

Introduction to GitHub Actions

Revision 3.2 – 04/08/23

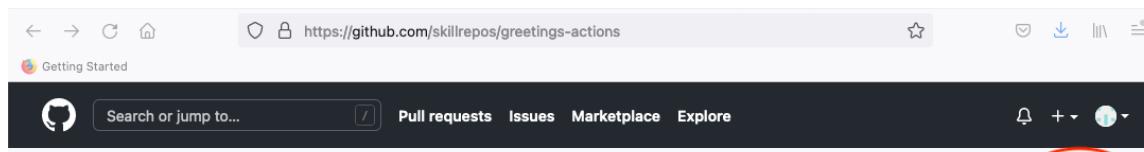
Tech Skills Transformations LLC / Brent Laster

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

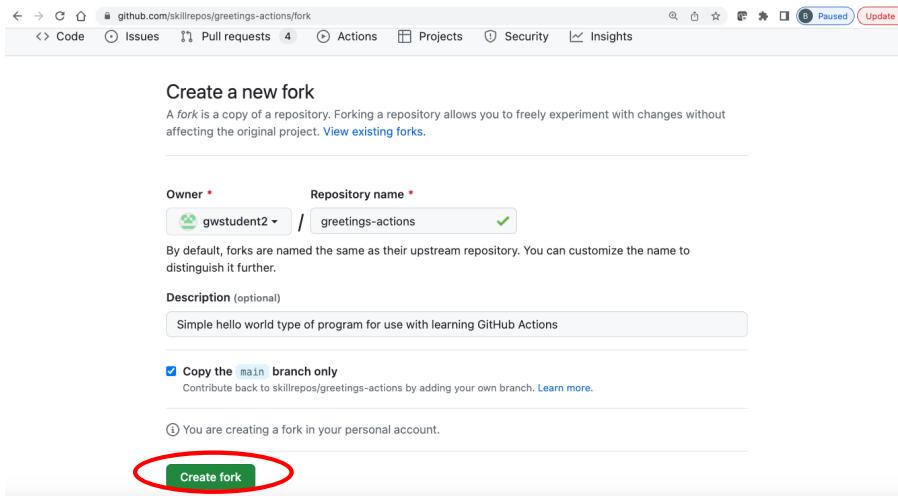
Lab 1 – Creating a simple example

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-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 the GitHub repository page for 'skillrepos/greetings-actions'. The 'Fork' button in the top right corner is circled in red. Below the header, there's a list of commits from 'Brent Laster' and a section for 'About' the repository.



The screenshot shows the 'Create a new fork' dialog box. It includes fields for 'Owner' (set to 'gwstudent2') and 'Repository name' (set to 'greetings-actions'). A note says 'By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.' There's an optional 'Description' field with the text 'Simple hello world type of program for use with learning GitHub Actions'. A checked checkbox says 'Copy the main branch only' with a note about contributing back. A note at the bottom says 'You are creating a fork in your personal account.' The 'Create fork' button at the bottom is circled in red.

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

The screenshot shows the GitHub repository page for `gwstudent2/greetings-actions`. The `Actions` tab is highlighted with a red circle. The repository has 1 branch and 0 tags. A message indicates the branch is up to date with the upstream. Below is a list of commits by `brentlaster`:

Commit	Message	Date
<code>extra</code>	Rename <code>.github/workflows/simple-pipe.yml</code> to <code>extra/simple-pipe.yml</code>	7 months ago
<code>gradle/wrapper</code>	Initial commit	14 months ago
<code>src/main/java</code>	update files for labs	14 months ago
<code>build.gradle</code>	update files for labs	14 months ago
<code>gradlew</code>	Initial commit	14 months ago
<code>gradlew.bat</code>	Initial commit	14 months ago

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 includes sections for Automation and a "Browse all categories" section. The "Continuous integration" category is circled in red.

- Automation**
 - Greetings: Greets users who are first time contributors to the repo. By GitHub Actions. Automation status: ●
 - Stale: Checks for stale issues and pull requests. By GitHub Actions. Automation status: ●
 - Manual workflow: Simple workflow that is manually triggered. By GitHub Actions. Automation status: ●
 - Labeler: Labels pull requests based on the files changed. By GitHub Actions. Automation status: ●
- Browse all categories**
 - Automation
 - Continuous integration** (circled in red)
 - Deployment
 - Security

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

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** 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**
- Java with Gradle**
- Publish Java Package**

- 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

Q Gradle

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.)

The screenshot shows the GitHub code editor interface. The top navigation bar includes links for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The current file is `greetings-actions/.github/workflows/gradle.yml`, which is in the `main` branch. The code content is:

```
1 # This workflow uses actions that are not certified by GitHub.
2 # They are provided by a third-party and are governed by
3 # separate terms of service, privacy policy, and support
```

TO

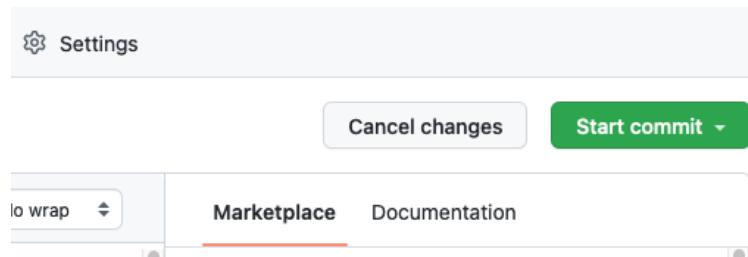
The screenshot shows the GitHub code editor interface. The top navigation bar includes links for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The current file is `greetings-actions/.github/workflows/simple-pipe.yml`, which is in the `main` branch. The code content is:

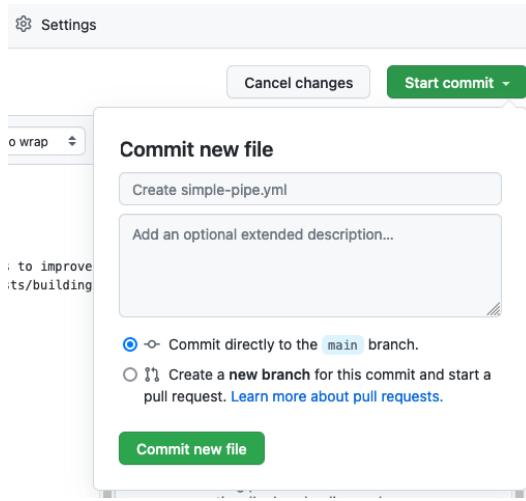
```
1 # This workflow uses actions that are not certified by GitHub.
2 # They are provided by a third-party and are governed by
3 # separate terms of service, privacy policy, and support
4 # documentation.
```

- 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:
```

- 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			
Java CI with Gradle	Management	Caches	New workflow
0 workflow runs Event Status Branch Actor			
Create simple-pipe.yml <small>Java CI with Gradle #1: Commit 9372ba3 pushed by gwstudent2</small> main 11 seconds ago In progress			

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. (You can ignore the warnings about 'save-state'.)

All workflows			
Simple Pipe	Management	Caches	New workflow
1 workflow run Event Status Branch Actor			
Create simple-pipe.yml <small>Simple Pipe #1: Commit 3bdac83 pushed by gwstudent2</small> main 2 minutes ago 29s			

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

The screenshot shows the GitHub repository gwstudent2/greetings-actions. The Actions tab is selected. Under "All workflows", the "Simple Pipe" workflow is selected. The "simple-pipe.yml" link is highlighted with a red circle. The "build" job is shown with a green checkmark and a red circle around it. The logs for the build job show the "Run actions/checkout@v3" step, which is also circled in red.

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 repository gwstudent/greetings-actions. The Actions tab is selected. Under "All workflows", the "Simple Pipe" workflow is selected. The "simple-pipe.yml" link is highlighted with a red circle. A workflow run for "Simple Pipe" is listed, showing a single run that renamed extra/simple-pipe.yml to .github/workflows/simple-pipe.yml. The commit message is "Simple Pipe #1: Commit 2393804 pushed by gwstudent".

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

```

1  # This workflow will build a Java project with Gradle initially
2  # For more information see: https://help.github.com/actions/language-and-framework-guides/building-and-testing-java-with-gradle
3
4  name: Simple Pipe
5
6  on:
7    push:
8      branches: [ main ]
  
```

- 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) ★ 13
- Cloud Storage Uploader** By google-github-actions (51) ★ 51
- Upload a Build Artifact** By actions (1.1k) ★ 1.1k

Upload a build artifact that can be used by subsequent workflow steps
- Run tfsec with sarif upload** By aquasecurity (17) ★ 17
- twine-upload** By yaananth ★ 1

Marketplace Documentation

Marketplace / Search results / Upload a Build Artifact

Upload a Build Artifact

By actions (1.1k) v3.0.0 ★ 1.5k

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

[View full Marketplace listing](#)

Installation

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

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

- This should open up the full GitHub Actions Marketplace listing for this action. Notice the URL at the top - <https://github.com/marketplace/actions/upload-a-build-artifact>. You can use this same relative URL to see other actions that are in the marketplace. For example, let's look at the checkout one we're already using. Go to <https://github.com/marketplace/actions/checkout>

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

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

[View on Marketplace](#)

Links

- [actions/checkout](#) (Red circle)
- [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

[View on Marketplace](#)

Actions (Red circle)

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

Releases 16

Release	Version	Last Published
v2.3.4	Latest	on Nov 3, 2020
+ 15 releases		

The screenshot shows the GitHub Actions checkout page. On the left, there's a sidebar with 'Workflows' and a 'All workflows' button highlighted in blue. The main area is titled 'All workflows' and shows 'Showing runs from all workflows'. Below this is a search bar labeled 'Filter workflow runs'. A table lists two workflow runs: 'Create check-dist.yml (#566)' and 'Create check-dist.yml (#566)'. Both runs were triggered by a 'Build and Test' event on the 'main' branch, 14 days ago, with a duration of 3m 39s and 45s respectively.

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.** 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
```

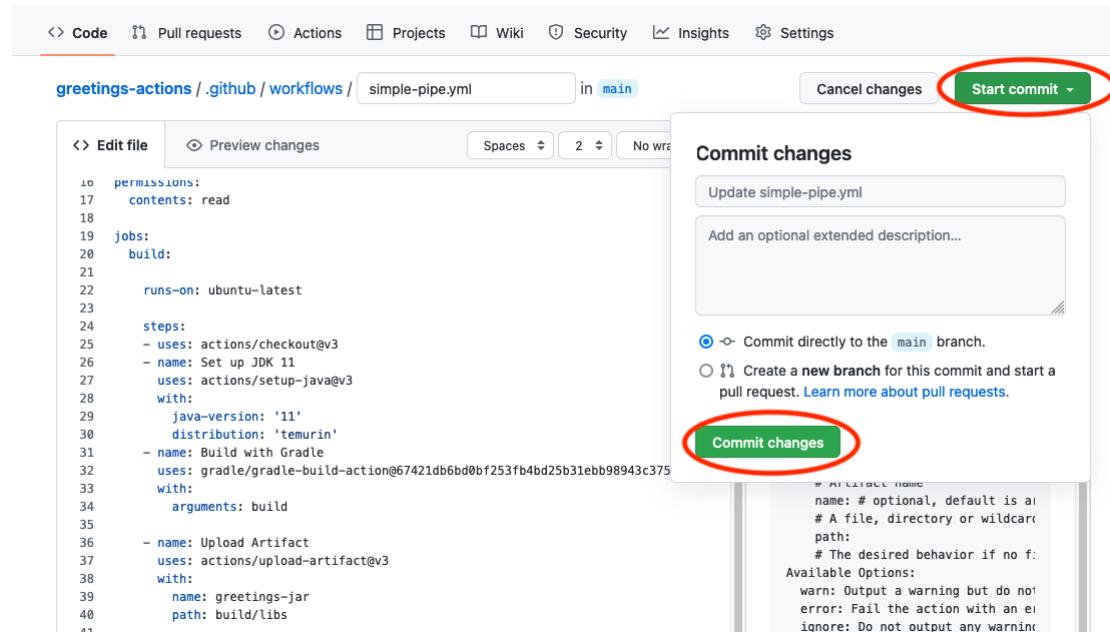
The screenshot shows the GitHub Actions workflow editor. The code editor displays the following YAML configuration:

```
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
```

Line 36, which adds the 'Upload Artifact' step, is highlighted in blue. Line 41 is also visible at the bottom of the code editor.

- 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.



- 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 the GitHub Actions run details for the 'Update simple-pipe.yml' workflow. The 'Summary' card provides an overview of the run: triggered via push 2 minutes ago, by gwstudent, main branch, success status, total duration of 26s, and 1 artifact produced. Below this, the 'simple-pipe.yml' job is detailed, showing it was triggered on push and completed successfully in 13s. At the bottom, the 'Artifacts' section lists a single artifact named 'greetings-jar' with a size of 1006 Bytes. The name 'greetings-jar' is circled in red.

9. 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 }}
```

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

<> 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 }}
```

- Click on the green "Start commit" 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.

The screenshot shows a GitHub Actions workflow named "simple-pipe.yml" for a repository titled "Simple Pipe #3". The workflow was triggered via push 17 seconds ago by gwstudent pushed to branch main. The status is "In progress". The total duration is listed as "-". There are no artifacts. The workflow graph shows a single job named "build" followed by a job named "test-run" with a duration of 7s between them. A "Cancel workflow" button is visible in the top right corner.

Lab 3: 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. 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 repository was forked from "gwstudent/arg-count-action". The fork count is highlighted with a red oval. The repository has 1 star and 1 fork. The "Code" tab is selected. A "Publish this Action to Marketplace" button is present. The repository has 8 tags and no packages published. The last commit was made by techupskills Delete iterate-args.sh 1 hour ago.

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

Take a look at the action.yml file and see if you can understand what its doing. The syntax is 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/extras/count-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 }}
```

<> Edit file  Preview changes Spaces 2 No wrap

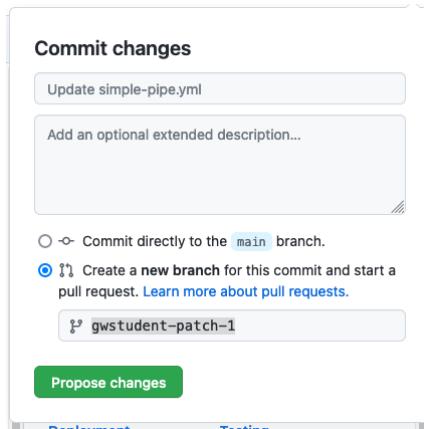
```

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 "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.

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.

Open 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** [Hide all checks](#)

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
	 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](#) ▾ You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

✖ Update simple-pipe.yml Simple Pipe #4

The screenshot shows the GitHub Actions pipeline summary for a pull request. 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 shows the job name 'count-args' and a status message 'failed 6 minutes ago in 0s'. Under 'Set up job', a list of steps is shown, ending with step 31 which has a red error message: 'Error: Unable to resolve action `<your github userid>/arg-count-action@main` , repository not found'.

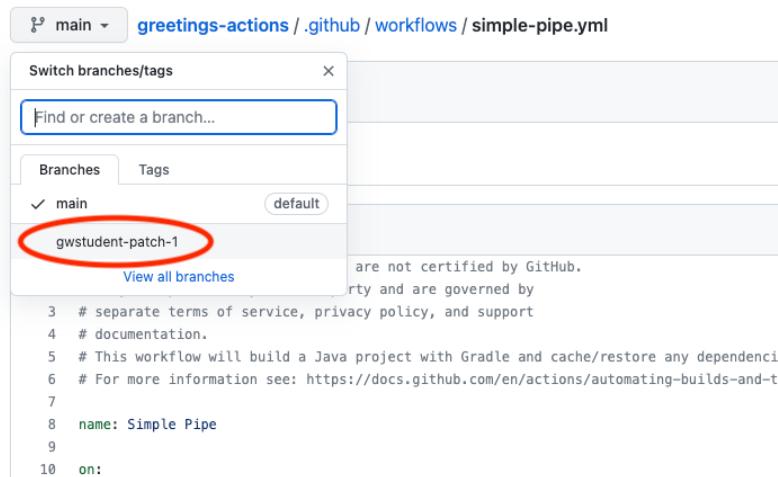
7. In the left column, click on the "Summary" link. This will take you back to the main graph page where you can also see the error.

✖ Update simple-pipe.yml Simple Pipe #5

This screenshot shows the GitHub Actions pipeline summary for a pull request. The left sidebar includes a 'Summary' link (circled in red) and a 'Jobs' section with 'build' (green), 'count-args' (red X), and 'test-run' (green). The main area features a pipeline graph. The 'simple-pipe.yml' workflow starts with 'build' (14s), followed by 'test-run' (5s). A separate box shows the 'count-args' job (33s) with a red X. Below the graph, an 'Annotations' section lists a single error from the 'count-args' job: 'Unable to resolve action `<your github userid>/arg-count-action@main` , repository not found'. At the bottom, an 'Artifacts' section shows a single artifact named 'greetings-jar' produced during runtime.

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.)



9. Edit the simple-pipe.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.

```
+v
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 "Start commit" 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".

greetings-actions/.github/workflows/simple-pipe.yml in gwstudent-patch-2

Commit changes

Update simple-pipe.yml
Fix GitHub userid

-o- Commit directly to the gwstudent-patch-2 branch.
 ⚡ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes

By actions
Close issues and pull requests with no recent activity

```

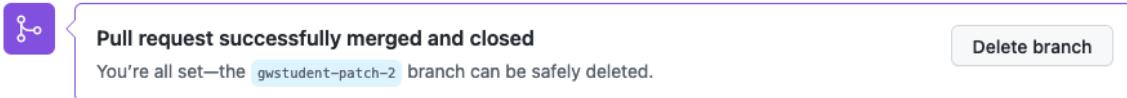
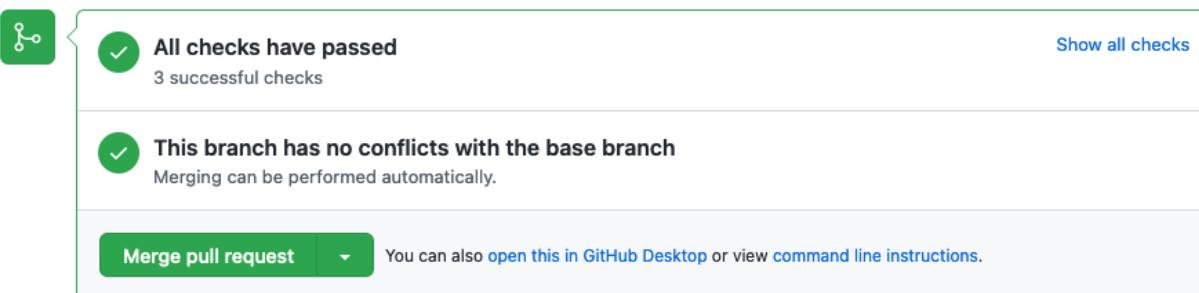
41   with:
42     name: greetings-jar
43   - shell: bash
44   run: |
45     java -jar greetings-actions.jar ${{ github.event.inputs.myValues }}
46
47 count-args:
48
49   runs-on: ubuntu-latest
50
51   steps:
52     - id: report-count
53       uses: gwstudent/arg-count-action@main
54     with:
55       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

```

Use **Control + Space** to trigger autocomplete in most situations.

- 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.

The screenshot shows the GitHub Workflows interface. On the left, there's a sidebar with 'Workflows' and 'New workflow' buttons, and a 'All workflows' button which is highlighted in blue. Below the sidebar, there's a search bar with the placeholder 'Filter workflow runs'. The main area is titled 'All workflows' with the subtitle 'Showing runs from all workflows'. It displays '7 workflow runs'. One run is visible: 'Merge pull request #2 from gwstudent/gwstudent-' followed by a truncated branch name. The status is 'main' with a green checkmark. To the right of the status are two timestamped entries: '43 seconds ago' and '40s'. There are also 'Event', 'Status', 'Branch', and 'Actor' dropdown filters at the top of the run list.

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.

The screenshot shows the GitHub Actions interface. The top navigation bar includes 'Code', 'Pull requests', 'Actions' (which is highlighted in red), 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. Below the navigation, there's a sidebar with 'Actions', 'New workflow', and a 'All workflows' button. The main content area is titled 'All workflows' with the subtitle 'Showing runs from all workflows'. A search bar on the right contains the query 'branch:patch-1'. The results show two workflow runs: one successful run 'Update simple-pipe.yml' and one failed run 'Update simple-pipe.yml'. Both runs are associated with 'patch-1'. To the right of each run are timestamps ('3 minutes ago', '16 minutes ago') and three-dot ellipsis buttons. There are also 'Event', 'Status', 'Branch', and 'Actor' dropdown filters at the top of the run list.

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

7 workflow runs

Merge pull request #2 from gwstudent/gwstudent-... main
Simple Pipe #7: Commit 274fa3a pushed by gwstudent

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

Filter by branch

Find a branch

main

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".

Actions

New workflow

All workflows

Simple Pipe

Management

Simple Pipe

simple-pipe.yml

7 workflow runs

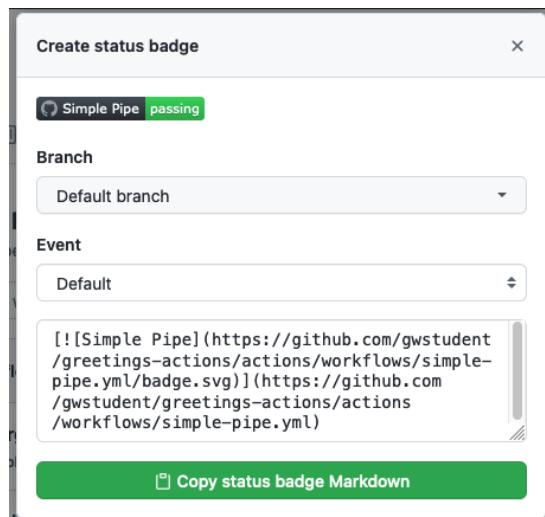
Merge pull request #1 from gwstudent2/patch-1 main
Simple Pipe #7: Commit 2bd7bbd pushed by gwstudent2

Filter workflow runs

Create status badge

Disable workflow

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 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-actions / README.md in main

Cancel changes

<> Edit new file

Preview

Spaces

2

No wrap

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

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

main ➔ greetings-actions / README.md

Go to file

...

gwstudent2 Create README.md ✓

Latest commit 8c7f16b 1 minute ago

History

1 contributor

3 lines (3 sloc) | 285 Bytes

<> Raw Blame

...

...

...

...

greetings-actions

Simple hello world type of program for use with learning GitHub Actions

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.

Code Pull requests Actions Projects Wiki Security Insights Settings

← Simple Pipe

✓ Create README.md #8

Re-run all jobs

...

Summary

Jobs

build

Triggered via push 3 minutes ago

gwstudent2 pushed → 8c7f16b main

Status

Success

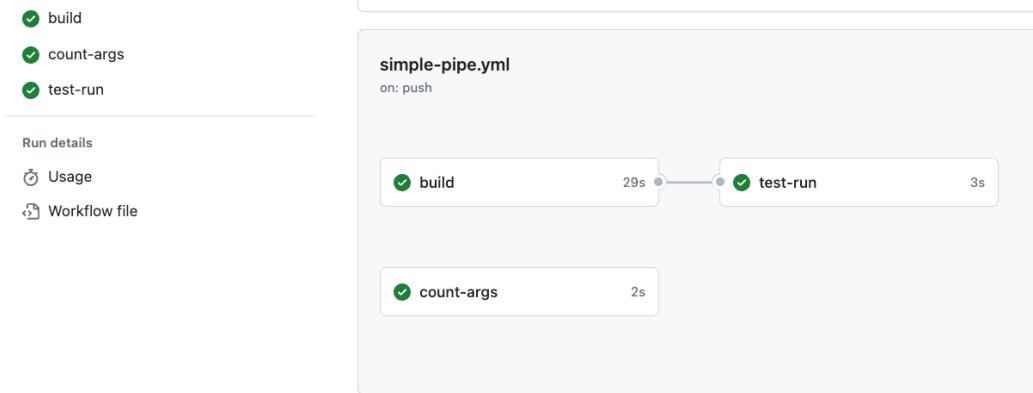
Total duration

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 the logs for the "test-run" job of a GitHub Actions pipeline. The job succeeded 3 minutes ago in 4 seconds. The logs are displayed in a tree view. The "Run java -jar greetings-actions.jar" step is expanded, showing the following log output:

```

1 Mon, 06 Sep 2021 17:06:05 GMT > Run java -jar greetings-actions.jar
2 Mon, 06 Sep 2021 17:06:05 GMT   java -jar greetings-actions.jar
3 Mon, 06 Sep 2021 17:06:05 GMT   shell: /usr/bin/bash --noprofile --norc -e -o pipefail {0}
4 Mon, 06 Sep 2021 17:06:06 GMT   Greetings!
5 Mon, 06 Sep 2021 17:06:06 GMT   No arguments were passed in.

```

A red circle highlights the arrowhead after the timestamp in the first log entry. Another red circle highlights the gear icon in the top right corner of the logs interface.

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.

Getting Started

```
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-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
2021-09-04T18:24:12.5723909Z Deployments: write
```

- 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" and "push")

workflow_dispatch:

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

The screenshot shows the GitHub Actions workflow editor for the file `simple-pipe.yml`. The code editor displays the following YAML configuration:

```
1 # This workflow will build a Java project with Gradle initially
2 # For more information see: https://help.github.com/actions/language-and-framework-guides/building-and-testing
3
4 name: Simple Pipe
5
6 on:
7   push:
8     branches: [ main ]
9   pull_request:
10    branches: [ main ]
11   workflow_dispatch:
12     inputs:
13       myValues:
14         description: 'Input Values'
15
16
17 jobs:
18   build:
```

The line `workflow_dispatch:` has been added to the `on` section. The GitHub interface shows the commit status as "Pending" and includes a "Start commit" button.

- 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 interface for the 'Simple Pipe' workflow. At the top, there are navigation links: Code, Pull requests, Actions (which is selected), Projects, Wiki, Security, Insights, and Settings. Below this, there's a 'Workflows' section with a 'New workflow' button and a 'simple-pipe.yml' file link. A 'Filter workflow runs' input field and a 'More' button are also present. The main area displays '9 workflow runs' with the following details:

- Update simple-pipe.yml**: Simple Pipe #9: Commit 6b4d7c8 pushed by gwstudent (main branch)
- Create README.md**: Simple Pipe #8: Commit 05235e3 pushed by gwstudent (main branch)
- Merge pull request #2 from gwstudent/gwstudent-p...**: (main branch)

To the right of the runs, there are filters for Event, Status, Branch, and Actor. On the far right, there are buttons for 'Run workflow' (with dropdown for branch and input values) and 'Run workflow' (green 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.

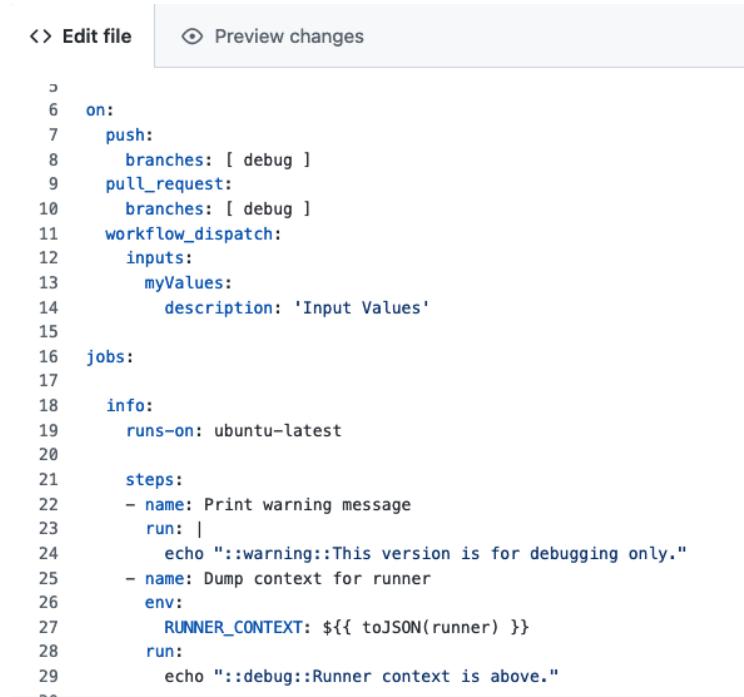
- First, let's create a new branch in GitHub for the debug instances of our workflows. On the repository's Code page, click on the drop-down under "main", and enter "debug" in the "Find or create a branch..." field. Then click on the "Create branch: debug from 'main'" link in the dialog.

The screenshot shows the GitHub repository's branches page. The main header includes Code, Issues (3), Pull requests, and a profile icon. Below the header, it shows 1 main branch, 2 branches, and 0 tags. A dropdown menu is open over the 'main' branch, showing the option 'Create branch: debug from \'main\''. This option is circled in red. Other options like 'Switch branches/tags' and 'Delete branch' are also visible.

- 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."
```

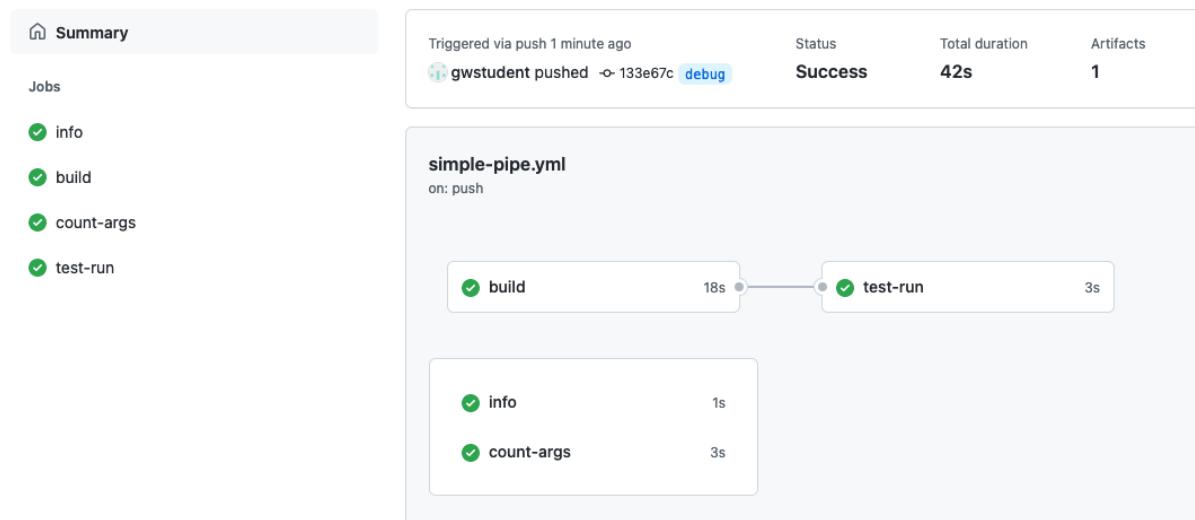


The screenshot shows a GitHub Actions workflow configuration in a code editor. At the top, there are buttons for 'Edit file' and 'Preview changes'. The code itself is a YAML file with the following content:

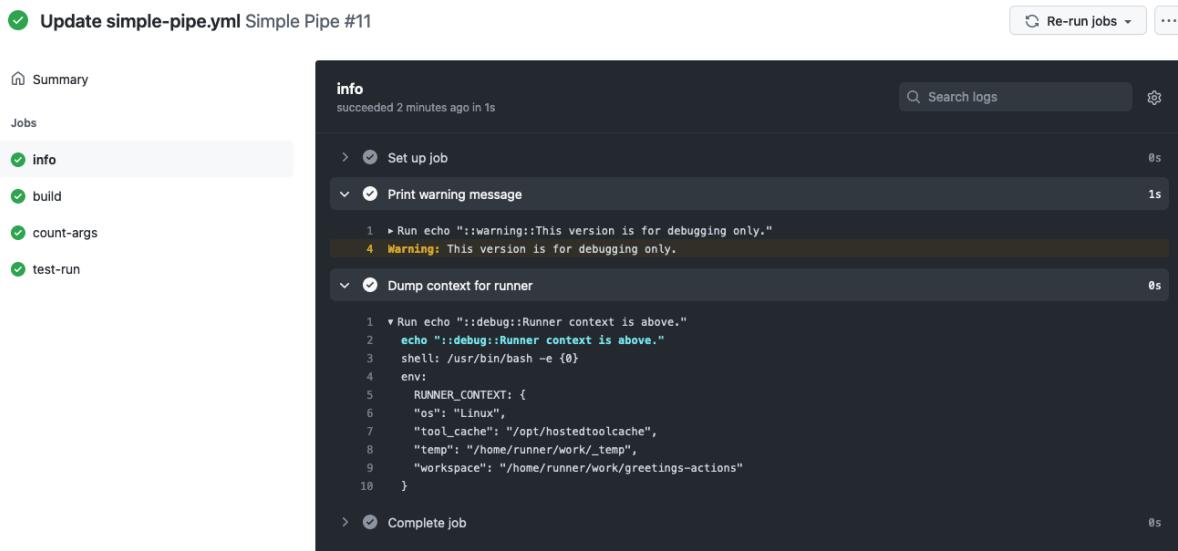
```
3  
4  on:  
5    push:  
6      branches: [ debug ]  
7    pull_request:  
8      branches: [ debug ]  
9    workflow_dispatch:  
10       inputs:  
11         myValues:  
12           description: 'Input Values'  
13  
14 jobs:  
15  
16   info:  
17     runs-on: ubuntu-latest  
18  
19     steps:  
20       - name: Print warning message  
21         run: |  
22           echo "::warning::This version is for debugging only."  
23       - name: Dump context for runner  
24         env:  
25           RUNNER_CONTEXT: ${{ toJSON(runner) }}  
26         run:  
27           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



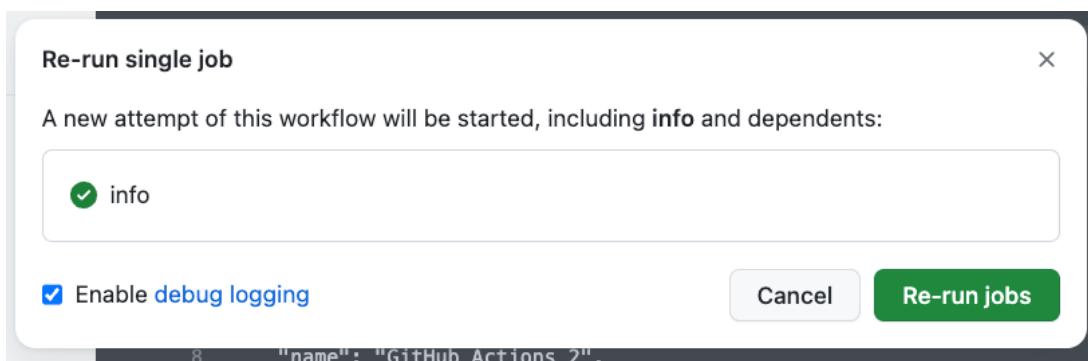
4. Expand the entries for "Print warning message" and "Dump context for runner" to see the outputs for those.



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'. Under 'Jobs', the 'info' job is highlighted with a blue bar and has a green checkmark icon. To its right is a list of other jobs: 'build', 'count-args', and 'test-run', each with a green checkmark icon. To the right of the job list is a dark panel with a search bar labeled 'Search logs' and icons for refresh and settings.

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 job log for the 'info' job. The job status is 'succeeded 3 minutes ago in 1s'. The log is displayed in a dark-themed interface with a search bar at the top. The log content is as follows:

```
info
succeeded 3 minutes ago in 1s
Search logs
> ✓ 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",
11 ##[debug]  "url": "https://github.com/actions/toolkit@v2"
12 ##[debug]}'
```

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". 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. A red circle labeled '1' highlights the 'Settings' tab in the top navigation bar. A red circle labeled '2' highlights the 'Secrets and variables' option in the left sidebar under the 'Actions' section. A red circle labeled '3' highlights the 'Variables' tab in the main content area, which is currently selected. A red circle labeled '4' highlights the green 'New repository variable' button.

Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. [Learn more about encrypted secrets](#). Variables are shown as plain text and are used for **non-sensitive** data. [Learn more about variables](#).

Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.

Variables New repository variable

Environment variables Manage environments

There are no variables for this repository's environments.

Repository variables

There are 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.

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 *

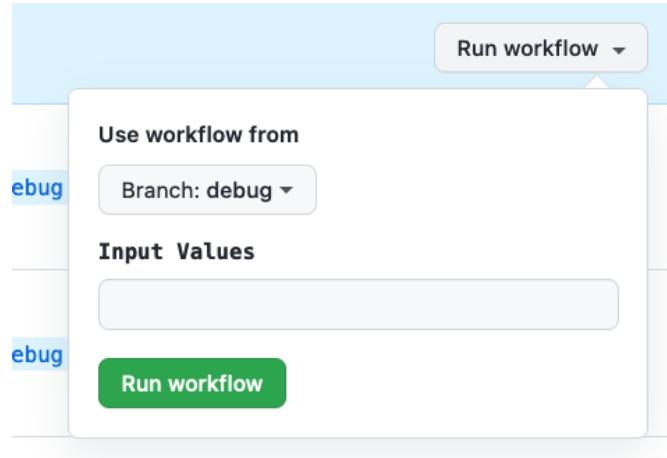
ACTIONS_STEP_DEBUG

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
Search logs
0s

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

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

- For this lab, we need to prepare a Personal Access Token (PAT) and add it to a secret that our workflow can reference. If you already have a PAT, you may be able to use it if it has access to the project. If not, you'll need to create a new one. Go to <https://github.com/settings/tokens>.

(Alternatively, on the GitHub repo screen, click on your profile picture in the upper right, then select "Settings" from the drop-down menu. You should be on the <https://github.com/settings/profile> screen. On this page on the left-hand side, select "Developer settings" near the bottom. On the next page, select "Personal access tokens", then "Tokens (classic)".)

Settings / Developer settings

GitHub Apps
OAuth Apps
Personal access tokens Beta
Fine-grained tokens
Tokens (classic)

Personal access tokens (classic)

Generate new token ▾

Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#).

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

2. In the upper right, click on "Generate new token". You can just use the "classic" token, so click on the "Generate new token (classic)" selection if needed. Confirm your password if prompted.

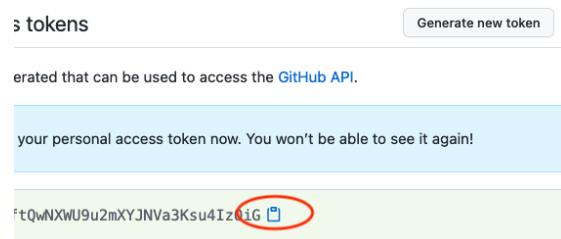
The screenshot shows the GitHub 'Personal access tokens (classic)' page. On the left, there's a sidebar with 'GitHub Apps', 'OAuth Apps', and 'Personal access tokens' (which is expanded, showing 'Fine-grained tokens' and 'Tokens (classic)'). A green 'Beta' badge is next to 'Personal access tokens'. On the right, the main area has a heading 'Personal access tokens (classic)' and a note: 'Need an API token for scripts or testing access to the [GitHub API](#)'. Below this are two buttons: 'Generate new token (Beta)' (highlighted with a red box) and 'Generate new token (classic)' (disabled, with a greyed-out note: 'For general use'). A tooltip for 'Generate new token (classic)' says: 'Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.'

3. Enter whatever text you want in the "Note" section. Confirm your password if asked. In the "Note" section enter some text, such as "workflows". You can set the "Expiration" time as desired or leave it as-is. Under "Select scopes", assuming your repository is public, you can just check the boxes for "repo" and "workflow". Then click on the green "Generate token" at the bottom.

The screenshot shows the 'New personal access token (classic)' configuration page. The left sidebar is identical to the previous one. The main area has a heading 'New personal access token (classic)' and a note: 'Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.' Below this is a 'Note' field containing 'workflows'. A question 'What's this token for?' follows. Under 'Expiration *', a dropdown shows '30 days' with a note: 'The token will expire on Mon, May 8 2023'. The 'Select scopes' section lists several options with checkboxes. Most checkboxes are unchecked except for 'repo' and 'workflow', which are checked. Descriptions for each scope are provided.

Scope	Description
<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input checked="" type="checkbox"/> workflow	Update GitHub Action workflows

- After the screen comes up that shows your new token, make sure to copy it and store it somewhere you can get to it for future steps.



- Now we'll create a new secret and store the PAT value in it. Go back to the repository (not your personal account) and select "Settings" from the top menu. Then on the left-hand side, select "Secrets and variables" and then "Actions". Now, click on the "New repository secret" button in the middle right to create a new secret for the workflow to use.

A screenshot of a GitHub repository's settings page. The top navigation bar includes 'Code', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings' (which is circled in red with a number '1'). The left sidebar has sections like 'General', 'Access', 'Collaborators', 'Moderation options', 'Code and automation', 'Branches', 'Tags', 'Actions' (which is circled in red with a number '2'), 'Webhooks', 'Environments', 'Codespaces', 'Pages', 'Security', 'Code security and analysis', 'Deploy keys', 'Secrets and variables' (which is circled in red with a number '3'), and 'Actions'. The main content area is titled 'Actions secrets and variables'. It contains a section for 'Environment secrets' with a 'Manage environments' button and a note that there are no secrets for this repository's environments. Another section for 'Repository secrets' shows a similar note. A green button labeled 'New repository secret' is highlighted with a red oval and a number '3'.

6. For the *Name* of the new secret, use *envPAT*. Paste the value from the PAT into the *Value* section. Then click on the "Add secret" button at the bottom. After this, the new secret should show up in the list of repository secrets.

The screenshot shows the GitHub Actions secrets configuration page for a repository. On the left, there's a sidebar with sections like General, Access, Collaborators, Moderation options, Code and automation, Branches, Tags, Actions, Webhooks, Environments, Codespaces, and Pages. The main area is titled 'Actions secrets' and contains fields for 'Name' (set to 'envPAT') and 'Secret' (set to 'ghp_ANNsMMd5taCGoi'). A blue box highlights the 'Add secret' button at the bottom of this section. To the right, there are tabs for 'Secrets' (selected) and 'Variables'. Below the tabs, there are two sections: 'Environment secrets' (which says 'There are no secrets for this repository's environments.') and 'Repository secrets' (which lists 'ENVPAT' with an 'Updated now' timestamp and edit/delete icons). A green 'New repository secret' button is located at the top right of the main content area.

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

The screenshot shows the GitHub repository settings page. On the left, there are navigation links for 'Getting Started', 'Pages', 'Moderation settings', and an 'Edit' button. The main content area is titled 'Features' and contains three checkboxes: 'Wikis' (checked), 'Restrict editing to collaborators only' (checked), and 'Issues' (checked, with a red circle around it). Below these checkboxes, there's a note: 'Issues integrate lightweight task tracking into your repository. Keep projects on track with issue labels and milestones, and reference them in commit messages.' At the bottom, there's a call-to-action box with the text 'Get organized with issue templates' and 'Set up templates'.

8. 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.

- In the repository, browse to the "extra" folder and to the "create-failure-issue.yml" file.
- Take a few moments to look over the file and see what it does. Notice that:
 - It has a `workflow_call` section in the "on" area, which means it can be run from another workflow.
 - 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.

```

1 # This is a basic workflow to help you get started with Actions
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](#)

9. 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 a 'New workflow' button. Below it are 'All workflows' and 'Simple Pipe'. The 'create-failure-issue' workflow is selected and highlighted in blue. The main area shows the workflow name 'create-failure-issue' and its file 'create-failure-issue.yml'. It indicates '1 workflow run'. A message says 'This workflow has a workflow_dispatch event trigger.' Below this, a list shows a single run: 'create-failure-issue' with the note 'create-failure-issue #1: Manually run by gwstudent'. To the right, there are dropdowns for 'Event', 'Status', 'Branch', and 'Actor'. A 'Run workflow' button is visible. A modal window is open, titled 'Run workflow from Branch: main'. It contains fields for 'Issue title *' (with placeholder 'This is a title') and 'Issue body *' (with placeholder 'This is the body text'). At the bottom of the modal is a green 'Run workflow' button.

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

The screenshot shows the GitHub Issues page. The navigation bar at the top includes 'Code', 'Issues 1', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The 'Issues' tab is active. Below the navigation is a search bar with 'is:issue is:open' and buttons for 'Labels 9' and 'Milestones 0'. A 'New issue' button is also present. The main list shows one open issue: '#2 opened now by github-actions (bot)' with the title 'FAILURE: This is a title'. There are filters for 'Open' (1) and 'Closed' (0), and dropdowns for 'Author', 'Label', 'Projects', 'Milestones', 'Assignee', and 'Sort'.

11. 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:
```

```
  runs-on: ubuntu-latest
  needs: [test-run, count-args]
  if: always() && failure()
  steps:
    - name: invoke workflow to create issue
      run: >
```

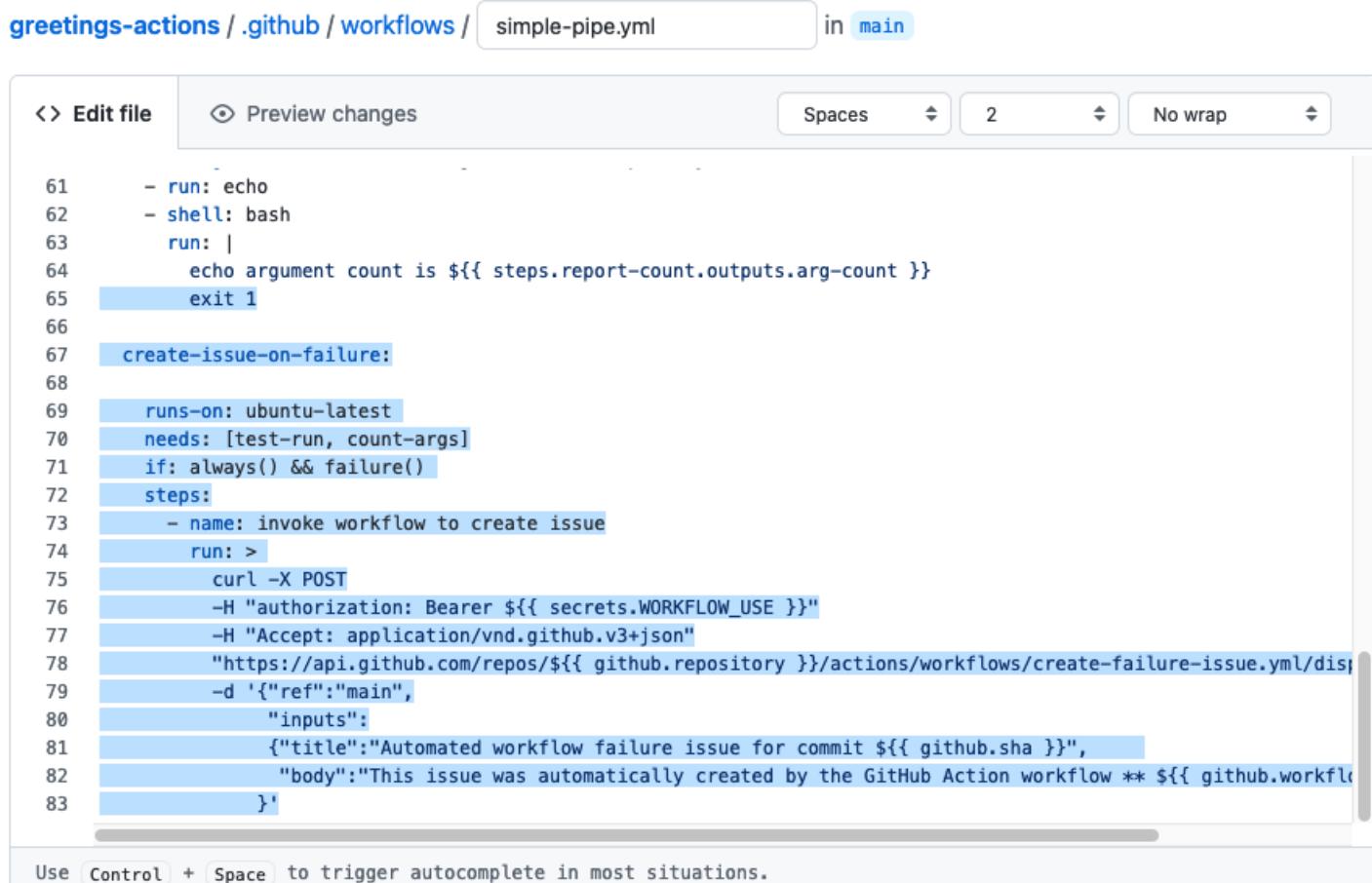
```

curl -X POST
-H "authorization: Bearer ${{ secrets.WORKFLOW_USE }}"
-H "Accept: application/vnd.github.v3+json"
"https://api.github.com/repos/${{ github.repository }}/actions/workflows/create-
failure-issue.yml/dispatches"
-d '{"ref":"main",
  "inputs":
    {"title":"Automated workflow failure issue for commit ${{ github.sha }}",
     "body":"This issue was automatically created by the GitHub Action workflow
** ${{ github.workflow }} **"}
}'

```

12. 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 main
<> Edit file Preview changes Spaces 2 No wrap
61   - run: echo
62   - shell: bash
63     run: |
64       echo argument count is ${{ steps.report-count.outputs.arg-count }}
65     exit 1
66
67   create-issue-on-failure:
68
69     runs-on: ubuntu-latest
70     needs: [test-run, count-args]
71     if: always() && failure()
72     steps:
73       - name: invoke workflow to create issue
74         run: >
75           curl -X POST
76           -H "authorization: Bearer ${{ secrets.WORKFLOW_USE }}"
77           -H "Accept: application/vnd.github.v3+json"
78           "https://api.github.com/repos/${{ github.repository }}/actions/workflows/create-failure-issue.yml/dis-
79           -d '{"ref":"main",
80             "inputs":
81               {"title":"Automated workflow failure issue for commit ${{ github.sha }}",
82                "body":"This issue was automatically created by the GitHub Action workflow ** ${{ github.workflow }} **
83             }'

```

Use `Control + Space` to trigger autocomplete in most situations.

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

Workflows New workflow

All workflows

Simple Pipe
create-failure-issue

All workflow runs

15 workflow runs

	Event	Status	Branch	Actor
✓ create-failure-issue create-failure-issue #2: Manually run by gwstudent	15 seconds ago	Success	main	gwstudent
✗ Update simple-pipe.yml Simple Pipe #14: Commit 1539282 pushed by gwstudent	1 minute ago	Failure	main	gwstudent

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"]
    B -- "2s" --> C["✓ create-issue-on-failure"]
    C -- "1s" --> D["✓ build"]
    style A fill:#ffccbc,color:#dc3545
    style B fill:#d9eaf7,color:#28a745
    style C fill:#d9eaf7,color:#28a745
    style D fill:#d9eaf7,color:#28a745
    
```

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" --> B
    style A fill:#d9eaf7,color:#28a745
    
```

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

The screenshot shows a GitHub repository interface. At the top, there are navigation links: Code, Issues (2), Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Issues' link is highlighted. Below the navigation, the title of the issue is displayed: "Automated workflow failure issue for commit 153928267f2fc38a4e91b5bd00d61f033abd59d6 #4". A green button labeled "Open" is next to the issue title. Below the title, it says "github-actions (bot) opened this issue 3 minutes ago · 0 comments". A comment from "github-actions (bot)" is shown, stating: "This issue was automatically created by the GitHub Action workflow ** Simple Pipe **". There are three small profile icons at the bottom left of the comment area.

THE END - THANKS!