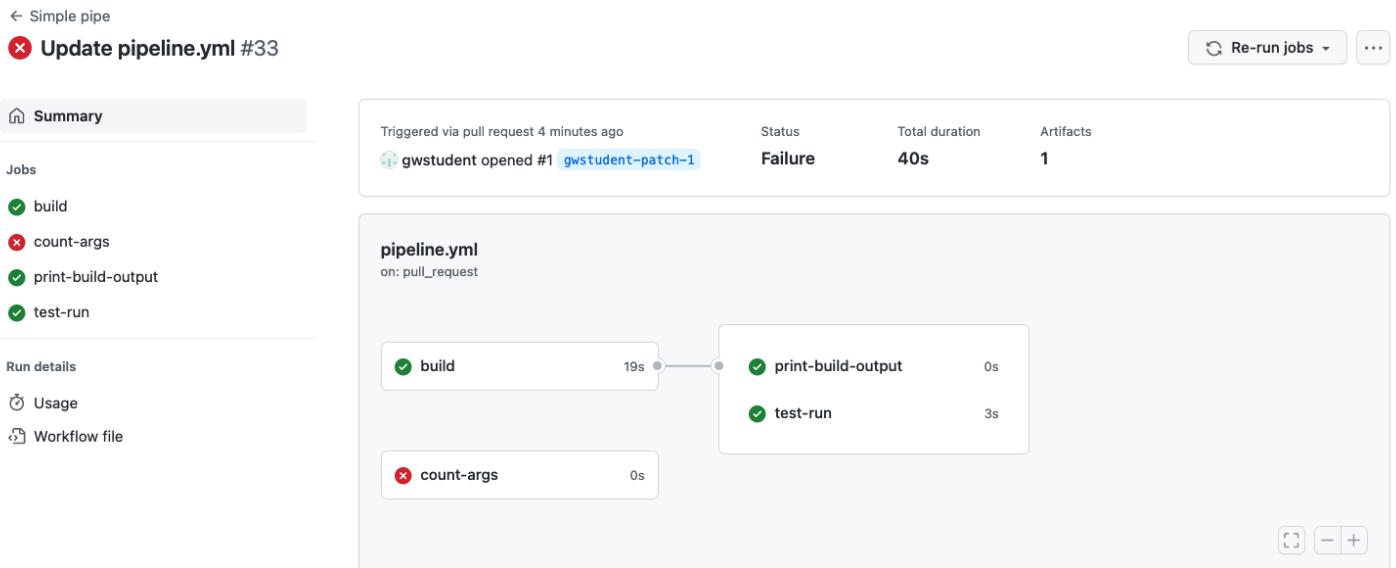
- In the next screen, click on the "Create pull request" button. In the following screen, update the comment if you want and then click on the "Create pull request" button. You'll then see it run through the jobs in our workflow as prechecks for merging.

- 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.

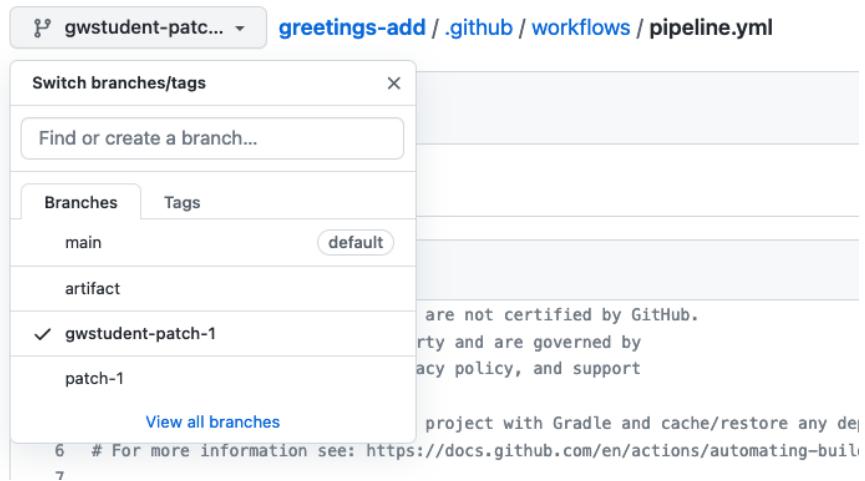
The screenshot shows a GitHub pull request page for 'Update pipeline.yml #1'. At the top, there's a green 'Open' button and a message from 'gwstudent' wanting to merge 1 commit into 'main' from 'gwstudent-patch-1'. Below this, a summary box says 'Some checks were not successful' with 3 successful and 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 it in GitHub Desktop or viewing command line instructions.

The screenshot shows the 'Actions' tab for the 'Update pipeline.yml #33' pull request. In the left sidebar, under 'Jobs', the 'count-args' job is highlighted with a red oval around its name. The main area shows the 'count-args' job details. It failed 2 minutes ago in 0s. The log output 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 search bar for logs is visible at the top right.

- 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.yml** 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.yml 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

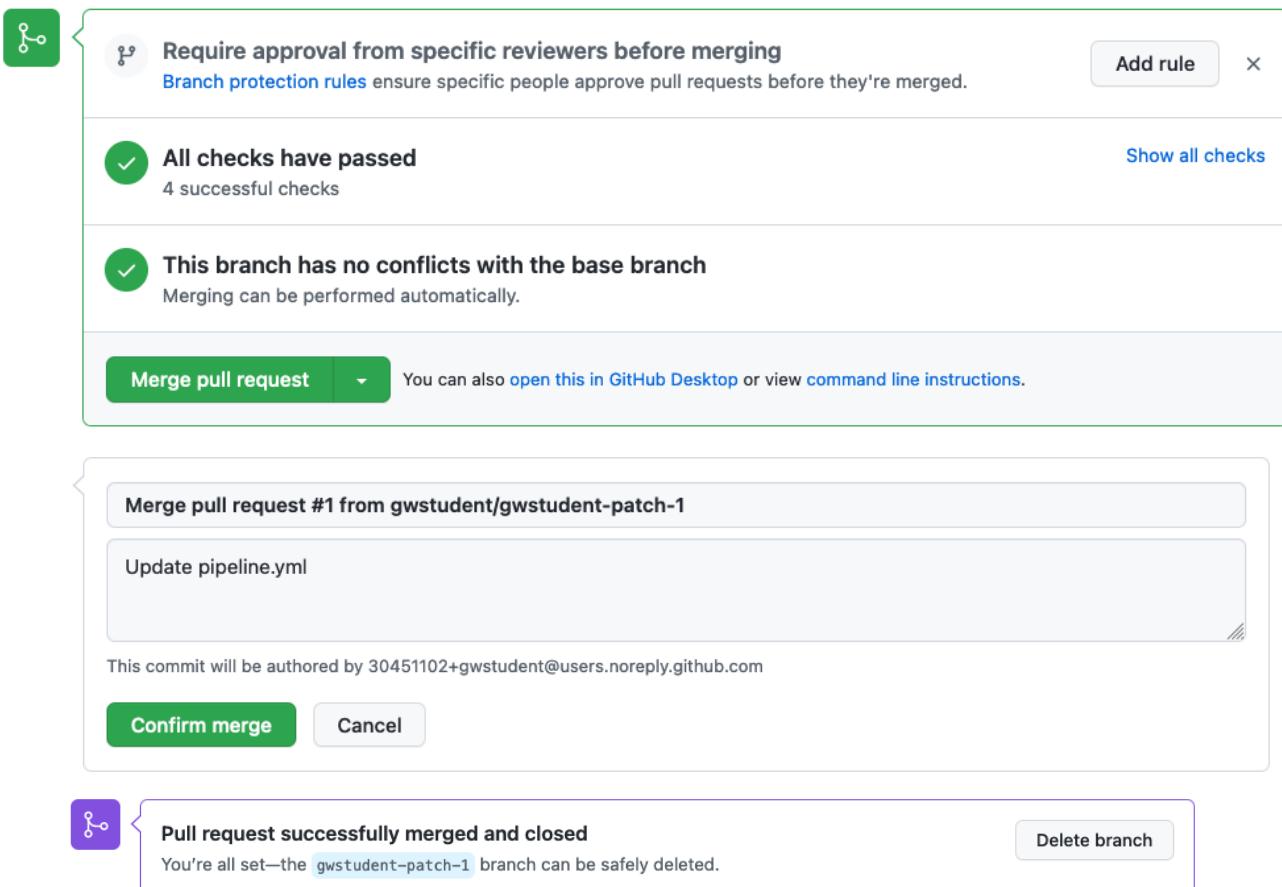
```

- When you're done, click on the green "Start commit" 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 pull request interface. On the left, the workflow file `pipeline.yml` is being edited. The right side shows the "Commit changes" dialog, which includes fields for a commit message ("Update pipeline.yml") and a description ("Add an optional extended description..."). It also contains 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". Below these options is a "Commit changes" button. At the bottom of the dialog, there is a section for artifacts with a "Download a Build Artifact" button.

- 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 GitHub pull request merge interface. At the top, a green button says "Merge pull request". Below it, a message says "All checks have passed" and "This branch has no conflicts with the base branch". A "Merge pull request" button is highlighted in green. In the middle, there's a "Merge pull request #1 from gwstudent/gwstudent-patch-1" button. Below that, a "Confirm merge" button is highlighted in green. A message at the bottom says "Pull request successfully merged and closed".

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



The screenshot shows the GitHub Actions workflow runs page. On the left, a sidebar lists "Actions" and "All workflows". Under "All workflows", items include "Simple pipe", "Management", and "Caches". On the right, a table shows "35 workflow runs". One specific run is highlighted: "Merge pull request #1 from gwstudent/gwstudent-patch-1" (Event: main, Status: success, Actor: gwstudent, Run ID: 35, Started: 10 minutes ago, Duration: 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.

The screenshot shows the GitHub Actions interface with the search bar set to 'branch:gwstudent-patch-1'. The results show two workflow runs:

- Simple pipe #34: Pull request #1 synchronize by gwstudent** (gwstudent-patch-1, 39s ago)
- Update pipeline.yml** (gwstudent-patch-1, 40s ago)

- 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.

The screenshot shows the GitHub Actions interface with the search bar set to 'Filter workflow runs'. A dropdown menu titled 'Filter by branch' is open, showing options: 'main', 'gwstudent-patch-1', and 'main'. The 'main' option is selected.

The results show two workflow runs:

- Merge pull request #1 from gwstudent/gwstudent-patch-1** (main, 30s ago)
- Update pipeline.yml** (gwstudent-patch-1, 30s ago)

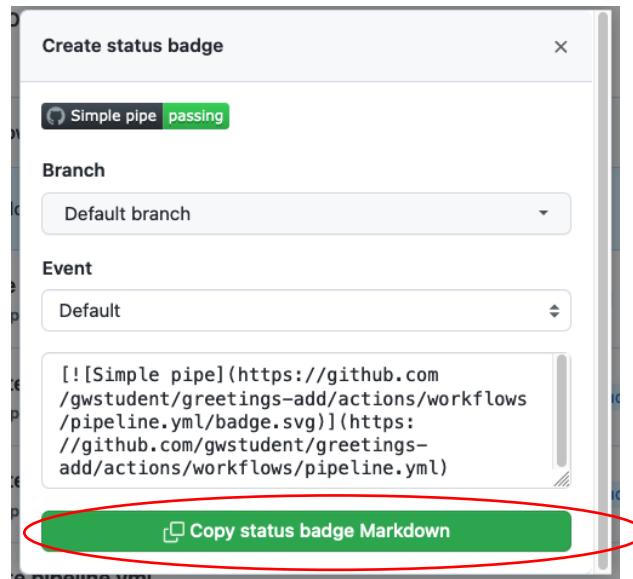
- Make sure you are on the branch *main*. (Switch back if you need to.) Select the "Simple Pipe" workflow. You'll now have a box with "..." beside the search box that has some additional options. Click on the "..." beside the search box to see some of them. They include disabling the workflow and setting 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".

The screenshot shows the GitHub Actions interface for the 'Simple pipe' workflow. The search bar is set to 'Filter workflow runs'. A red circle highlights the 'Create status badge' button in the dropdown menu.

The results show one workflow run:

- Merge pull request #1 from gwstudent/gwstudent-patch-1** (main, 17 minutes ago)

- 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/workflows/pipeline.yml)
```

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

README.md

greetings-add

Simple hello world type of program for use with learning GitHub Actions [Simple pipe](#) [passing](#)

- 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.

- 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.

- 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.

- 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.

```

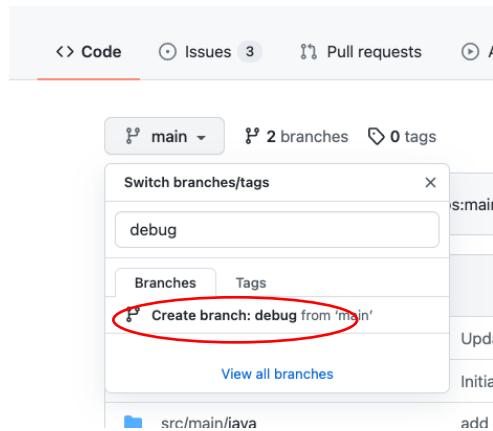
2023-01-31T02:14:14.4216423Z Requested labels: ubuntu-latest
2023-01-31T02:14:14.4216457Z Job defined at: gwstudent/greetings-add/.github/workflows/pipeline.yml@refs/heads/main
2023-01-31T02:14:14.4216482Z Waiting for a runner to pick up this job...
2023-01-31T02:14:14.5892733Z Job is waiting for a hosted 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 simple-pipe.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](#)

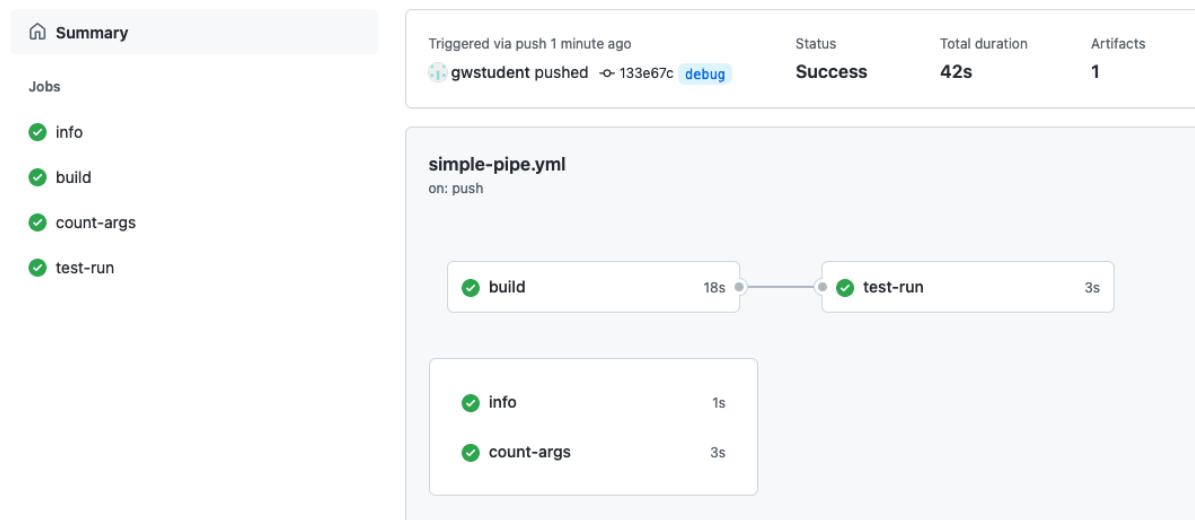
```

3
4
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."

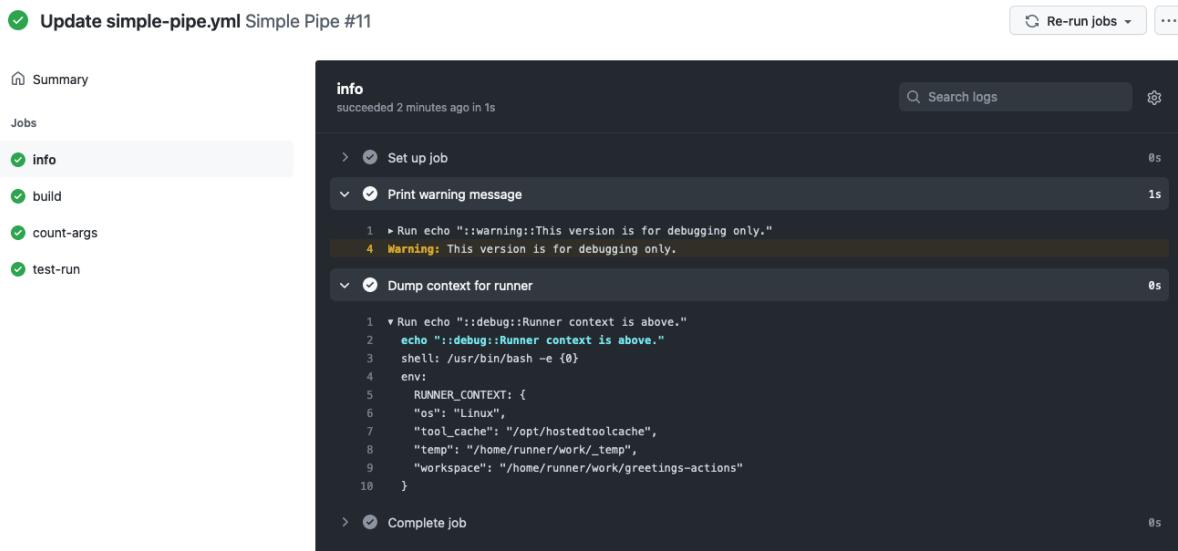
```

3. 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



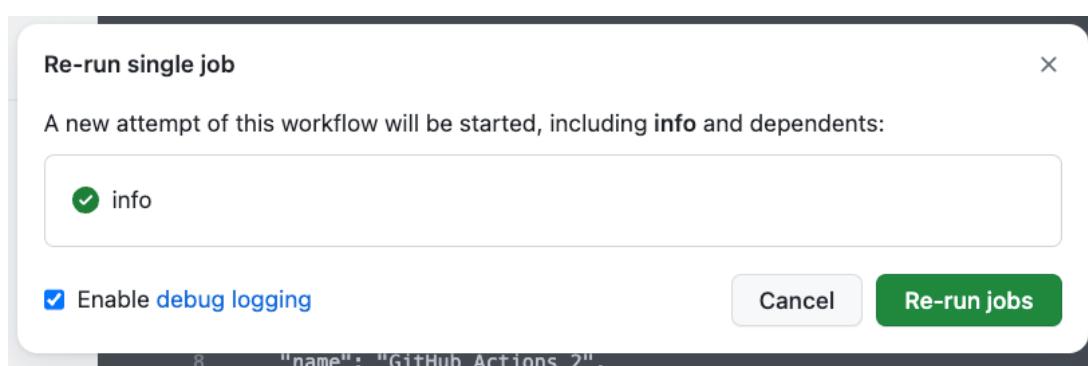
4. Expand the entries for "Print warning message" and "Dump context for runner" to see the outputs for those.



5. 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.
6. 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. On the left, there's a sidebar with 'Summary', 'Jobs', and other options like 'Run details', 'Usage', and 'Workflow file'. The 'Jobs' section has a list of four jobs: 'info' (selected), 'build', 'count-args', and 'test-run'. To the right is a log viewer with a search bar labeled 'Search logs' and a refresh/cog icon.

7. 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.

The screenshot shows the GitHub Actions log for the 'info' job. The job status is 'succeeded 3 minutes ago in 1s'. The log shows the execution of two steps: 'Print warning message' and 'Dump context for runner'. The 'Dump context for runner' step is expanded, showing the following 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",
11 ##[debug]  "url": "https://github.com/actions/toolkit@v2"
12 ##[debug]`
```

- 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.
- To do this, go to the repository's top menu and select "Settings". Then on the left-hand side, select "Secrets and variables", and then "Actions" under that. Select the "Variables" tab and "New repository variable". Underneath "Secrets", click on "Actions". Now, click on the "New repository secret" in the upper right to create a new secret for the action to use.

The screenshot shows the GitHub repository settings page for managing secrets and variables. The top navigation bar includes links for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings (which is circled with a red number 1). The left sidebar has sections for General, Access, Collaborators, Moderation options, Code and automation (with Branches, Tags, Actions, Webhooks, Environments, Codespaces, and Pages), and Security (with Code security and analysis, Deploy keys, and Secrets and variables, which is circled with a red number 2). The main content area is titled "Actions secrets and variables". It contains a section for Environment variables (Manage environments) and another for Repository variables (Manage environments). A red circle highlights the "Variables" tab (circled with a red number 3) and the "New repository variable" button (circled with a red number 4).

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

Actions variables / New variable

Note: Variable values are exposed as plain text. If you need to encrypt and mask sensitive information, [create a secret](#) instead.

Name *

ACTIONS_STEP_DEBUG

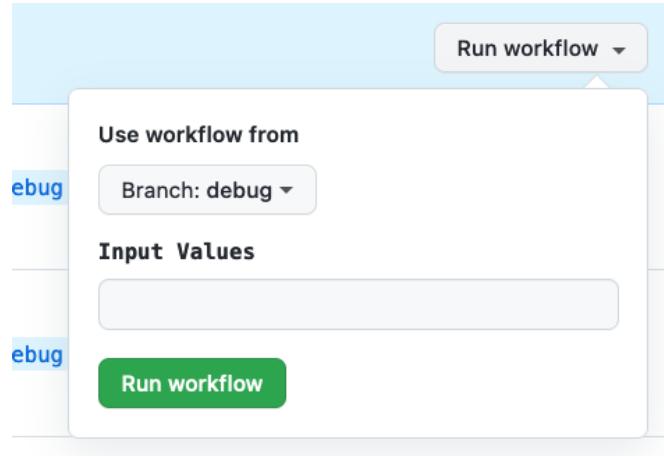
- Alphanumeric characters ([a-z], [A-Z], [0-9]) or underscores (_) only.
- Spaces are not allowed.
- Cannot start with a number.
- Cannot start with GITHUB_ prefix.

Value *

true

Add variable

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.

```

info
succeeded 32 seconds ago in 1s
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.
30 ##[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".

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.

4. 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:
  ARG$: ${{ github.event.inputs.myValues }}
run: |
  chmod +x ./test-script.sh
  ./test-script.sh ${{ needs.build.outputs.artifact-path }} "$ARG$"
```

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.

Actions New workflow

All workflows

Simple Pipe

Management Caches

Simple Pipe

Simple Pipe pipeline.yml

16 workflow runs Event Status Branch Actor

This workflow has a `workflow_dispatch` event trigger. Run workflow

Update pipeline.yml	main	Use workflow from
Simple Pipe #16: Commit 047f1a5 pushed by gwstudent2		Branch: main
Simple Pipe		Input Version
Simple Pipe #15: Manually run by gwstudent2	1.0.2-	
Simple Pipe		Input Values
Simple Pipe #14: Manually run by gwstudent2	'ls -la'	
Update pipeline.yml	dahun	Run workflow

24 minutes ago ...

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

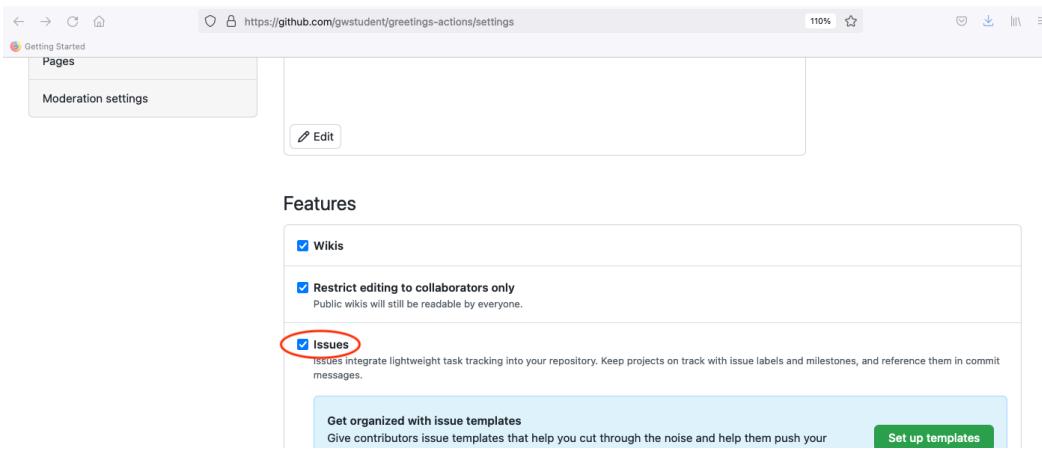
END OF LAB

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

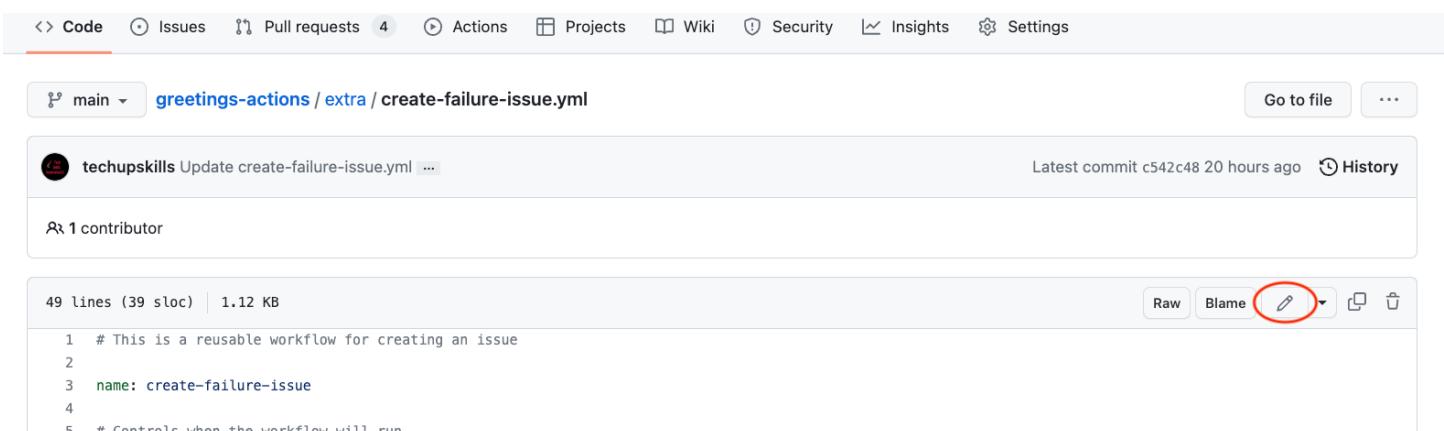
© 2023 Tech Skills Transformations, LLC & Brent Laster

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.



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 ...

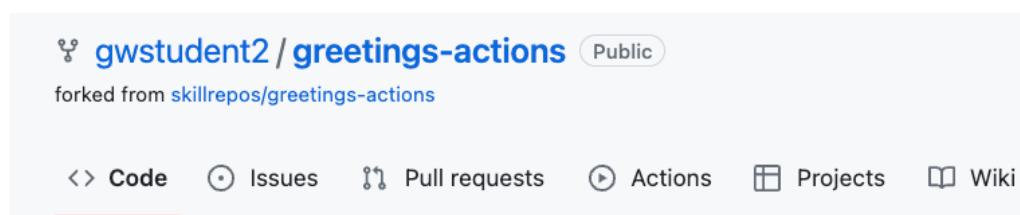
techupskills Update create-failure-issue.yml ... Latest commit c542c48 20 hours ago History

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

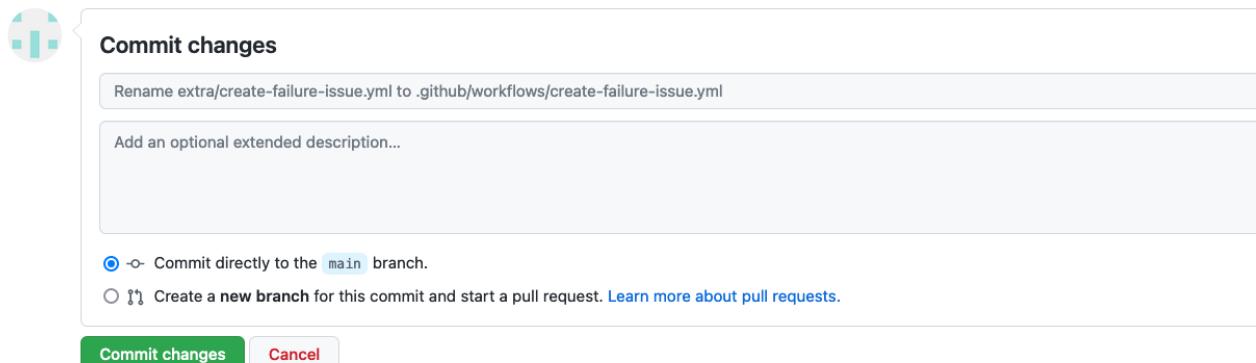
Code Issues Pull requests Actions Projects Wiki

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.



Commit changes

Rename extra/create-failure-issue.yml to .github/workflows/create-failure-issue.yml

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 Cancel

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 'New workflow' buttons, and a list of 'All workflows' including 'Simple Pipe' and 'create-failure-issue'. The 'create-failure-issue' workflow is selected and highlighted in blue. The main area shows a single workflow run for 'create-failure-issue'. It displays the message: 'This workflow has a workflow_dispatch event trigger.' Below this, it shows a successful run labeled 'create-failure-issue #1: Manually run by gwstudent'. To the right, a modal window is open titled 'Run workflow'. It contains fields for 'Issue title *' (set to 'This is a title') and 'Issue body *' (set to 'This is the body text'). A 'Run workflow' button is at the bottom of the modal.

- 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. At the top, there are tabs for 'Code', 'Issues' (which is selected), 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. Below the tabs, there's a search bar with 'is:issue is:open' and buttons for 'Labels' (9) and 'Milestones' (0). A 'New issue' button is also visible. The main area lists issues: one open issue titled 'FAILURE: This is a title' with the note '#2 opened now by github-actions (bot)'. There are filters at the bottom for '1 Open' and '0 Closed'.

- 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 simple-pipe.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:

```
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

```

<> Edit file ⌂ Preview changes Spaces 2 No wrap ↻
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

```

- 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 Show runs from all workflows

All workflows Filter workflow runs

Workflow	Event	Status	Branch	Actor
Simple Pipe	15 seconds ago	Success	main	...
create-failure-issue	13s	Success	main	...
Update simple-pipe.yml	1 minute ago	Failure	main	...

- 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 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 2s"]; B --> C["✓ build 16s"]; C --> D["✓ create-issue-on-failure 1s"]
```

[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 13s Artifacts -

Jobs

- ✓ create_issue_on_failure

create-failure-issue.yml
on: workflow_dispatch

```
graph LR; A["✓ create_issue_on_failure 1s"]
```

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!