

Intro to GitHub Actions

Revision 1.0 – 10/21/23

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-actions> and fork that project into your own GitHub space. Click on the *Fork* button in the upper right. On the next screen, you can just accept the options and click on the green *Create fork* button.

The screenshot shows two screenshots of the GitHub interface. The top screenshot shows the repository page for 'skillrepos / greetings-actions'. The 'Fork' button in the top right corner is circled in red. The bottom screenshot shows the 'Create a new fork' dialog box. The 'Create fork' button at the bottom is circled in red.

Repository Page Screenshot:

- URL: https://github.com/skillrepos/greetings-actions
- Repository Name: skillrepos / greetings-actions
- Owner: skillrepos
- Branches: main (selected), 1 branch
- Tags: 0 tags
- Last Commit: Brent Laster update files for labs (c175a1c, 1 hour ago)
- Commits: 4 commits
- Actions: 4
- Projects: 0
- Wiki: 0
- Security: 0
- Insights: 0
- About: Simple hello world type of program for use with learning GitHub Actions
- Releases: No releases published

Create a new fork Dialog Screenshot:

- Owner: gwstudent2 (selected)
- Repository name: greetings-actions
- Description (optional): Simple hello world type of program for use with learning GitHub Actions
- Copy the main branch only: (checkbox is checked)
- Note: By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.
- Note: You are creating a fork in your personal account.
- Create fork button: Circled in red.

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

The screenshot shows the GitHub repository page for `gwstudent2/greetings-actions`. The *Actions* tab is highlighted with a red circle. Below the header, there are buttons for *Code*, *Pull requests*, *Actions*, *Projects*, *Wiki*, *Security*, *Insights*, and *Settings*. Under the *Actions* tab, it says "This branch is up to date with skillrepos/greetings-actions:main." There are buttons for *Go to file*, *Add file*, and *Code*. Below this, a list of commits is shown:

Author	Commit Message	Date
brentlaster	Rename .github/workflows/simple-pipe.yml to extra/ simple-pipe.yml	de3212d on Apr 6 7 commits
	extra	Rename .github/workflows/simple-pipe.yml to extra/ simple-pipe.yml
	gradle/wrapper	Initial commit
	src/main/java	update files for labs
	build.gradle	update files for labs
	gradlew	Initial commit
	gradlew.bat	Initial commit

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”.

The screenshot shows the GitHub Actions categories page. It has sections for "Automation" and "Browse all categories". The "Automation" section contains four cards: "Greetings", "Stale", "Manual workflow", and "Labeler". The "Browse all categories" section lists "Automation", "Continuous integration", "Deployment", and "Security". The "Continuous integration" link is circled in red.

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.

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

Found 52 workflows

- Android CI** By GitHub Actions Build an Android project with Gradle. [Configure](#) Java
- Java with Ant** By GitHub Actions Build and test a Java project with Apache Ant. [Configure](#) Java
- Clojure** By GitHub Actions Build and test a Clojure project with Leiningen. [Configure](#) Clojure
- Publish Java Package** [Configure](#) Java
- Java with Gradle** [Configure](#) Java
- Publish Java Package** [Configure](#) Java

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

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

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
- Pages

Found 4 workflows

- Android CI** By GitHub Actions Build an Android project with Gradle. [Configure](#) Java
- SLSA Generic generator** By Open Source Security Foundation (OpenSSF) Generate SLSA3 provenance for your existing release workflows. [Configure](#) Go
- Publish Java Package with Gradle** By GitHub Actions Build a Java Package using Gradle and publish to GitHub Packages. [Configure](#) Java
- Java with Gradle** By GitHub Actions Build and test a Java project using a Gradle wrapper script. [Configure](#) Java

- 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 simple-pipe.yaml. (You can just backspace over or delete the name and type the new name.)

Code Pull requests Actions Projects Wiki Security Insights Settings

greetings-actions/.github/workflows/gradle.yml in main

Edit Preview Code 55% faster with GitHub Copilot Spaces 2 No wrap

```
1 # This workflow uses actions that are not certified by GitHub.
2 # They are provided by a third-party and are governed by
```

TO

Code Pull requests Actions Projects Wiki Security Insights Settings

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

Edit Preview Code 55% faster with GitHub Copilot Spaces 2 No wrap

```
1 # This workflow uses actions that are not certified by GitHub.
2 # They are provided by a third-party and are governed by
```

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

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

9. 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. (If you happen to get a 404 error, you can ignore it.)

The screenshot shows the GitHub commit dialog box. It has fields for 'Commit message' (containing 'Create simple-pipe.yml') and 'Extended description'. At the bottom, there are two radio button options: one selected ('Commit directly to the main branch') and one unselected ('Create a new branch for this commit and start a pull request'). A 'Commit changes' button is at the bottom right.

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.

The screenshot shows the GitHub Actions interface. On the left, there's a sidebar with 'Actions' selected, showing 'All workflows' and a list of items: 'Java CI with Gradle', 'Management', and 'Caches'. The main area is titled 'All workflows' and shows '0 workflow runs'. A single workflow run is listed: 'Create simple-pipe.yml' (status: 'In progress', event: 'main', 11 seconds ago). A search bar at the top right says 'Filter workflow runs'.

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 for the run to get to the details for that run.

This screenshot shows the same GitHub Actions interface after the workflow has succeeded. The 'Create simple-pipe.yml' run is now listed with a green checkmark icon and the status 'Succeeded'. The commit message 'Simple Pipe #1: Commit abdac83 pushed by gwstudent2' is visible. The 'Event' dropdown shows 'main'. The commit was made 2 minutes ago and took 29s. The 'Status' dropdown shows 'Success'.

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.

This screenshot shows the detailed view of the 'Create simple-pipe.yml #1' job. The left sidebar has 'Summary' and 'Jobs' sections, with 'build' selected. The main area shows a graph of the build process. One step, 'run actions/checkout@v3', is expanded, showing its log output. The log shows the command being run, repository sync, Git version info, and temporary HOME variable overriding. A red circle highlights the 'run actions/checkout@v3' step in the graph and its expanded log entry.

Lab 2 – Learning more about Actions

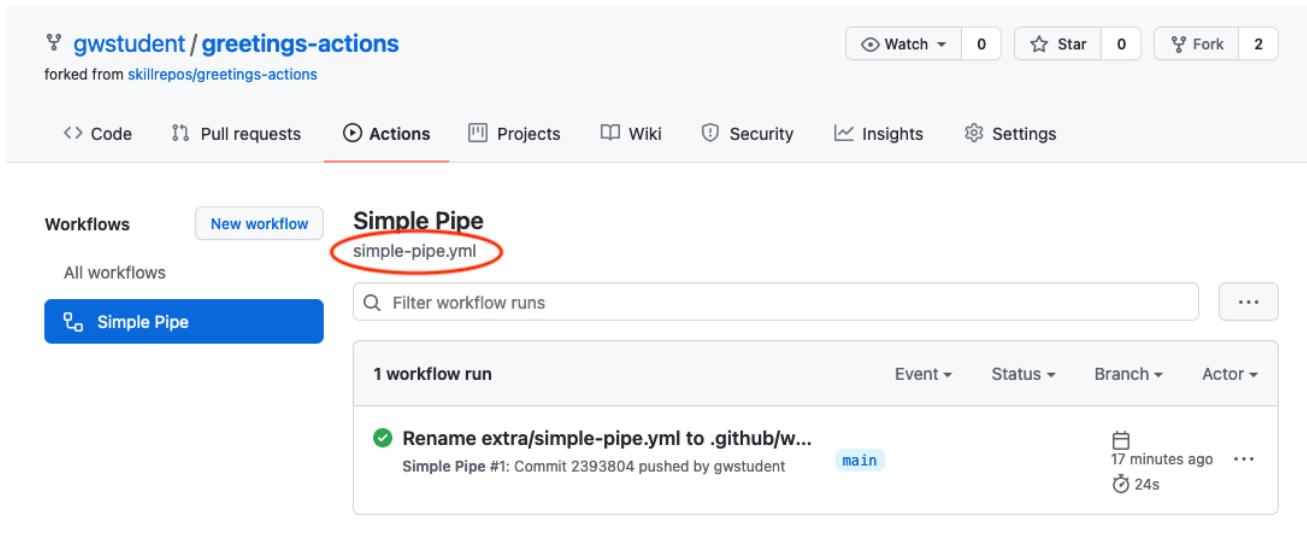
Purpose: In this lab, we'll see how to get more information about actions and how to update our workflow to use others.

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 simple-pipe.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.

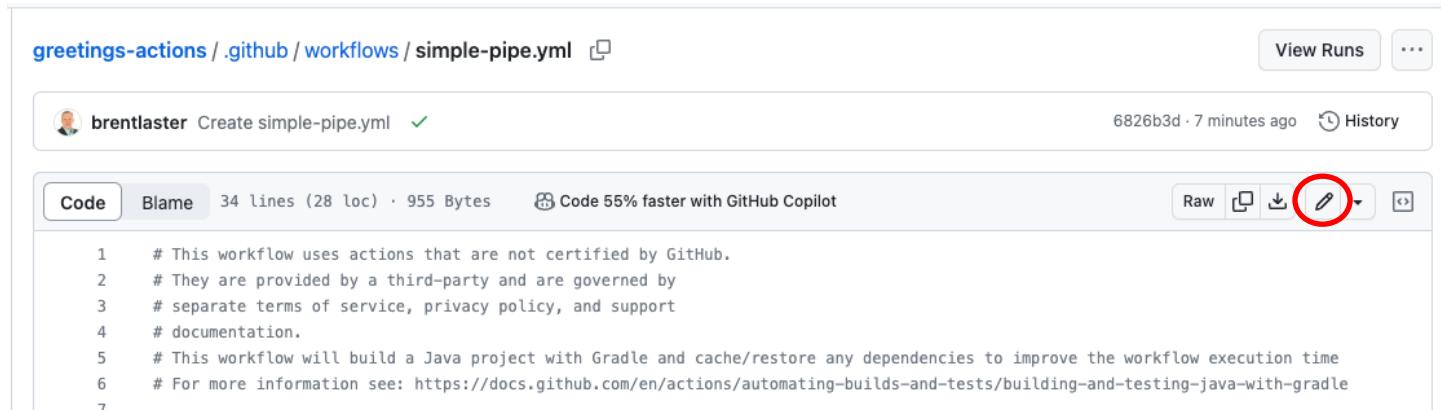
Under "All workflows", select the "Simple Pipe" workflow.

After that, select the "simple-pipe.yml" link near the middle top.



The screenshot shows the GitHub Actions interface for the repository 'gwstudent/greetings-actions'. The 'Actions' tab is selected. Under 'Workflows', the 'Simple Pipe' workflow is listed, with its file name 'simple-pipe.yml' circled in red. Below the workflow list is a search bar labeled 'Filter workflow runs' and a table showing '1 workflow run'. The first run is a successful 'Rename extra/simple-pipe.yml to .github/w...' commit, pushed by 'gwstudent' on the 'main' branch 17 minutes ago. A blue button labeled 'Simple Pipe' is visible on the left.

2. Once the file opens up, click on the pencil icon in the top right to edit it.



The screenshot shows the GitHub workflow editor for the file 'simple-pipe.yml'. The 'Code' tab is active. The code content is as follows:

```
1 # This workflow uses actions that are not certified by GitHub.
2 # They are provided by a third-party and are governed by
3 # separate terms of service, privacy policy, and support
4 # documentation.
5 # This workflow will build a Java project with Gradle and cache/restore any dependencies to improve the workflow execution time
6 # For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-gradle
7
```

To the right of the code area, there are several icons: 'Raw', 'Blame', 'Code 55% faster with GitHub Copilot', and a red-circled 'Edit' icon. Above the code area, a message from 'brentlaster' says 'Create simple-pipe.yml ✓'. To the right, it shows the commit details: '6826b3d · 7 minutes ago' and a 'History' link.

3. 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 artifact" 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".

The screenshot shows the GitHub Marketplace search results for "upload artifact". The search bar at the top contains "upload artifact". Below it, the results section is titled "Marketplace / Search results". It lists three actions:

- Upload GitHub Pages artifact** By actions (171) - A composite action that prepares your static assets to be deployed to GitHub Pages.
- Upload a Build Artifact** By actions (2.4k) - Upload a build artifact that can be used by subsequent workflow steps. This item is circled in red.
- S3 Upload Artifact** By blakjak44 (0) - Upload artifacts to S3.

On the right, a detailed view of the "Upload a Build Artifact" action is shown. The title is "Upload a Build Artifact" with a play button icon. It says "By actions" and has a version "v3.1.3" with 2.4k stars. The description is "Upload a build artifact that can be used by subsequent workflow steps". A red oval highlights the "View full Marketplace listing" button. Below it, the "Installation" section shows a snippet of YAML code:

```
- name: Upload a Build Artifact
  uses: actions/upload-artifact@v3.1.3
  with:
    # Artifact name
```

4. 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 listing for the "Checkout" action. The title is "GitHub Action Checkout" with a play button icon. It says "v4.0.0" and "Latest version". A green button says "Use latest version".

Below the title, there are two status indicators: "Build and Test" (passing) and "Verified creator".

The main content area is titled "Checkout V4". It includes a description of the action, usage instructions, and a "What's new" section.

On the right side, there are sections for "Stars" (4.4k), "Contributors" (a grid of user icons), and "Categories" (Utilities).

At the bottom, there is a "Links" section with three items: "actions/checkout" (highlighted with a red circle), "Open issues", and a page number "371".

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 top screenshot shows the GitHub Actions checkout repository page. The 'Actions' tab is highlighted with a red circle. The bottom screenshot shows the 'actions/checkout' page, specifically the 'All workflows' section. It displays two workflow runs:

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

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 in line with the build job steps. **Pay attention to the indenting.** 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
```

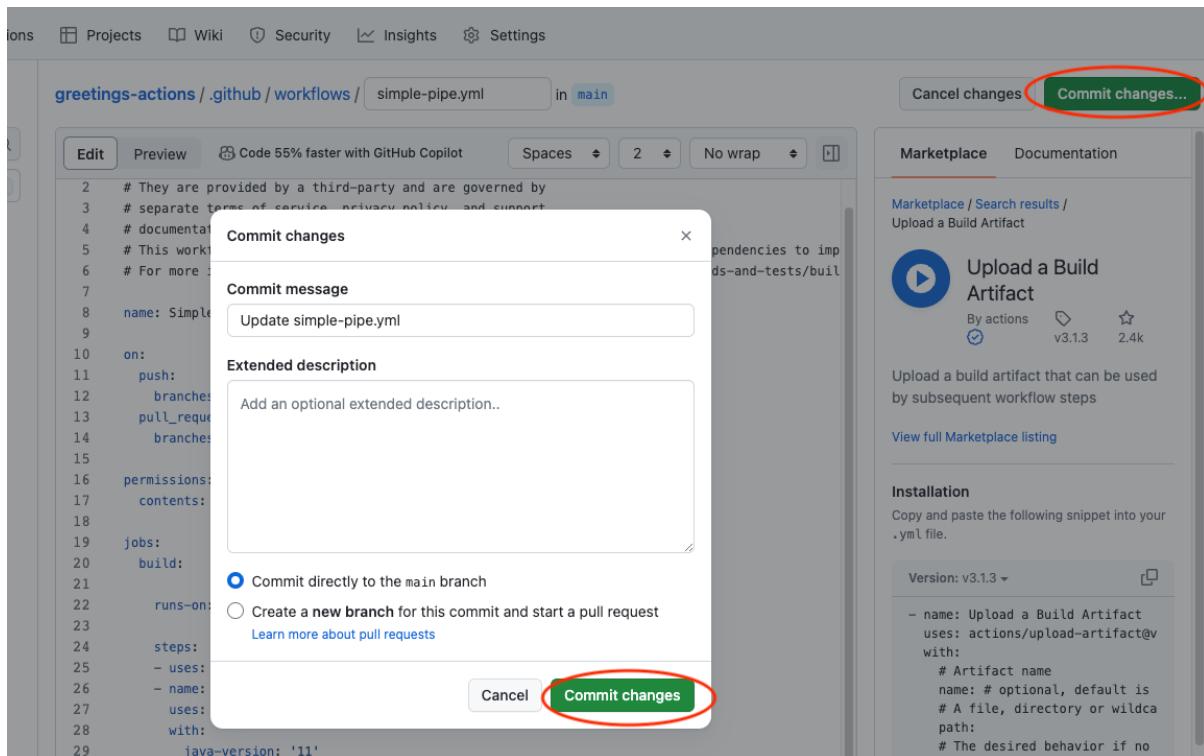
<> Edit file Preview changes

```

10  permissions:
11    contents: read
12
13  jobs:
14    build:
15
16      runs-on: ubuntu-latest
17
18      steps:
19        - uses: actions/checkout@v3
20        - name: Set up JDK 11
21          uses: actions/setup-java@v3
22          with:
23            java-version: '11'
24            distribution: 'temurin'
25        - name: Build with Gradle
26          uses: gradle/gradle-build-action@67421db6b
27          with:
28            arguments: build
29
30        - name: Upload Artifact
31          uses: actions/upload-artifact@v3
32          with:
33            name: greetings-jar
34            path: build/libs
35
36
37
38
39
40
41

```

- 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.



- 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 simple-pipe.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 GitHub Actions run summary for a workflow named 'simple-pipe.yml'. The run was triggered via push 2 minutes ago by user 'gwstudent' to branch 'main'. The status is 'Success' with a total duration of 26s and 1 artifact produced. The 'build' job is shown as completed with a green checkmark and a duration of 13s. In the 'Artifacts' section, a table lists a single artifact named 'greetings-jar' with a size of 1006 Bytes. The name 'greetings-jar' is circled in red.

- Now let's add a new job to our workflow (in simple-pipe.yml) to download this artifact and execute the jar file. The code is straightforward because there's already a "download_artifact" action for us to use. And we can just use a shell run command to execute "java -jar" on this. Add the code below into your workflow, indenting test-run to line up with the "build" job entry above it. See also the screenshot further down. (For convenience, this code is also in "extra/test-run.txt".) Again, pay attention to indentation.

test-run:

```
runs-on: ubuntu-latest
needs: build

steps:
- name: Download candidate artifacts
  uses: actions/download-artifact@v2
  with:
    name: greetings-jar
- shell: bash
  run: |
    java -jar greetings-actions.jar ${{ github.event.inputs.myValues }}
```

```

<> Edit file ⏎ Preview changes Spaces 2 No wrap
26   run: ./gradlew build
27   - name: Upload Artifact
28     uses: actions/upload-artifact@v2
29     with:
30       name: greetings-jar
31       path: build/libs
32
33 test-run:
34
35 runs-on: ubuntu-latest
36 needs: build
37
38 steps:
39   - name: Download candidate artifacts
40     uses: actions/download-artifact@v2
41     with:
42       name: greetings-jar
43     - shell: bash
44     run: |
45       java -jar greetings-actions.jar ${{ github.event.inputs.myValues }}

```

10. Click on the green "Commit changes..." button as before, commit the change, and then switch over the "Actions" tab. Click on the latest entry. You should see in the workflow graph two jobs now. (Note there is also a button to cancel the workflows.). Eventually, both should succeed.

Code Pull requests Actions Projects Wiki Security Insights Settings

Update simple-pipe.yml Simple Pipe #3

Cancel workflow ...

Summary	Triggered via push 17 seconds ago	Status	Total duration	Artifacts
Jobs	gwstudent pushed -o 4c12b4e main	In progress	-	-

simple-pipe.yml
on: push

```

graph LR
    build --> testRun
    subgraph build [build]
        build
        7s
    end
    subgraph testRun [test-run]
        testRun
    end

```

Lab 3: Adding your own action

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

1. 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. Again, you can just accept the defaults and click on the "Create fork" button.

The screenshot shows the GitHub repository page for `skillrepos/arg-count-action`. The top navigation bar includes 'Watch' (0), 'Star' (0), 'Fork' (1, circled in red), and 'Code' (selected), 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. A prominent blue banner at the top says 'Publish this Action to Marketplace'. The repository description is 'Simple GitHub Action demo'. Below the description, there's a 'Code' tab (selected), a 'main' branch dropdown, and buttons for 'Go to file', 'Add file', and 'Code'. The commit history shows three commits:

- techupskills Delete iterate-args.sh ... 1 hour ago 16
- action.yml Update action.yml 2 days ago
- count-args.sh Initial commit 3 days ago

On the right side, there are sections for 'About', 'Releases' (8 tags), 'Create a new release', and 'Packages' (No packages published).

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

Take a look at the `action.yml` file and see if you can understand what its doing. The syntax is mostly 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-actions` project and edit the `simple-pipe.yml` file). 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 2 spaces to align with the other job names). (For convenience, this code is also in "`greetings-actions/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-actions](#) / [.github](#) / [workflows](#) / simple-pipe.yml in [main](#)

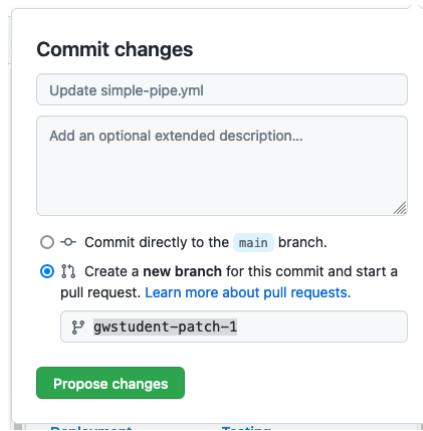
```

42      name: greetings-jar
43      - shell: bash
44      run: |
45        java -jar greetings-actions.jar ${{ github.event.inputs.myValues }}
46
47
48      count-args:
49
50        runs-on: ubuntu-latest
51
52        steps:
53          - id: report-count
54            uses: <your github userid>/arg-count-action@main
55            with:
56              arguments-to-count: ${{ github.event.inputs.myValues }}
57            - run: echo
58            - shell: bash
59            run: |
60              echo argument count is ${{ steps.report-count.outputs.arg-count }}
61

```

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".



© 2023 Tech Skills Transformations, LLC & Brent Laster

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

The change you just made was written to a new branch named `gwstudent-patch-1`. Create a pull request below to propose these changes.

Update simple-pipe.yml

Write Preview

Leave a comment

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

Create pull request

6. 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.

Update simple-pipe.yml #2
gwstudent wants to merge 1 commit into `main` from `gwstudent-patch-2`

Add more commits by pushing to the `gwstudent-patch-2` branch on `gwstudent/greetings-actions`.

Some checks were not successful
2 successful and 1 failing checks

- Simple Pipe / build (pull_request)** Successful in 14s [Details](#)
- Simple Pipe / count-args (pull_request)** Failing after 33s — count-args [Details](#) (circled)
- Simple Pipe / test-run (pull_request)** Successful in 5s [Details](#)

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request [Details](#) You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

✖ Update simple-pipe.yml Simple Pipe #4

The screenshot shows a GitHub Actions pipeline named "Simple Pipe #4". On the left, there's a sidebar with a "Summary" link circled in red. Below it, under "Jobs", there are three entries: "build" (green checkmark), "count-args" (red X), and "test-run" (green checkmark). The main area displays the "count-args" job details. It includes a "Set up job" step log and an error message at the bottom: "Error: Unable to resolve action `<your github userid>/arg-count-action@main` , repository not found".

- 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.

This screenshot shows the main GitHub Actions pipeline page for "Simple Pipe #5". The top navigation bar includes "Re-run jobs" and a three-dot menu. The main area features a graph of the workflow steps: "build" (green checkmark) followed by a 14-second duration, then "test-run" (green checkmark) followed by a 5-second duration. Below this, the "count-args" step (red X) is shown with a 33-second duration. To the right of the graph are controls for zooming in and out. Below the graph are sections for "Annotations" (1 error) and "Artifacts" (produced during runtime). The artifact "greetings-jar" is listed with a size of 1006 Bytes and a delete icon.

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 **simple-pipe.yaml** file (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.)

The screenshot shows the GitHub interface with the "Code" tab selected. The repository path is "greetings-actions / .github / workflows / simple-pipe.yaml". A dropdown menu shows the current branch is "brentlaster-patch-2". The code editor displays the YAML configuration for the workflow:

```

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 de
6  # For more information see: https://docs.github.com/en/actions/automating-buil
7
8  name: Simple Pipe
9
10 on:
11   push:
12     branches: [ "main" ]

```

9. Edit the simple-pipe.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.

```

47   count-args:
48
49     runs-on: ubuntu-latest
50
51     steps:
52       - id: report-count
53         uses: gwstudent/arg-count-action@main
54           with:
55             arguments-to-count: ${{ github.event.inputs.myValues }}
56       - run: echo
57       - shell: bash
58         run: |
59           echo argument count is ${{ steps.report-count.outputs.arg-count }}
60

```

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 in to the same patch branch as before. Then select "Commit changes".

The screenshot shows the GitHub interface for editing a workflow file. The left sidebar shows the repository structure with 'simple-pipe.yml' selected. The main area displays the YAML code for the workflow. A red circle highlights the 'Commit changes...' button in the top right corner of the editor. Another red circle highlights the 'uses' line in the code where 'brentlaster/arg-count-action@main' is specified.

```

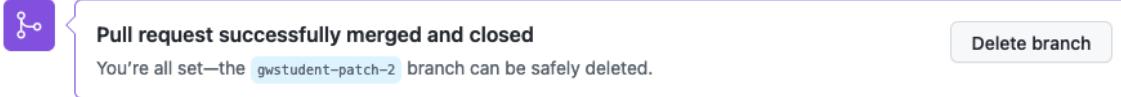
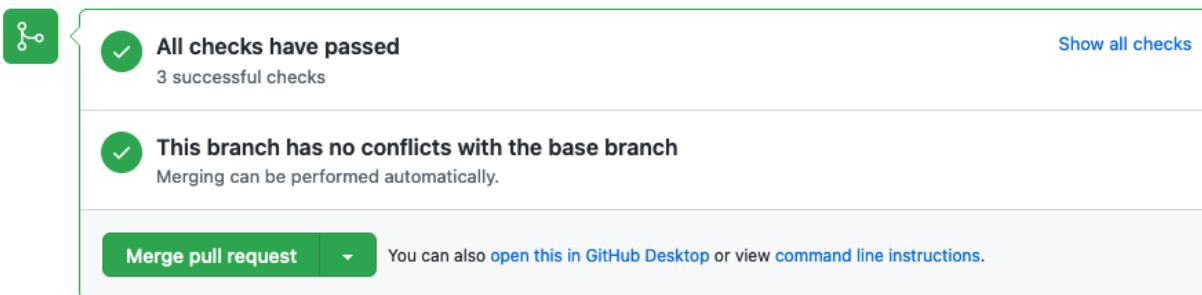
50       with:
51         name: greetings-jar
52       - shell: bash
53         run: |
54           java -jar greetings-actions.jar ${github.event.inputs.myValues}
55
56       count-args:
57
58       runs-on: ubuntu-latest
59
60       steps:
61       - id: report-count
62         uses: brenlaster/arg-count-action@main
63         with:
64           arguments-to-count: ${{ github.event.inputs.myValues }}

```

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-2` branch on `gwstudent/greetings-actions`.



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

All workflows

New workflow

All workflows

Simple Pipe

Filter workflow runs

7 workflow runs

Merge pull request #2 from gwstudent/gwstudent... main

Simple Pipe #7: Commit 274fa3a pushed by gwstudent

43 seconds ago ...
40s

Lab 4: 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 log 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.

Code Pull requests Actions Projects Wiki Security Insights Settings

All workflows

branch:patch-1

2 workflow run results

Update simple-pipe.yml patch-1 3 minutes ago ...
Simple Pipe #6: Pull request #1 synchronize by gwstudent2

Update simple-pipe.yml patch-1 16 minutes ago ...
Simple Pipe #5: Pull request #1 opened by gwstudent2

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

All workflows

Showing runs from all workflows

Filter workflow runs

7 workflow runs

Merge pull request #2 from gwstudent/gwstudent... main

Simple Pipe #7: Commit 274fa3a pushed by gwstudent

Update simple-pipe.yml gwstudent 16 minutes ago ...
Simple Pipe #6: Pull request #2 synchronize by gwstudent

Filter by branch

Find a branch

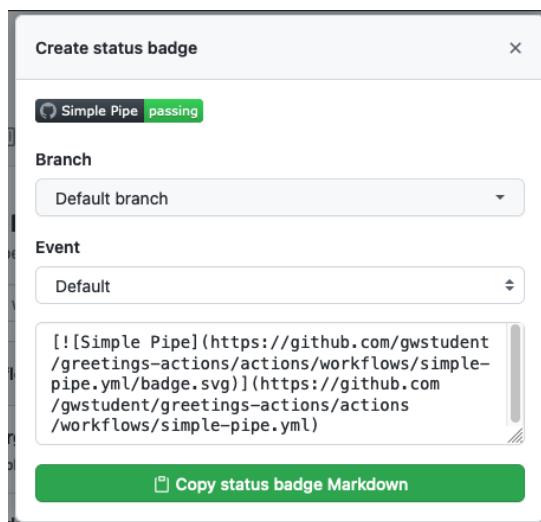
gwstudent-patch-1

gwstudent-patch-2

3. 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. On the left, there's a sidebar with 'Actions' (selected), 'New workflow', 'All workflows', 'Simple Pipe' (selected), and 'Management'. The main area shows the 'Simple Pipe' workflow with its file name 'simple-pipe.yml'. It displays 7 workflow runs, one of which is a successful merge pull request from 'gwstudent2/patch-1' to 'main'. The status bar indicates it was pushed 4 minutes ago. At the top right of the workflow card, there's a 'Create status badge' button, which is circled in red.

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



5. Click on the "<> Code" tab at the top left 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.

The screenshot shows the GitHub repository interface for the 'greetings-actions' repository. It's currently viewing the 'README.md' file. At the top, there are tabs for 'Code' (selected), 'Compare', 'Pull requests', and 'Issues'. Below the tabs, there's a message encouraging users to add a README. On the right, there's a green 'Add a README' button. The main area shows the content of the README.md file, which contains a single line of code:

```
1 # greetings-actions
```

. At the bottom of the file list, there's a green 'Add a README' button, which is circled in red.

6. Commit your changes. Then you should see the badge showing up as part of the README.md content.

A screenshot of the GitHub README.md editor. The file path is 'main / greetings-actions / README.md'. The content of the file is:

```
greettings-actions
Simple hello world type of program for use with learning GitHub Actions
```

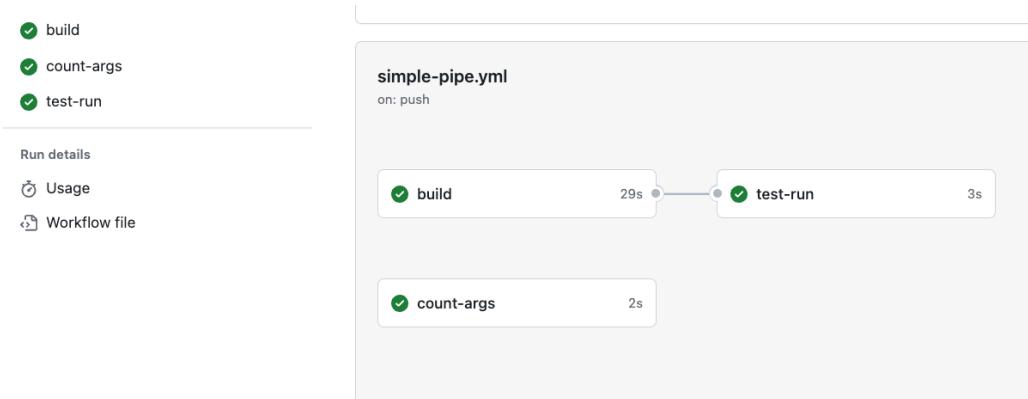
At the bottom of the editor, there is a green badge with a checkmark icon and the text 'Simple Pipe passing'.

7. 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.

A screenshot of the GitHub Actions tab. The workflow run is titled 'Create README.md #8'. The summary bar shows:

- Triggered via push 3 minutes ago
- gwstudent2 pushed ~ 8c7f16b main
- Status: Success
- Total duration: 48s
- Artifacts: 1

8. 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 happened.



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 "Run java..."** 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 a GitHub Actions pipeline named 'Create README.md Simple Pipe #8'. The 'test-run' job has three steps: 'Set up job', 'Download candidate artifacts', and 'Run java -jar greetings-actions.jar'. The third step is expanded, showing log entries from September 6, 2021, at 17:06:05 GMT. The first entry, which starts with '1 Mon, 06 Sep 2021 17:06:05 GMT', has its arrowhead circled in red. The last entry, '5 Mon, 06 Sep 2021 17:06:06 GMT No arguments were passed in.', is also circled in red. A gear icon in the top right corner of the log viewer is also circled in red.

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

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

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

The screenshot shows a browser window displaying the raw log output for the workflow. The URL is https://pipelines.actions.githubusercontent.com/SV7ZIYPW2ZEh3tn9Plq8HPOymvfPoeUAYn. The log output is as follows:

```
2021-09-04T18:24:07.7246019Z Found online and idle hosted runner in the current repository's organization account that matches the required labels: 'ubuntu-latest'
2021-09-04T18:24:07.7843970Z Waiting for a Hosted runner in the 'organization' to pick this job...
2021-09-04T18:24:08.0283205Z Job is waiting for a hosted runner to come online.
2021-09-04T18:24:10.7866800Z Job is about to start running on the hosted runner: Hosted Agent (hosted)
2021-09-04T18:24:12.5682931Z Current runner version: '2.280.3'
2021-09-04T18:24:12.5709160Z ##[group]Operating System
2021-09-04T18:24:12.5710048Z Ubuntu
2021-09-04T18:24:12.5710438Z 20.04.3
2021-09-04T18:24:12.5710860Z LTS
2021-09-04T18:24:12.5711302Z ##[endgroup]
2021-09-04T18:24:12.5712059Z ##[group]Virtual Environment
2021-09-04T18:24:12.5712713Z Environment: ubuntu-20.04
2021-09-04T18:24:12.5713332Z Version: 20210831.9
2021-09-04T18:24:12.5714366Z Included Software: https://github.com/actions/virtual-environments/blob/ubuntu20/20210831.9/images/linux/Ubuntu2004-README.md
2021-09-04T18:24:12.5715874Z Image Release: https://github.com/actions/virtual-environments/releases/tag/ubuntu20%2F20210831.9
2021-09-04T18:24:12.5716827Z ##[endgroup]
2021-09-04T18:24:12.5717449Z ##[group]Virtual Environment Provisioner
2021-09-04T18:24:12.5718158Z 1.0.0.0-master-20210816-1
2021-09-04T18:24:12.5718690Z ##[endgroup]
2021-09-04T18:24:12.5720787Z ##[group]GITHUB_TOKEN Permissions
2021-09-04T18:24:12.5722135Z Actions: write
2021-09-04T18:24:12.5722726Z Checks: write
2021-09-04T18:24:12.5723289Z Contents: write
2021-09-04T18:24:12.5723900Z Deployments: write
```

12. Let's make one more change to make it easier to run our workflow manually to try things out, start runs, etc. Edit the simple-pipe.yaml file as before. With the yaml file for the workflow open, add the

code below at the bottom of the "on" section. ("workflow_dispatch" should line up with "pull_request" and "push")

workflow_dispatch:

```
inputs:  
  myValues:  
    description: 'Input Values'
```

The screenshot shows the GitHub Actions workflow editor. The workflow file is named simple-pipe.yml and is currently in the main branch. The code editor shows the following YAML configuration:

```
9  
10  on:  
11    push:  
12      branches: [ "main" ]  
13    pull_request:  
14      branches: [ "main" ]  
15    workflow_dispatch:  
16      inputs:  
17        myValues:  
18          description: 'Input Values'  
19  
20  
21    permissions:  
22      contents: read  
23
```

The line `workflow_dispatch:` is highlighted with a blue selection bar. To the right of the editor, there is a sidebar titled "Marketplace" which lists "Featured Actions". The first action listed is "Setup Node.js environment" by actions, and the second is "Upload a Build Artifact" by actions.

13. Commit your changes to the main branch. Since we added the additional "on" section, if you look at the Actions page and select the workflow and select the "Simple Pipe" workflow, you'll see that after the merge we now have a note that "This workflow has a workflow_dispatch event trigger." And there is a "Run workflow" button that we can use to run the workflow manually. Click on that button, enter some data and then click "Run workflow" to see this in action. After a few moments, you should see another run of the workflow startup and complete.

The screenshot shows the GitHub Actions page for the "Simple Pipe" workflow. The workflow has 9 runs. A tooltip indicates that the workflow has a workflow_dispatch event trigger. The most recent runs are:

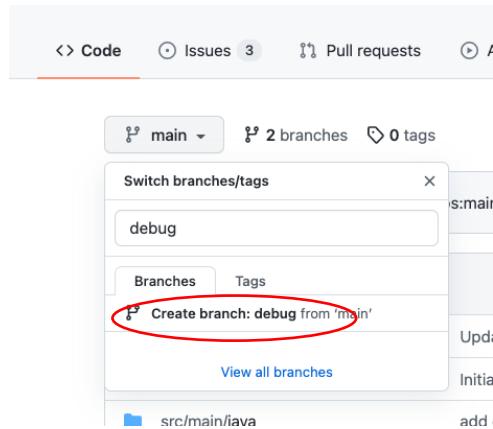
- Update simple-pipe.yml (main) - Triggered by a push from gwstudent
- Create README.md (main) - Triggered by a push from gwstudent
- Merge pull request #2 from gwstudent/gwstudent-p...

A modal window is open for the latest run, showing the "Use workflow from" dropdown set to "Branch: main", the "Input Values" field containing "abc def ghi", and a "Run workflow" button.

Lab 5: 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 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.)

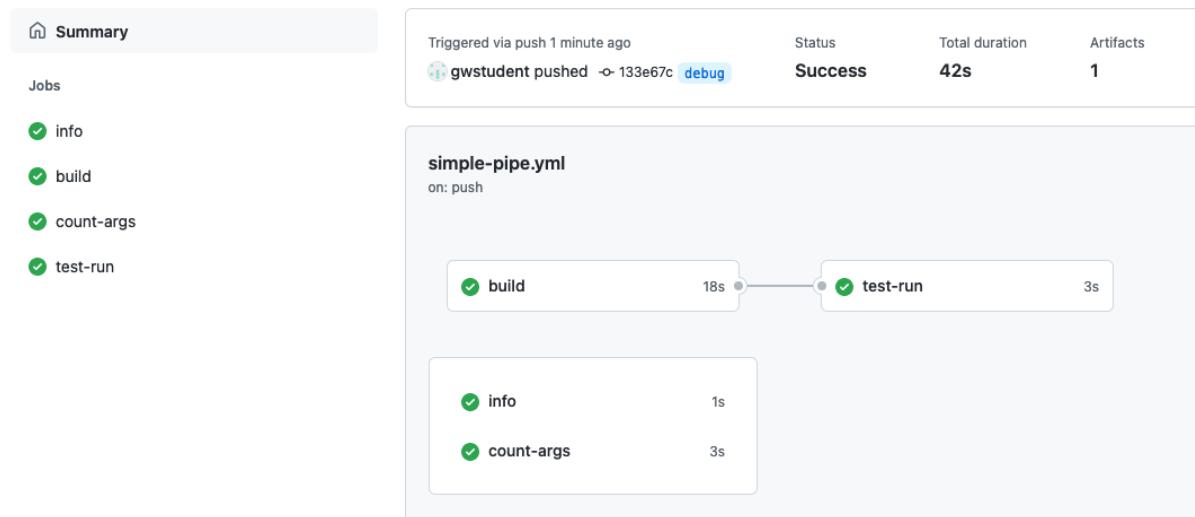
```
info:  
  runs-on: ubuntu-latest  
  
steps:  
  - name: Print warning message  
    run: |  
      echo "::warning::This version is for debugging only."  
  - name: Dump context for runner  
    env:  
      RUNNER_CONTEXT: ${{ toJSON(runner) }}  
    run:  
      echo "::debug::Runner context is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

<> Edit file <> Preview changes

```
1
2
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."
```

- 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.

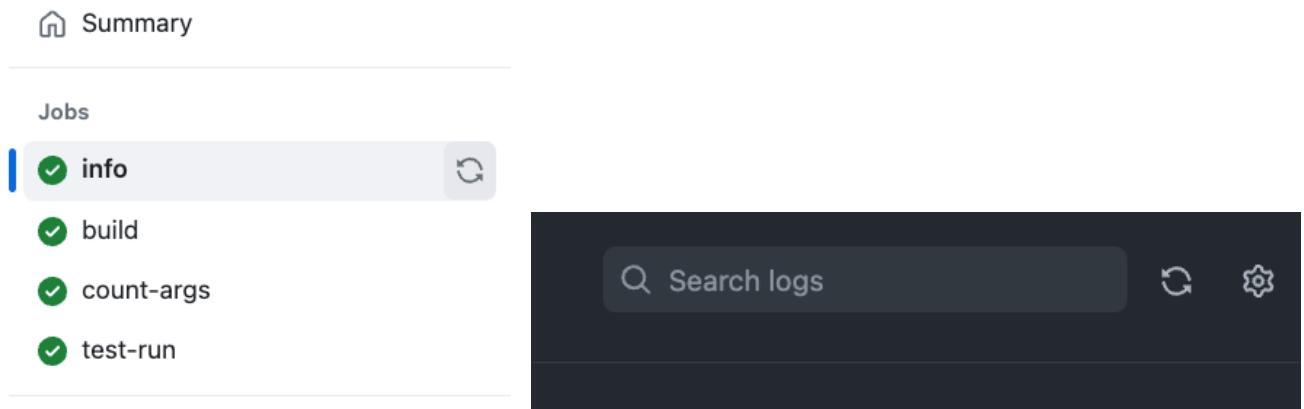
The screenshot shows the GitHub Actions logs for a workflow named "Simple Pipe #11". The "info" job has completed successfully. The log output is as follows:

```

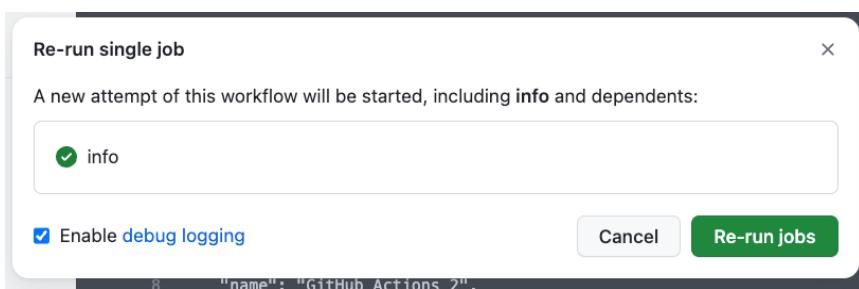
info
succeeded 2 minutes ago in 1s
Search logs
0s
> Set up job
0s
< Print warning message
1 ▶ Run echo ::warning::This version is for debugging only.
4 Warning: This version is for debugging only.
0s
< Dump context for runner
0s
1 ▶ Run echo ::debug::Runner context is above.
2 echo ::debug::Runner context is above.
3 shell: /usr/bin/bash -e {0}
4 env:
5   RUNNER_CONTEXT: {
6     "os": "Linux",
7     "tool_cache": "/opt/hostedtoolcache",
8     "temp": "/home/runner/work/_temp",
9     "workspace": "/home/runner/work/greetings-actions"
10 }
0s
< Complete job
0s

```

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



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

The screenshot shows a GitHub Actions run summary. On the left, a sidebar lists jobs: 'info' (green checkmark), 'build' (green checkmark), 'count-args' (green checkmark), and 'test-run' (green checkmark). The 'info' job is selected. The main area displays the 'info' job's logs. The status bar at the top right indicates the run is 'Paused'.

info
succeeded 3 minutes ago in 1s

Search logs

Jobs

info

build

count-args

test-run

Run details

Usage

Workflow file

info

> ✓ Print warning message 0s

✓ Dump context for runner 0s

```
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.

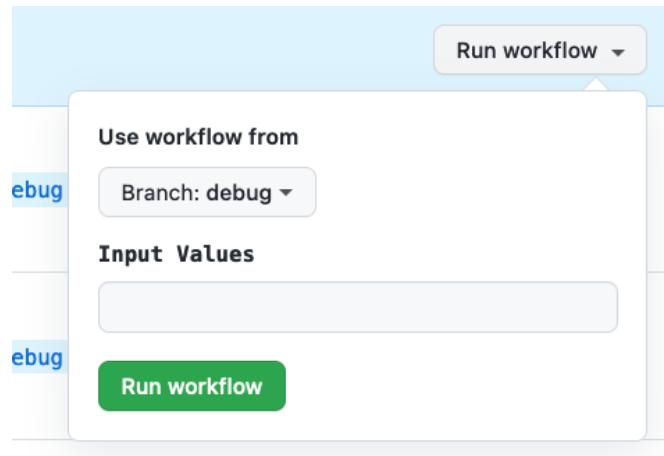
10. To do this, go to the repository's top menu and select "Settings". Then on the left-hand side, select "Secrets and variables", and then "Actions" under that. Select the "Variables" tab and "New repository variable".

The screenshot shows the GitHub repository settings page for managing actions secrets and variables. The top navigation bar includes links for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings (circled with a red box and labeled '1'). The left sidebar has sections for General, Access, Collaborators, and Moderation options; Code and automation (Branches, Tags, Actions, Webhooks, Environments, Codespaces, Pages); and Security (Code security and analysis, Deploy keys). A red circle labeled '2' highlights the 'Secrets and variables' link under the Actions section. The main content area is titled 'Actions secrets and variables'. It explains that secrets and variables manage reusable configuration data, with secrets being encrypted for sensitive data and variables being plain text for non-sensitive data. It notes that anyone with collaborator access can use these for actions, but they are not passed to workflows triggered by pull requests from forks. Below this, there are tabs for 'Secrets' (circled with a red box and labeled '3') and 'Variables' (circled with a red box and labeled '4'), with the 'Variables' tab selected. A green button labeled 'New repository variable' is visible. The 'Environment variables' section shows no variables for this repository's environments. The 'Repository variables' section also shows no variables for this repository.

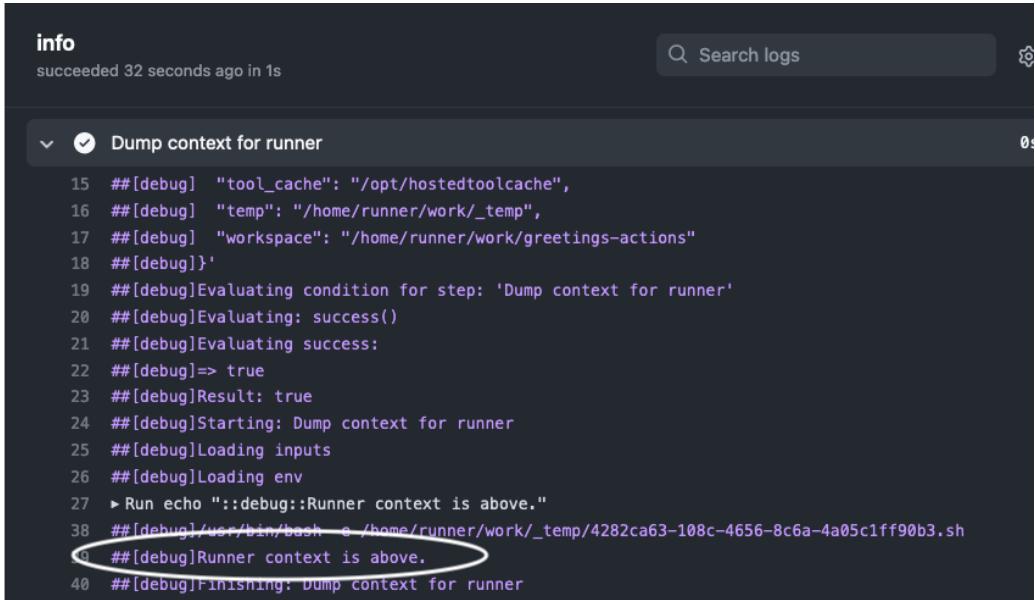
11. On the next screen, enter **ACTIONS_STEP_DEBUG** for the name, set the value to *true* and click on the *Add variable* button.

The screenshot shows the "Actions variables / New variable" page. On the left, there's a sidebar with various GitHub features like General, Access, Collaborators, and Moderation options. Under "Code and automation", there are branches, tags, actions, webhooks, environments, codespaces, and pages. Under "Security", there are code security analysis and deploy keys. Under "Secrets and variables", there are actions and codespaces. A red circle highlights the "Add variable" button at the bottom of the main form area. The form itself has fields for "Name *" (set to "ACTIONS_STEP_DEBUG") and "Value *" (set to "true"). A note at the top right says: "Note: Variable values are exposed as plain text. If you need to encrypt and mask sensitive information, [create a secret](#) instead."

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.



```

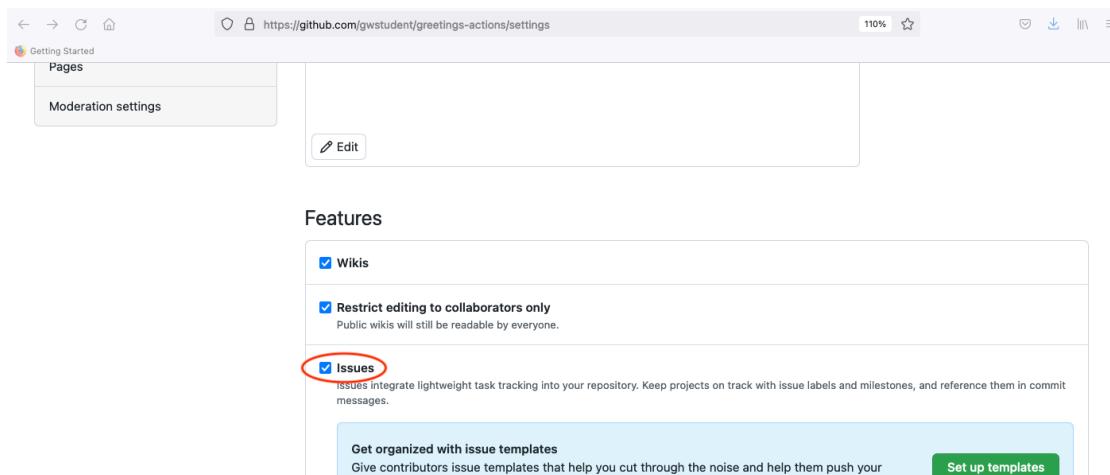
info
succeeded 32 seconds ago in 1s
Search logs
0:00:00.000000 15 ##[debug] "tool_cache": "/opt/hostedtoolcache",
0:00:00.000000 16 ##[debug] "temp": "/home/runner/work/_temp",
0:00:00.000000 17 ##[debug] "workspace": "/home/runner/work/greetings-actions"
0:00:00.000000 18 ##[debug]]'
0:00:00.000000 19 ##[debug]Evaluating condition for step: 'Dump context for runner'
0:00:00.000000 20 ##[debug]Evaluating: success()
0:00:00.000000 21 ##[debug]Evaluating success:
0:00:00.000000 22 ##[debug]=> true
0:00:00.000000 23 ##[debug]Result: true
0:00:00.000000 24 ##[debug]Starting: Dump context for runner
0:00:00.000000 25 ##[debug]Loading inputs
0:00:00.000000 26 ##[debug]Loading env
0:00:00.000000 27 ► Run echo "::debug::Runner context is above."
0:00:00.000000 38 ##[debug]/usr/bin/bash -e /home/runner/work/_temp/4282ca63-108c-4656-8c6a-4a05c1ff90b3.sh
0:00:00.000000 39 ##[debug]Runner context is above.
0:00:00.000000 40 ##[debug]Finishing: Dump context for runner

```

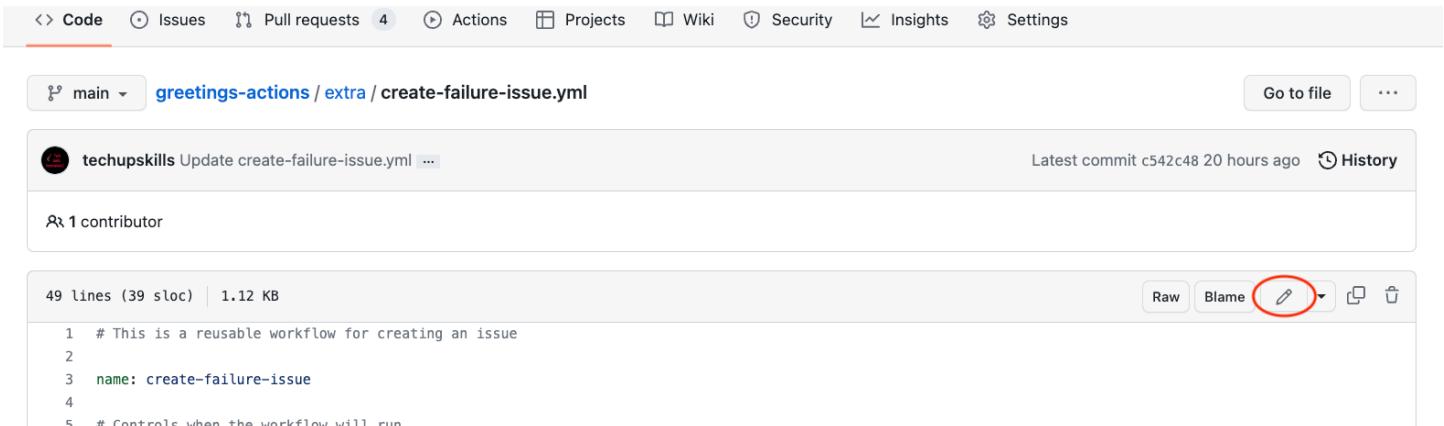
14. Note that the debug log info will be turned on for all runs from here on - as long as the repository variable exists and is set to "true".

Lab 6: Using reusable workflows

- 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.



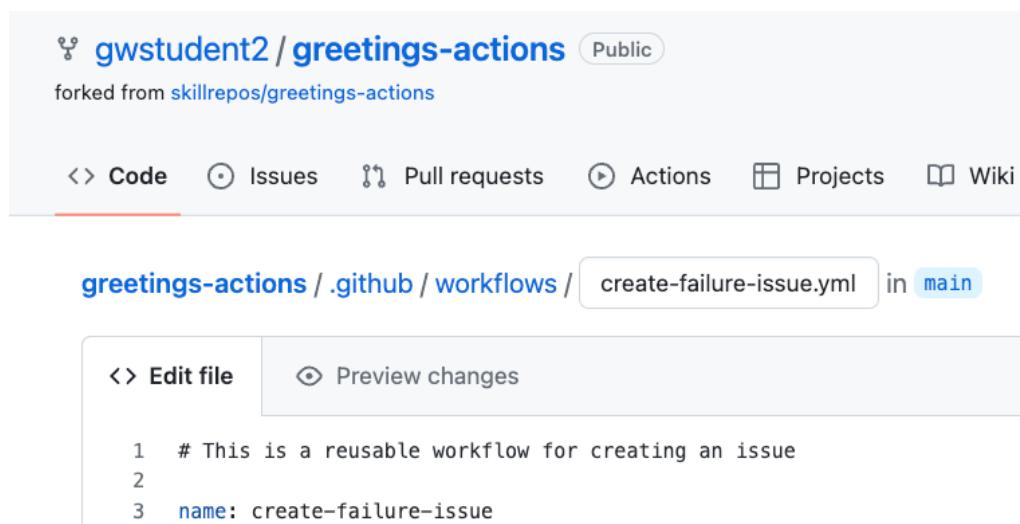
The screenshot shows a GitHub repository interface. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, Settings, and a dropdown for the main branch. Below the navigation is a breadcrumb trail: main > greetings-actions / extra / create-failure-issue.yml. A commit card for a user named 'techupskills' is displayed, showing an update to the file 'create-failure-issue.yml'. The commit message is 'Update create-failure-issue.yml ...'. The commit was made 20 hours ago with a SHA of c542c48. Below the commit card, it says '1 contributor'. The code editor shows the file content with 49 lines (39 sloc) and a size of 1.12 KB. At the bottom of the code editor, there are buttons for Raw, Blame, and a circled 'Edit' button.

```

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

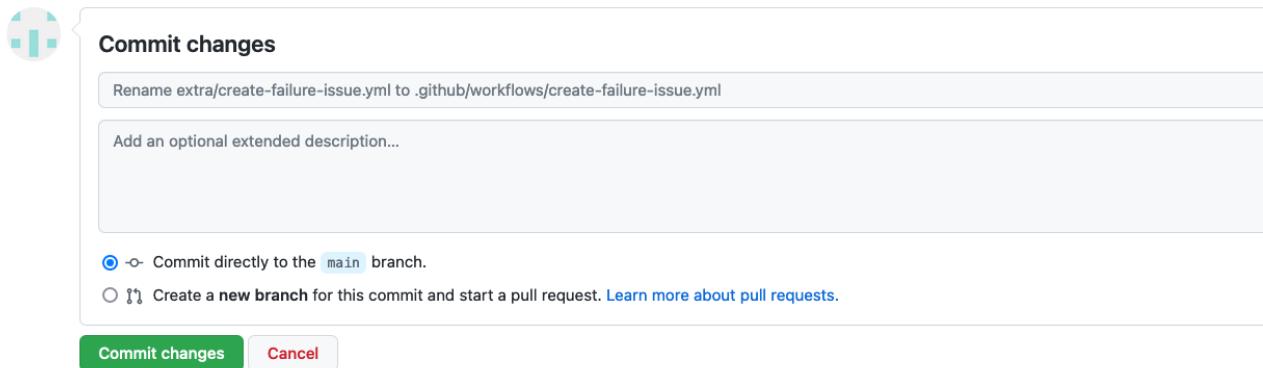
```

- 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".



The screenshot shows the same GitHub repository interface after the file has been moved. The breadcrumb trail now shows 'greetings-actions / .github / workflows / create-failure-issue.yml' in the main branch. The code editor shows the file content with 3 lines of code: '# This is a reusable workflow for creating an issue', 'name: create-failure-issue', and '# Controls when the workflow will run'.

- e. To complete the change, click on the green "Commit changes" button. (Ignore the 404 error if you get one.)



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".

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.

- 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:

```
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

```
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

Simple Pipe
create-failure-issue

All workflows

Showing runs from all workflows

Filter workflow runs

15 workflow runs

Event ▾ Status ▾ Branch ▾ Actor ▾

Workflow	Status	Triggered	Duration	Actions
create-failure-issue create-failure-issue #2: Manually run by gwstudent	Success	15 seconds ago	13s	...
Update simple-pipe.yml Simple Pipe #14: Commit 1539282 pushed by gwstudent	Failure	1 minute ago	49s	...

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

X 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 (green checkmark)
- count-args (red X)
- test-run (green checkmark)
- create-issue-on-failure (green checkmark)

simple-pipe.yml
on: push

```

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

Annotations
1 error

X 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 (green checkmark)

create-failure-issue.yml
on: workflow_dispatch

```

graph LR
    A:create_issue_on_failure -- "1s" --> B
    
```

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

The screenshot shows a GitHub repository's Issues page. The top navigation bar includes links for Code, Issues (2), Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main title of the issue is "Automated workflow failure issue for commit 153928267f2fc38a4e91b5bd00d61f033abd59d6 #4". Below the title, there is a green "Open" button and a timestamp indicating the issue was opened 3 minutes ago with 0 comments. A single comment from the "github-actions bot" is present, stating "This issue was automatically created by the GitHub Action workflow ** Simple Pipe **".

THE END - THANKS!