

Introduction to GitHub Actions

Revision 3.0 – 11/07/22

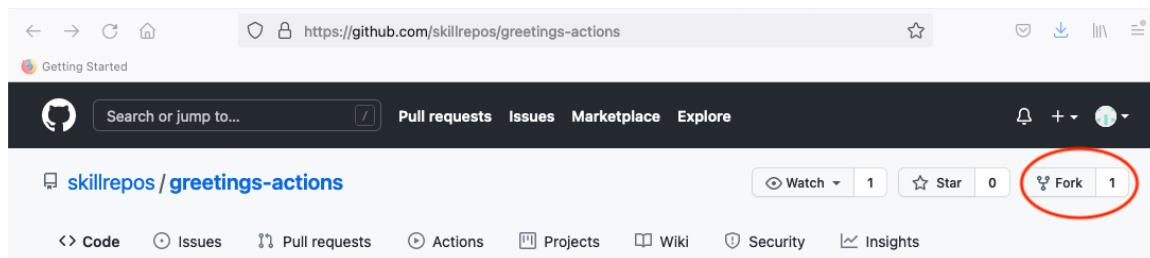
Tech Skills Transformations LLC / Brent Laster

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.



The screenshot shows the GitHub repository page for 'greetings-actions'. The URL in the address bar is <https://github.com/skillrepos/greetings-actions>. The repository name 'skillrepos / greetings-actions' is displayed. The 'Fork' button, which has a value of '1', is highlighted with a red circle. The repository has 1 branch and 0 tags. The 'Code' tab is selected. The commit history shows several commits from 'Brent Laster' and 'extra' with the message 'update files for labs'. The 'About' section describes it as a 'Simple hello world type of program for use with learning GitHub Actions'. The 'Releases' section shows 'No releases published'.

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

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

Author	Commit Message	Time
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 7 months ago
	gradle/wrapper	Initial commit 14 months ago
	src/main/java	update files for labs 14 months ago
	build.gradle	update files for labs 14 months ago
	gradlew	Initial commit 14 months ago
	gradlew.bat	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".

Automation

Greetings
 By GitHub Actions
 Greets users who are first time contributors to the repo

Stale
 By GitHub Actions
 Checks for stale issues and pull requests

Manual workflow
 By GitHub Actions
 Simple workflow that is manually triggered.

Labeler
 By GitHub Actions
 Labels pull requests based on the files changed

Browse all categories

- [Automation](#)
- [Continuous integration](#) **Configure** Automation
- [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.

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](#) **Configure** Automation
- [Deployment](#)
- [Security](#)

Q: Gradle

Found 52 workflows

Android CI
 By GitHub Actions
 Build an Android project with Gradle.

Java with Ant
 By GitHub Actions
 Build and test a Java project with Apache Ant.

Clojure
 By GitHub Actions
 Build and test a Clojure project with Leiningen.

Publish Java Package

Java with Gradle

Publish Java Package

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

7. 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.)

TO

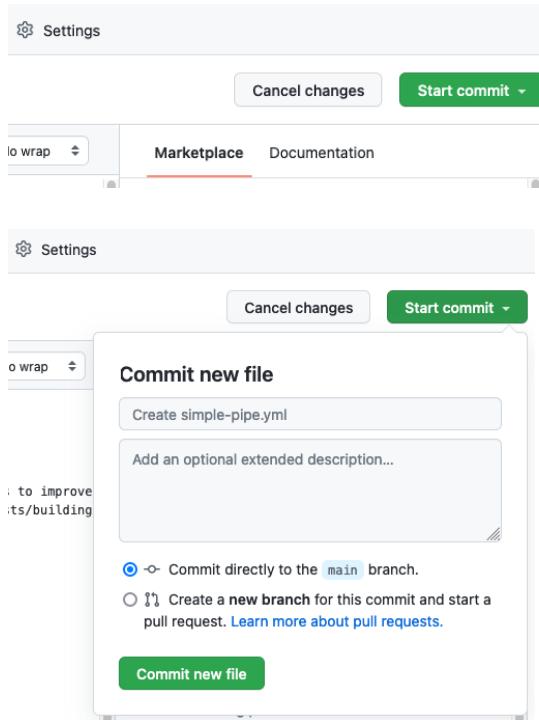
8. Now, edit the name of the workflow - change line 8 from "name: Java CI with Gradle" to "name: Simple Pipe".

```

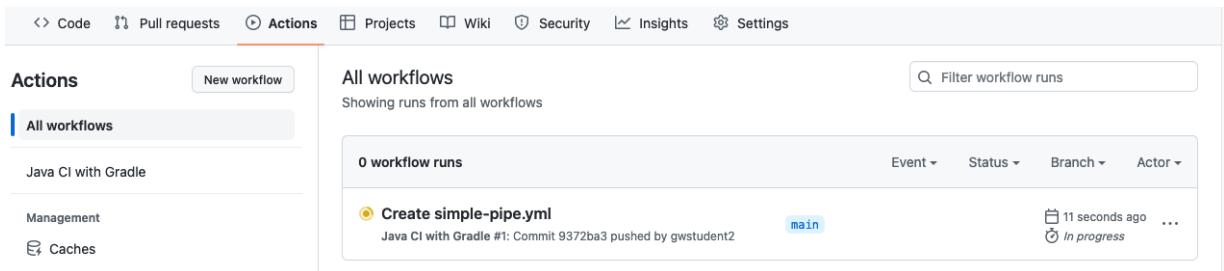
5 # This workflow will build a Java project 5 # This workflow will build a Java project
6 # For more information see: https://docs.github.com/actions/learn-github-actions/configure-a-workflow. 6 # For more information see: https://docs.github.com/actions/learn-github-actions/configure-a-workflow.
7
8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

```

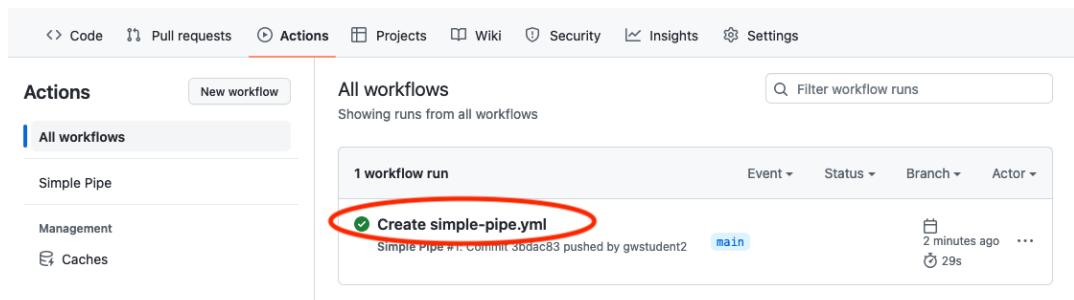
9. Now, we can go ahead and commit the new workflow via the “Start commit” button in the upper right. In the dialog that comes up, you can enter an optional comment if you want. Leave the “Commit directly...” selection checked and then click on the “Commit new file” button.



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



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.



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 a GitHub repository page for '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 also highlighted with a red circle. The job details show a log of steps, with the first step 'Run actions/checkout@v3' expanded, showing its sub-steps: 1. Run actions/checkout@v3, 12. Syncing repository: gwstudent2/greetings-actions, 13. Getting Git version info, and 17. Temporarily overriding HOME='/home/runner/work/_temp/7725ece7-0992-4326-ffa-60d9c4726904'.

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.

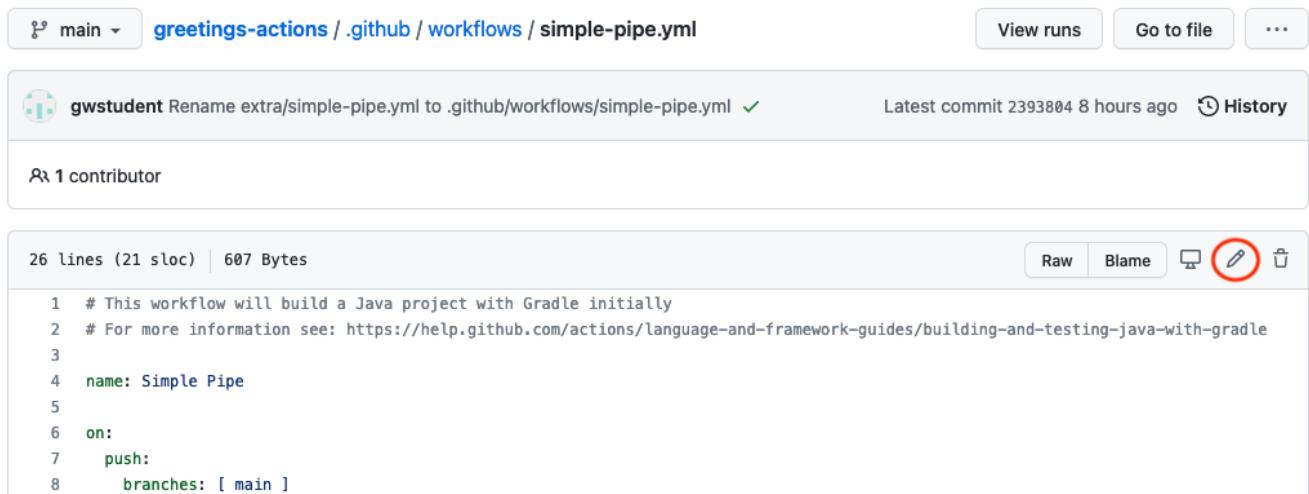
In your GitHub repository, click the Actions button at the top if not already on the Actions screen.

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

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

The screenshot shows a GitHub repository page for '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. The workflow run table shows 1 workflow run for the 'Simple Pipe' workflow. The run details show a step: 'Rename extra/simple-pipe.yml to .github/w...' with status 'Success' and timestamp '17 minutes ago'. The run was triggered by 'Simple Pipe #1: Commit 2393804 pushed by gwstudent' on the 'main' branch.

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



```

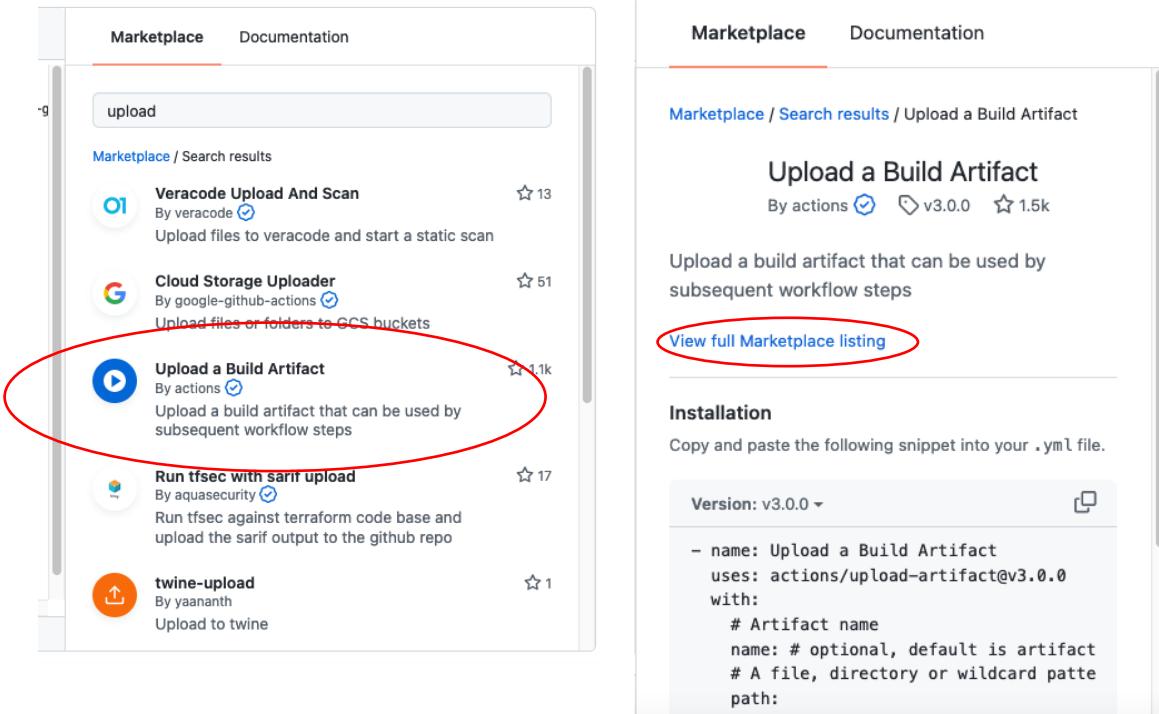
1 # This workflow will build a Java project with Gradle initially
2 # For more information see: https://help.github.com/actions/language-and-framework-guides/building-and-testing-java-with-gradle
3
4 name: Simple Pipe
5
6 on:
7   push:
8     branches: [ main ]

```

- 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 / Search results

upload

Veracode Upload And Scan By veracode (13)
Upload files to veracode and start a static scan

Cloud Storage Uploader By google-github-actions (51)
Upload files or folders to GCS buckets

Upload a Build Artifact By actions (1.1k)
Upload a build artifact that can be used by subsequent workflow steps

Run tfsec with sarif upload By aquasecurity (17)
Run tfsec against terraform code base and upload the sarif output to the github repo

twine-upload By yaananth (1)
Upload to twine

Marketplace / Search results / Upload a Build Artifact

Upload a Build Artifact

By actions (1.1k) v3.0.0 1.5k

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

[View full Marketplace listing](#)

Installation

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

Version: v3.0.0

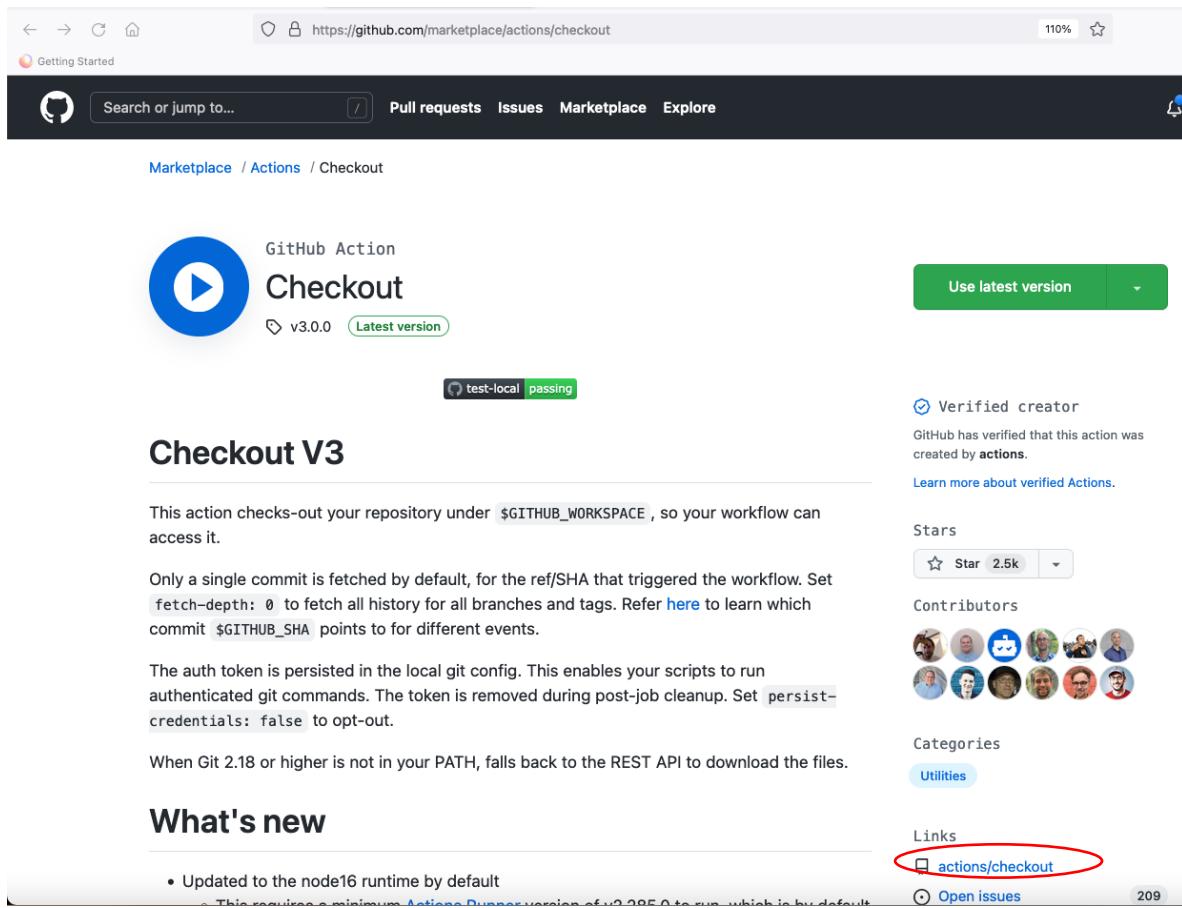
```

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

```

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

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



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

Checkout V3

This action checks-out your repository under `$GITHUB_WORKSPACE`, so your workflow can access it.

Only a single commit is fetched by default, for the ref/SHA that triggered the workflow. Set `fetch-depth: 0` to fetch all history for all branches and tags. Refer [here](#) to learn which commit `$GITHUB_SHA` points to for different events.

The auth token is persisted in the local git config. This enables your scripts to run authenticated git commands. The token is removed during post-job cleanup. Set `persist-credentials: false` to opt-out.

When Git 2.18 or higher is not in your PATH, falls back to the REST API to download the files.

What's new

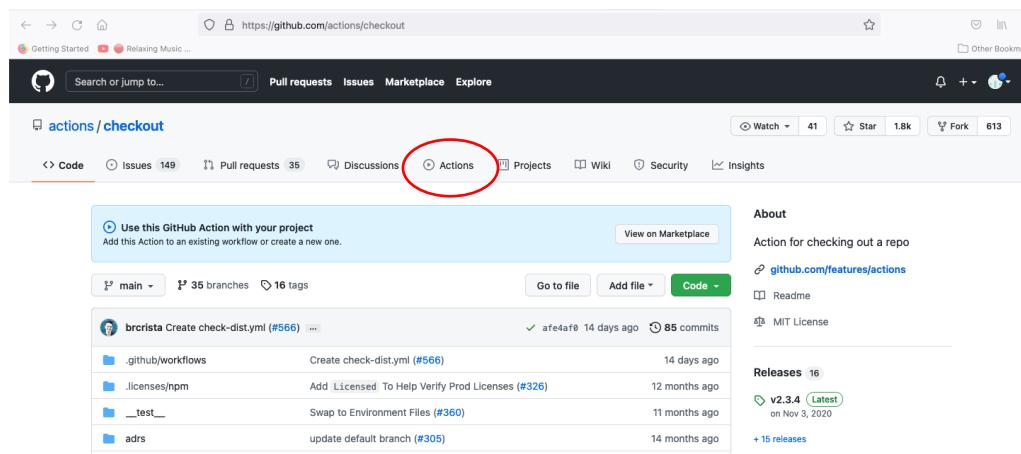
- Updated to the node16 runtime by default

This requires a minimum Actions Runner version of v2.205.0 to run, which is by default.

Links

- [actions/checkout](#)
- [Open issues](#)

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



[actions / checkout](#)

[Actions](#)

Code Issues 149 Pull requests 35 Discussions 16 Actions Projects Wiki Security Insights

Use this GitHub Action with your project
Add this Action to an existing workflow or create a new one.

main 35 branches 16 tags

Actions

Author	Action	Created	Commits
brcrista	Create check-dist.yml (#566)	14 days ago	85 commits
	.github/workflows Create check-dist.yml (#566)	14 days ago	
	.licenses/npm Add Licensed To Help Verify Prod Licenses (#326)	12 months ago	
	__test__ Swap to Environment Files (#360)	11 months ago	
	adrs update default branch (#305)	14 months ago	

About
Action for checking out a repo

[github.com/features/actions](#)

[Readme](#)

[MIT License](#)

Releases 16

v2.3.4 [\[Latest\]](#)
on Nov 3, 2020

+ 15 releases

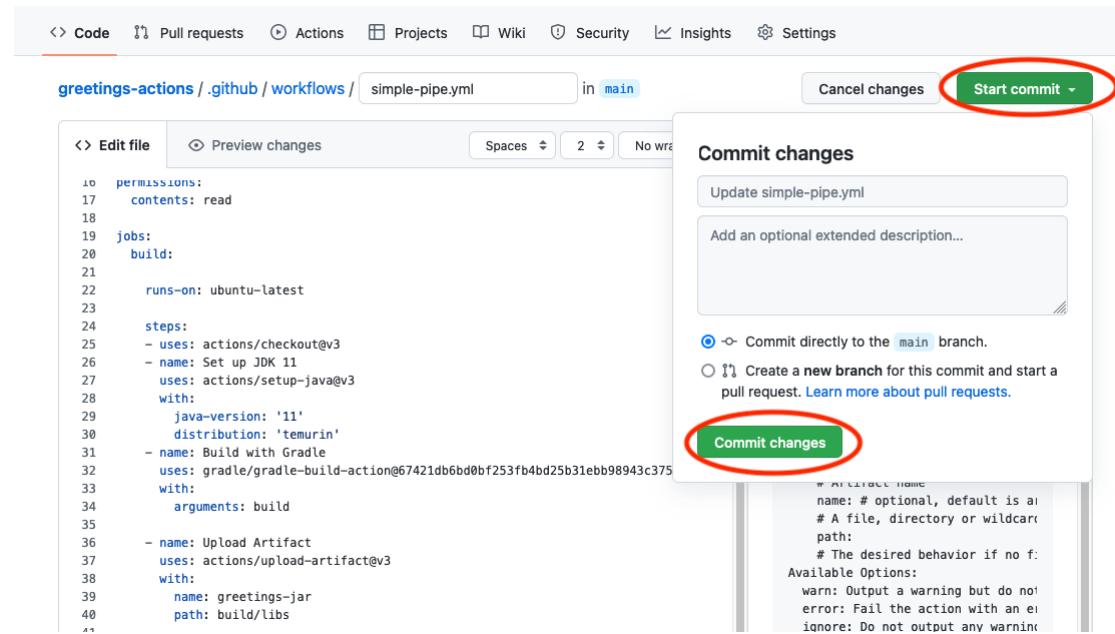
6. Switch back to the browser tab where you are editing the workflow for greetings-actions. Update the build job to include a new step to use the "upload-artifact" action to upload the jar the build job creates. To do this, add the following lines inline with the build job steps. **Pay attention to the indenting.** See the screenshot (lines 27-31) for how this should look afterwards. (Your line numbers may be different.)

```

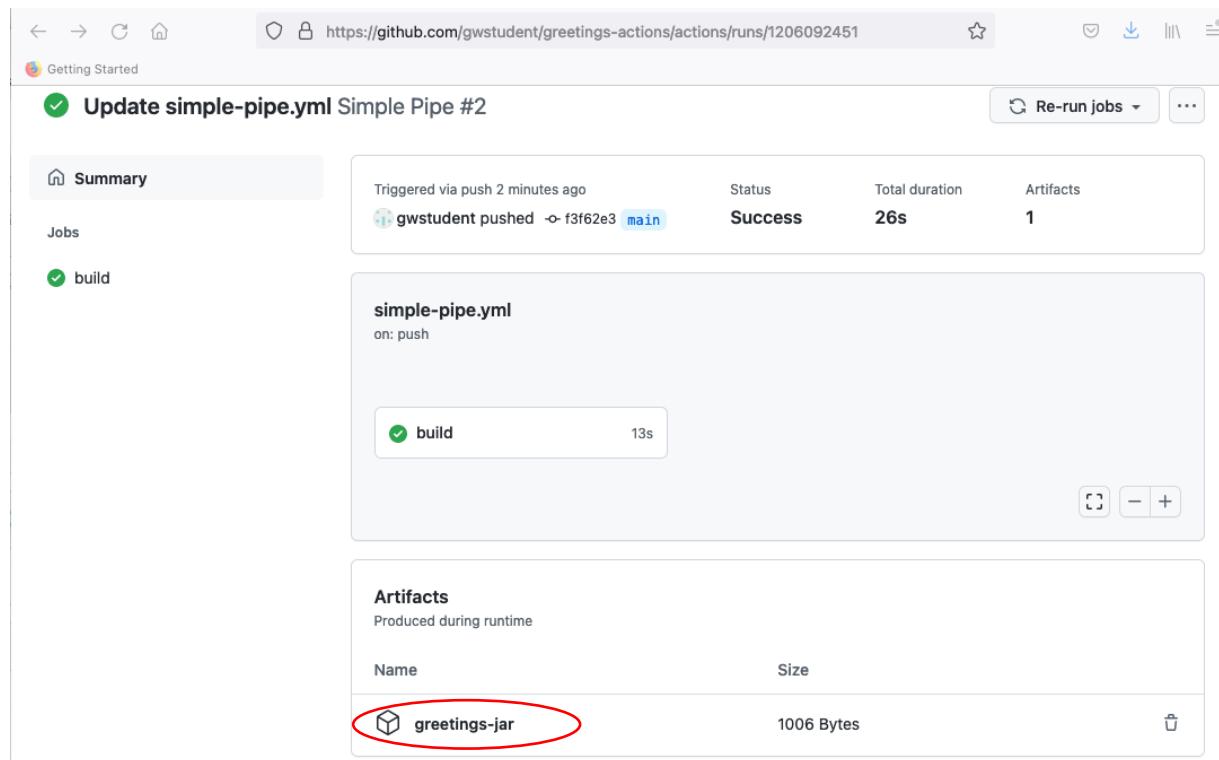
- name: Upload Artifact
  uses: actions/upload-artifact@v3
  with:
    name: greetings-jar
    path: build/libs
  
```

Page 8

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



8. Switch to the "Actions" tab in your repository to see the workflow run. After a few moments, you should see that the run was successful. Click on the title of that run "Update 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.



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

```

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 to 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 run for a repository. The workflow is named 'simple-pipe.yml' and is triggered by a push. The status is 'In progress'. The workflow graph shows two jobs: 'build' and 'test-run'. The 'build' job has a duration of 7s and is connected to the 'test-run' job. The 'test-run' job is currently in progress.

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.

The screenshot shows the GitHub repository for 'arg-count-action'. The repository has 1 fork. The repository page includes sections for 'About', 'Releases', and 'Packages'. The 'About' section describes it as a 'Simple GitHub Action demo'. The 'Releases' section shows 8 tags. The 'Packages' section shows 0 packages published. The repository has 0 stars and 0 issues.

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.yaml file in the Actions tab). 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:
```

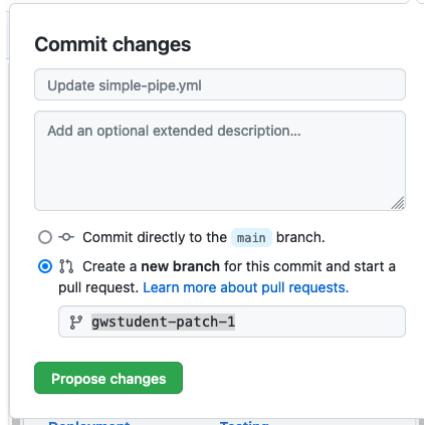
```
  runs-on: ubuntu-latest

  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 }}
```

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.

4. 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".



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.

base: main ▾ ← compare: patch-1 ▾ ✓ **Able to merge.** These branches can be automatically merged.

Discuss and review the changes in this comparison with others. [Learn about pull requests](#) Create pull request

1 commit 1 file changed 1 contributor

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

base: main ▾ ← compare: `gwstudent-patch-1` ▾ ✓ **Able to merge.** These branches can be automatically merged.

Update simple-pipe.yml

Write Preview H B I ≡ ↔ 🔗 ≡ ≡ ✓ ✉ ↗ ↶

Leave a comment

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

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.

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**

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

Summary

Jobs

- ✓ build
- ✖ count-args
- ✓ test-run

count-args
failed 6 minutes ago in 0s

Set up job

```

1 Current runner version: '2.285.0'
2 ▶ Operating System
6 ▶ Virtual Environment
11 ▶ Virtual Environment Provisioner
13 ▶ GITHUB_TOKEN Permissions
27 Secret source: Actions
28 Prepare workflow directory
29 Prepare all required actions
30 Getting action download info
31 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

↻ Re-run jobs ...

Summary

Jobs

- ✓ build
- ✖ count-args
- ✓ test-run

Triggered via pull request 14 minutes ago

gwstudent opened #2 gwstudent-patch-2

Status

Failure

Total duration

45s

Artifacts

1

simple-pipe.yml

on: pull_request

build

14s

test-run

5s

✖ count-args

33s

[] - +

Annotations

1 error

✖ count-args

Unable to resolve action `<your github userid>/arg-count-action@main` , repository not found

Artifacts

Produced during runtime

Name

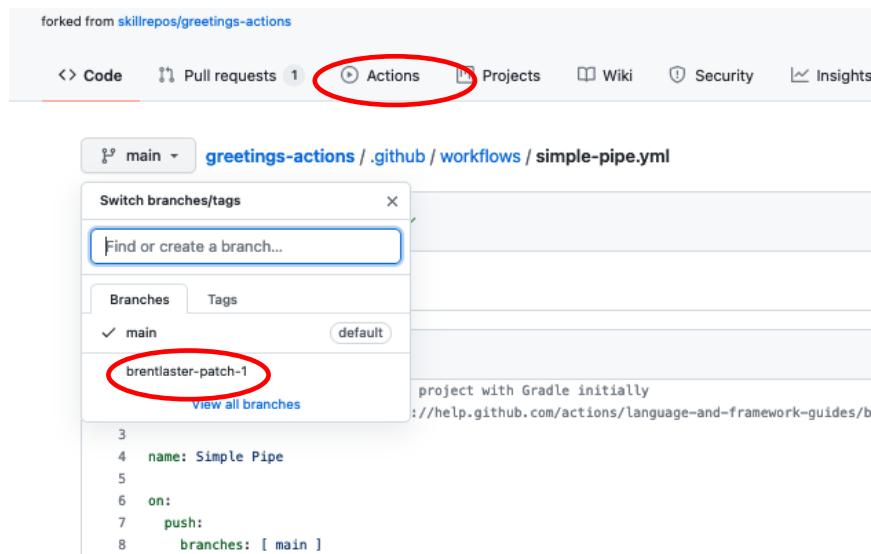
Size

greetings-jar

1006 Bytes

[]

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**.



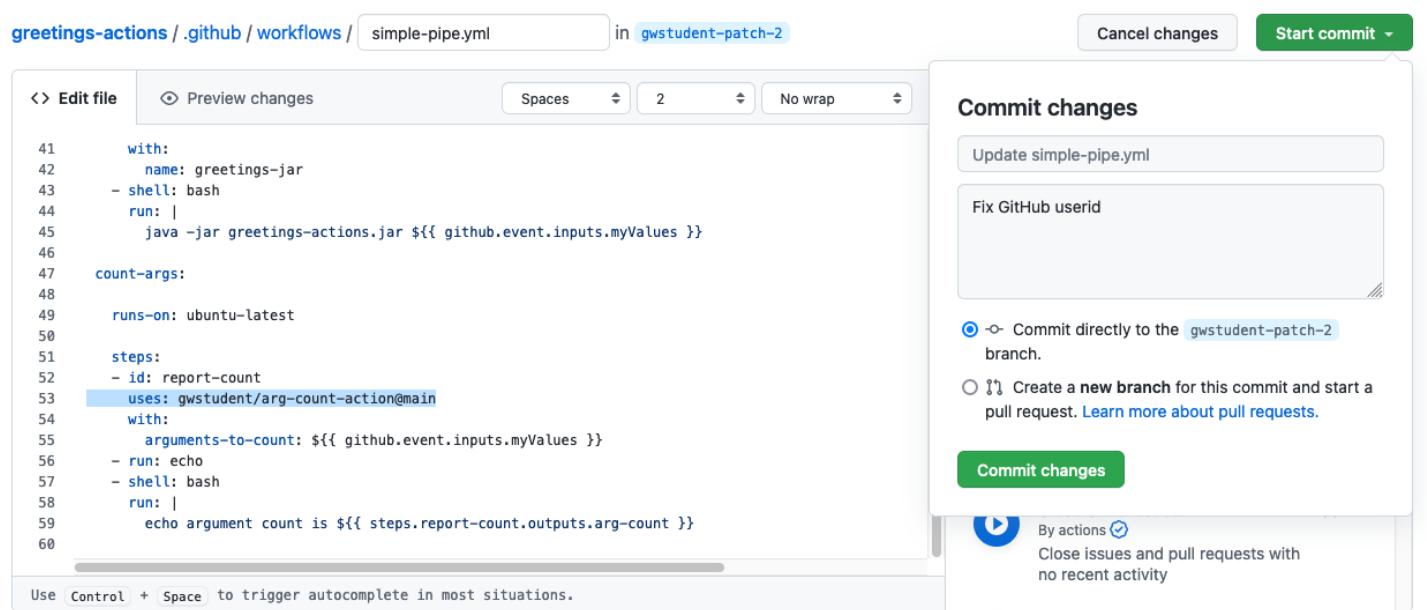
9. Edit the simple-pipe.yaml file (use the pencil icon). Then update the line that has "uses :<your github userid>/arg-count-action@main" to actually have your GitHub userid in it.

```

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 "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

```

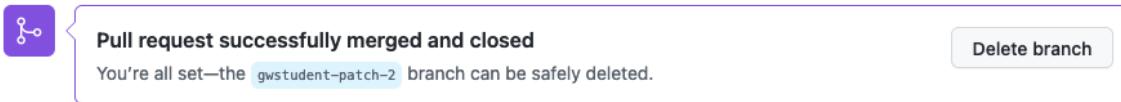
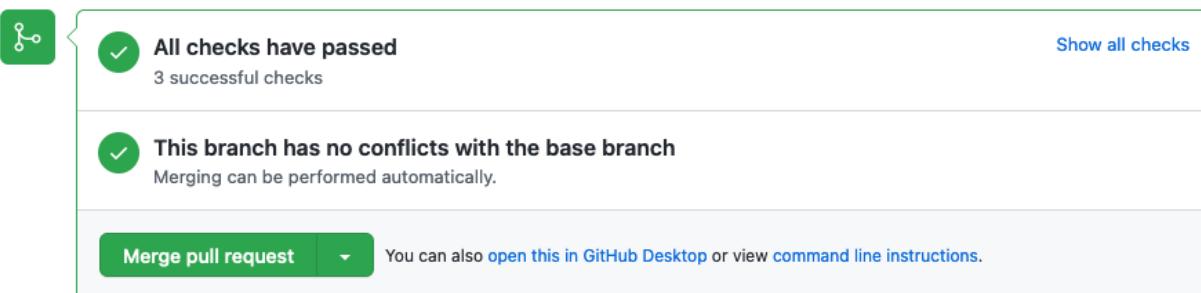
41 with:
42   name: greetings-jar
43 - shell: bash
44   run: |
45     java -jar greetings-actions.jar ${{ github.event.inputs.myValues }}
46
47 count-args:
48
49 runs-on: ubuntu-latest
50
51 steps:
52 - id: report-count
53   uses: gwstudent/arg-count-action@main
54   with:
55     arguments-to-count: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

```

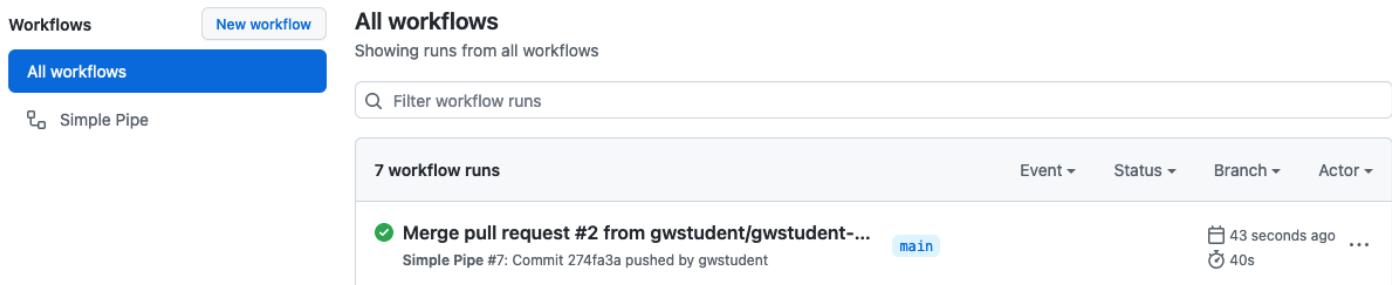
Use **Control + Space** to trigger autocomplete in most situations.

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.

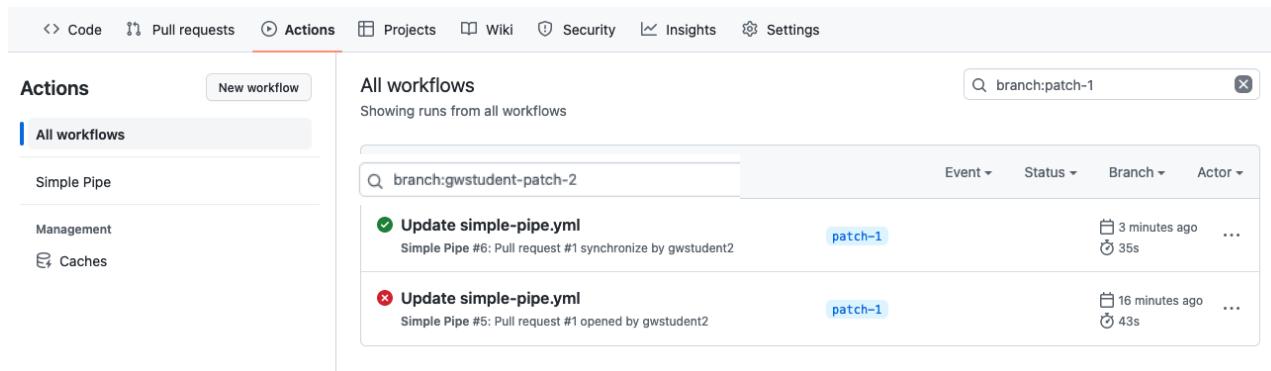


The screenshot shows the GitHub Actions interface. The top navigation bar has 'Workflows' and 'New workflow' buttons. A blue bar highlights 'All workflows'. Below it, a search bar contains 'Simple Pipe'. The main area is titled 'All workflows' with the sub-instruction 'Showing runs from all workflows'. A search bar 'Filter workflow runs' is present. A table lists '7 workflow runs'. The first run is for 'Merge pull request #2 from gwstudent/gwstudent-...' on the 'main' branch, completed 43 seconds ago with a status of '40s'. The table includes columns for Event, Status, Branch, and Actor.

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.

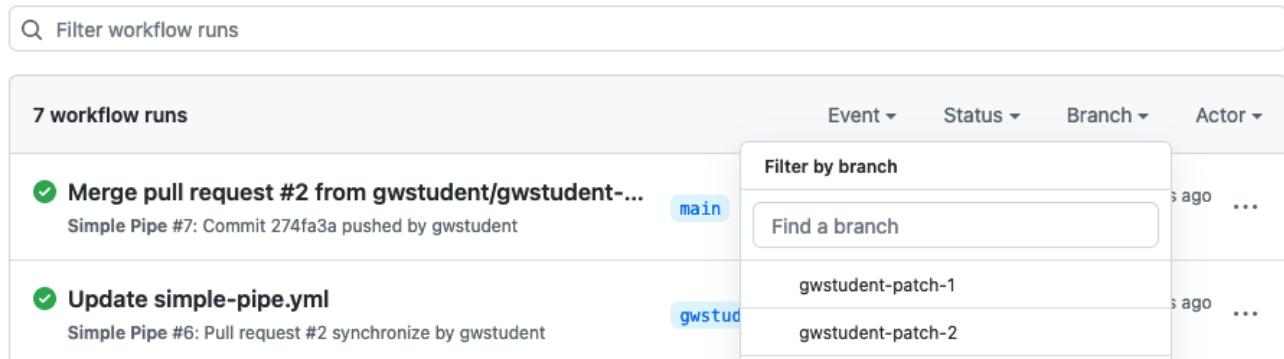


The screenshot shows the GitHub Actions interface with the 'Actions' tab selected. The top navigation bar includes 'Code', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. A search bar on the right contains 'branch:patch-1'. The main area is titled 'All workflows' with the sub-instruction 'Showing runs from all workflows'. A search bar 'branch:gwstudent-patch-2' is present. A table lists workflow runs. The first run is for 'Update simple-pipe.yml' on 'patch-1' branch, completed 3 minutes ago with a status of '35s'. The second run is for 'Update simple-pipe.yml' on 'patch-1' branch, completed 16 minutes ago with a status of '43s'. The table includes columns for Event, Status, Branch, and Actor.

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

Event	Status	Branch	Actor
Merge pull request #2 from gwstudent/gwstudent...	main	Simple Pipe #7: Commit 274fa3a pushed by gwstudent	gwstudent
Update simple-pipe.yml	gwstud	Simple Pipe #6: Pull request #2 synchronize by gwstudent	gwstudent

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

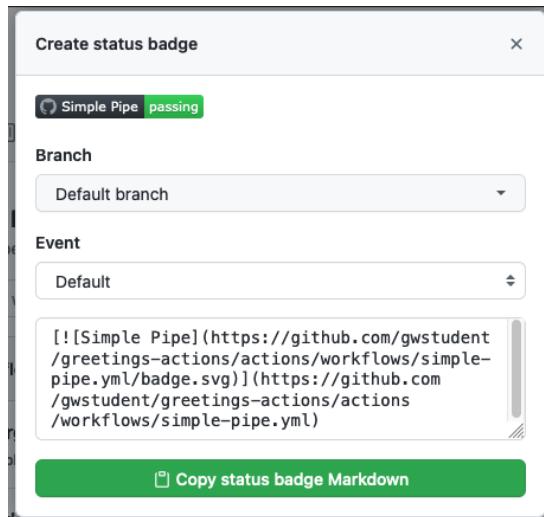
Simple Pipe

simple-pipe.yml

7 workflow runs

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

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



Create status badge

Simple Pipe passing

Branch

Default branch

Event

Default

```
[! [Simple Pipe](https://github.com/gwstudent/greetings-actions/actions/workflows/simple-pipe.yml/badge.svg)](https://github.com/gwstudent/greetings-actions/actions/workflows/simple-pipe.yml)
```

Copy status badge Markdown

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](#)

...

 gwstudent Create README.md ✓

Latest commit 33e7dd4 11 hours ago [History](#)

1 contributor

3 lines (3 sloc) | 294 Bytes

[Raw](#) [Blame](#)   

greetings-actions

Simple hello world type of program for use with learning GitHub Actions  Java CI w/ Gradle and test-run passing

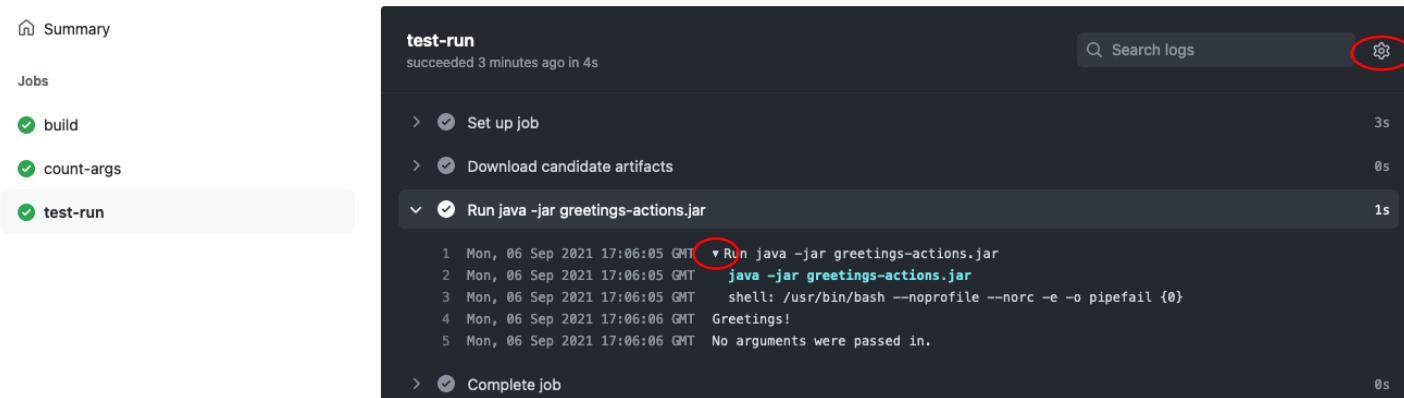
7. Click back on the Actions tab. Click on the name of the first run in the Workflow runs list (probably the one from adding the README). Notice that we have information in the large bar at the top about who initiated the change, the SHA1, the status, duration, and number of artifacts.
8. In the main 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.

Create README.md Simple Pipe #8

Re-run jobs ...



Summary
Jobs
build
count-args
test-run

test-run
succeeded 3 minutes ago in 4s

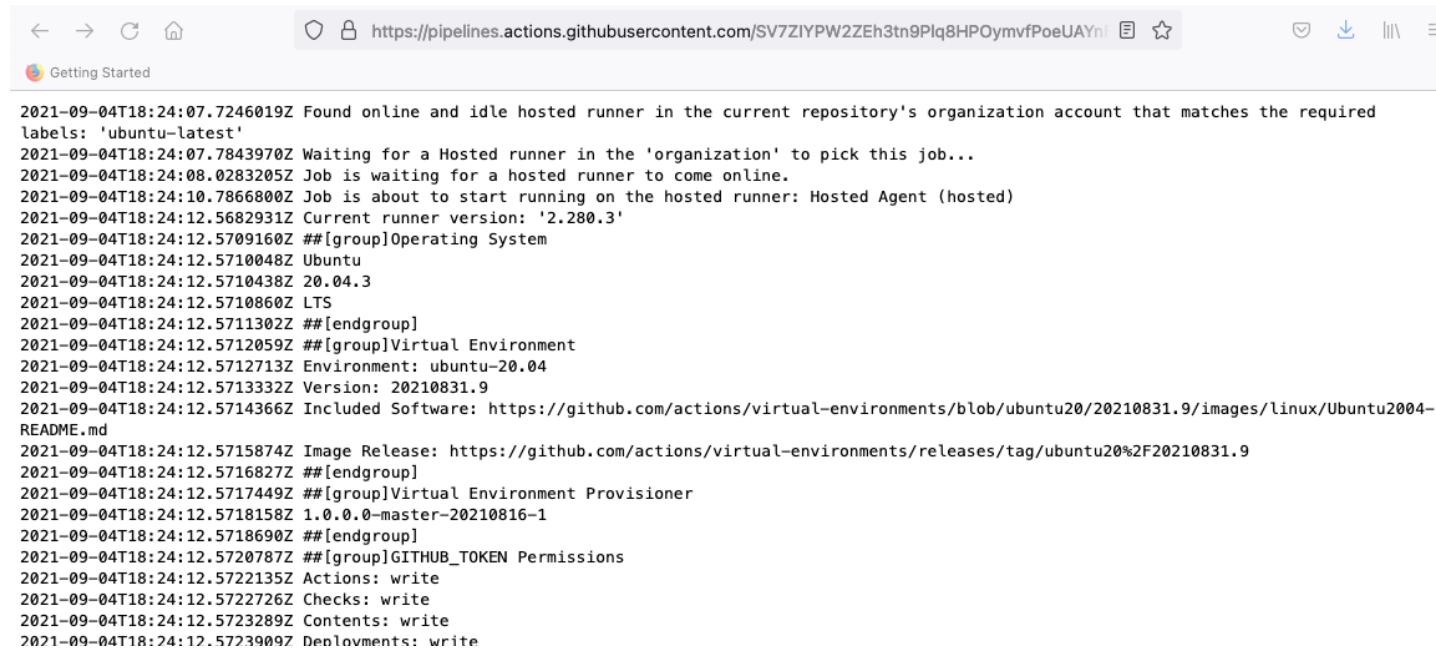
Set up job 3s
Download candidate artifacts 0s
Run java -jar greetings-actions.jar 1s

```
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 --norc --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.
```

Complete job 0s

9. We can get links to share to any line. Hover over any of the line numbers and then right-click to Copy Link, Open in a New Tab, or whatever you would like to do.
10. Click on the gear icon again. Notice there is an option to "Download log archive" if we want to get a copy of the logs locally. Or we can get a full view of the raw logs by clicking on the last entry.

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



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 1.0.0.0-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

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" and "push")

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

```
greetings-actions/.github/workflows/simple-pipe.yml in main
Cancel changes Start commit ▾
Edit file Preview changes Spaces 2 No wrap
Marketplace Documentation
Search Marketplace for Actions
Featured Actions
Setup Node.js environment 1.4k
By actions
Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH
Upload a Build Artifact 1.1k
By actions
Upload a build artifact that can be used by subsequent workflow steps
Setup Java JDK 480
By actions
Setup Java JDK
```

```
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:
```

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, 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.

Code Pull requests Actions Projects Wiki Security Insights Settings

Workflows New workflow

All workflows

Simple Pipe

Simple Pipe

Filter workflow runs

9 workflow runs

This workflow has a workflow_dispatch event trigger.

Run workflow

Use workflow from

Branch: main

Input Values

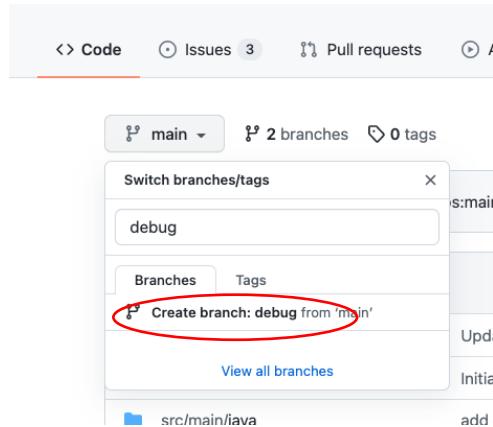
abc def ghi

Run workflow

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.)

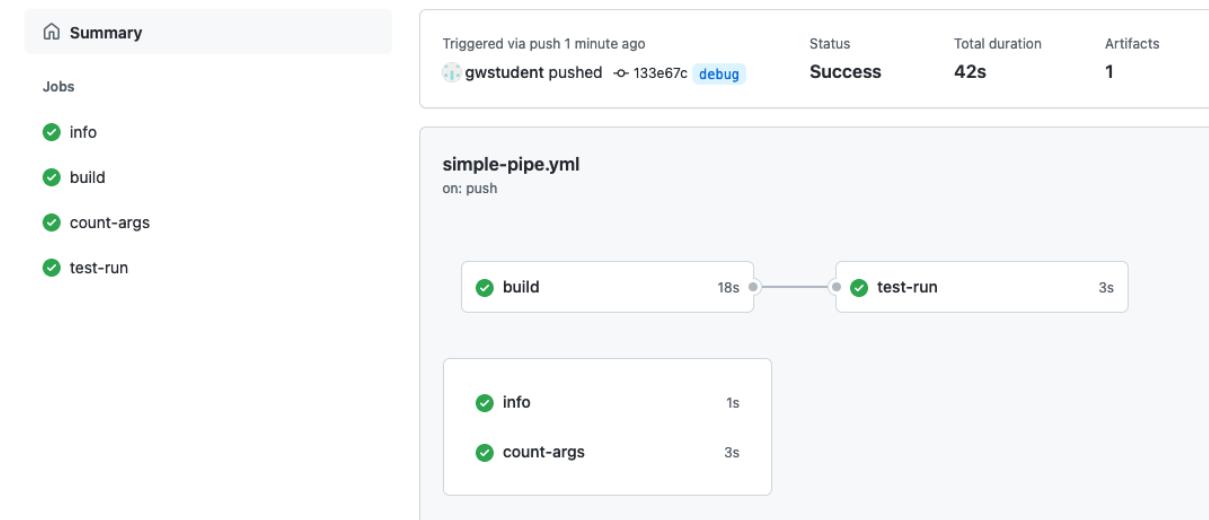
```
info:  
  runs-on: ubuntu-latest  
  
steps:  
  - name: Print warning message  
    run: |  
      echo "::warning::This version is for debugging only."  
  - name: Dump context for runner  
    env:  
      RUNNER_CONTEXT: ${{ toJSON(runner) }}  
    run:  
      echo "::debug::Runner context is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

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.

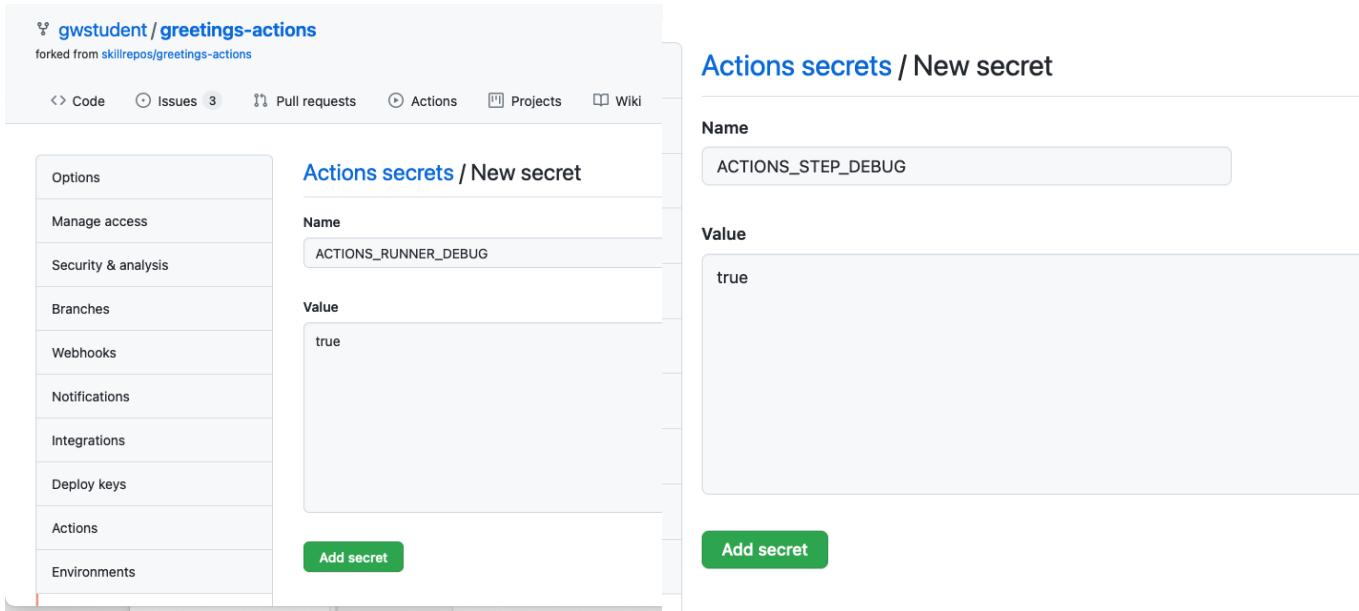
✓ Update simple-pipe.yml Simple Pipe #11



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

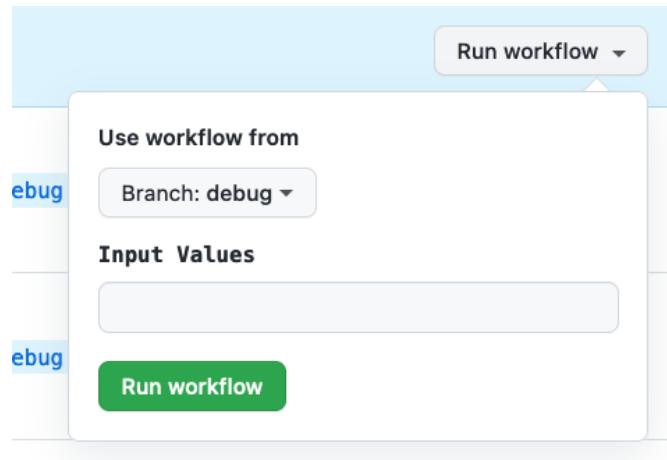
- Notice that while we can see both commands that echo our custom "warning" and "debug" messages - only the output of the warning message actually is displayed, not the output of the debug message. This is because we need to turn on debugging info in our logs. We do that by enabling two secrets - one for `ACTIONS_RUNNER_DEBUG` and one for `ACTIONS_STEP_DEBUG`.
- To turn on these secrets, we need to create and set to "true" the two secrets mentioned above. To do this, go to the repository's top menu and select "Settings". Then on the left-hand side, select "Secrets". 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.

7. Now create each of the new secrets (one named **ACTIONS_RUNNER_DEBUG** and one named **ACTIONS_STEP_DEBUG**) and set their value to "true".

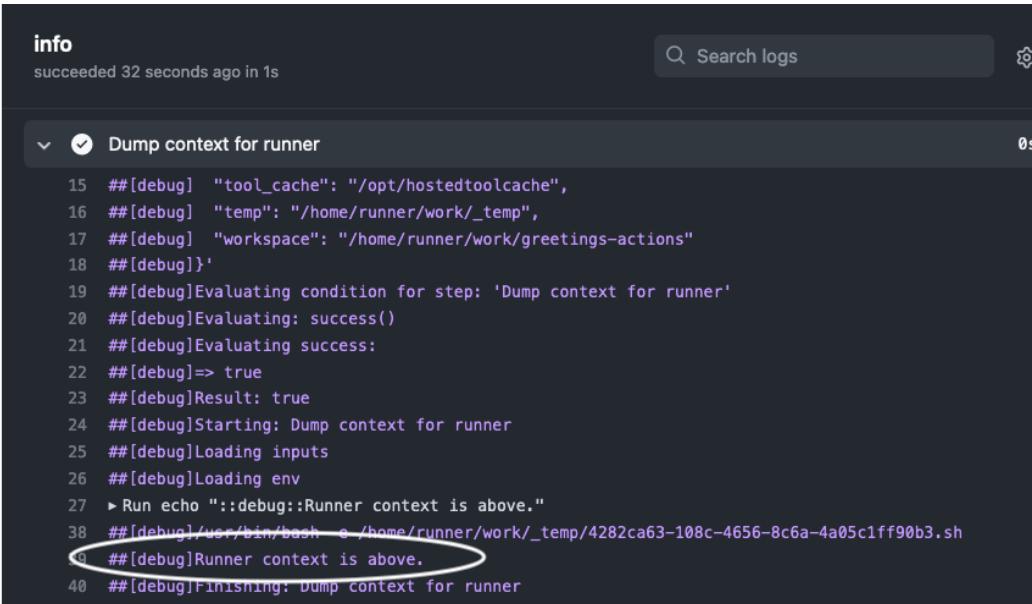


The image shows two side-by-side screenshots of the GitHub Actions secrets creation interface. The left screenshot shows the 'Actions secrets / New secret' page for the 'gwstudent/greetings-actions' repository. It has a sidebar with options like Options, Manage access, Security & analysis, Branches, Webhooks, Notifications, Integrations, Deploy keys, Actions (which is selected), and Environments. The main area shows a secret named 'ACTIONS_RUNNER_DEBUG' with a value of 'true'. A green 'Add secret' button is at the bottom. The right screenshot shows the same interface for a 'New secret', with a 'Name' field containing 'ACTIONS_STEP_DEBUG' and a 'Value' field containing 'true'. A green 'Add secret' button is also present here.

8. 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.



9. 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

```

10. Note that the debug log info will be turned on for all runs from here on - as long as the secrets exist and are 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.

1. 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".)

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. Confirm your password if prompted.
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.

GitHub Apps

OAuth Apps

Personal access tokens

New personal access token

Personal access tokens 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.

Note

workflows

What's this token for?

Expiration *

30 days The token will expire on Tue, Oct 5 2021

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<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
<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry

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

tokens

Generate new token

erated that can be used to access the GitHub API.

your personal access token now. You won't be able to see it again!

tQwNXWU9u2mXYJNVa3Ksu4Iz01G 

5. Now we'll create a new secret and store the PAT value in it. As we did in the earlier lab , go back to the repository (not your personal account) and in the top menu and select "Settings". Then on the left-hand side, select "Secrets" and then "Actions". Now, click on the "New repository secret in the upper right to create a new secret for the action to use.

Actions secrets

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Actions

Webhooks

Environments

Pages

Security

Code security and analysis

Deploy keys

Secrets

Actions

Dependabot

Actions secrets

Environment secrets

There are no secrets for this repository's environments.

Encrypted environment secrets allow you to store sensitive information, such as access tokens, in your repository environments.

Manage your environments and add environment secrets

Repository secrets

There are no secrets for this repository.

Encrypted secrets allow you to store sensitive information, such as access tokens, in your repository.

- For the Name of the new secret, use WORKFLOW_USE. 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 at the bottom.

Actions secrets / New secret

Name

WORKFLOW_USE

Value

ghp_BvOLiw0ftQwNXWU9u2mXYJNVa

Add secret

Repository secrets

`ACTIONS_RUNNER_DEBUG`

`ACTIONS_STEP_DEBUG`

`WORKFLOW_USE`

- 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 'Features' section of the GitHub Pages settings. It includes checkboxes for 'Wikis', 'Restrict editing to collaborators only', and 'Issues'. The 'Issues' checkbox is checked and circled in red. Below these checkboxes is a section titled 'Get organized with issue templates' with a 'Set up templates' button.

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. To do that, you can clone and move it. Or you can just do 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_dispatch section in the "on" area, which means it can be run manually.
 - ii. It has two inputs - a title and body for the issue.
 - iii. 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 the 'extra/create-failure-issue.yml' file content on GitHub. The file contains the following YAML code:

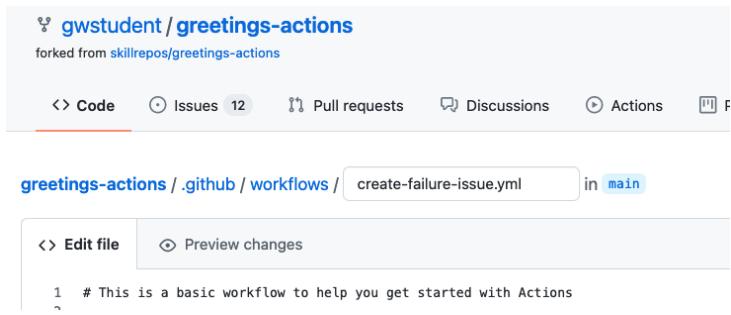
```

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

```

The edit icon in the top right corner of the code editor is circled in red.

- 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".

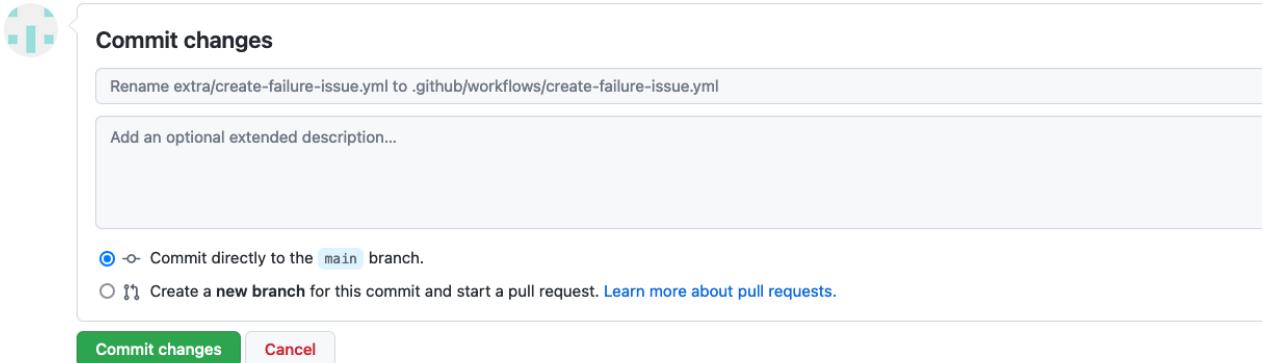


```

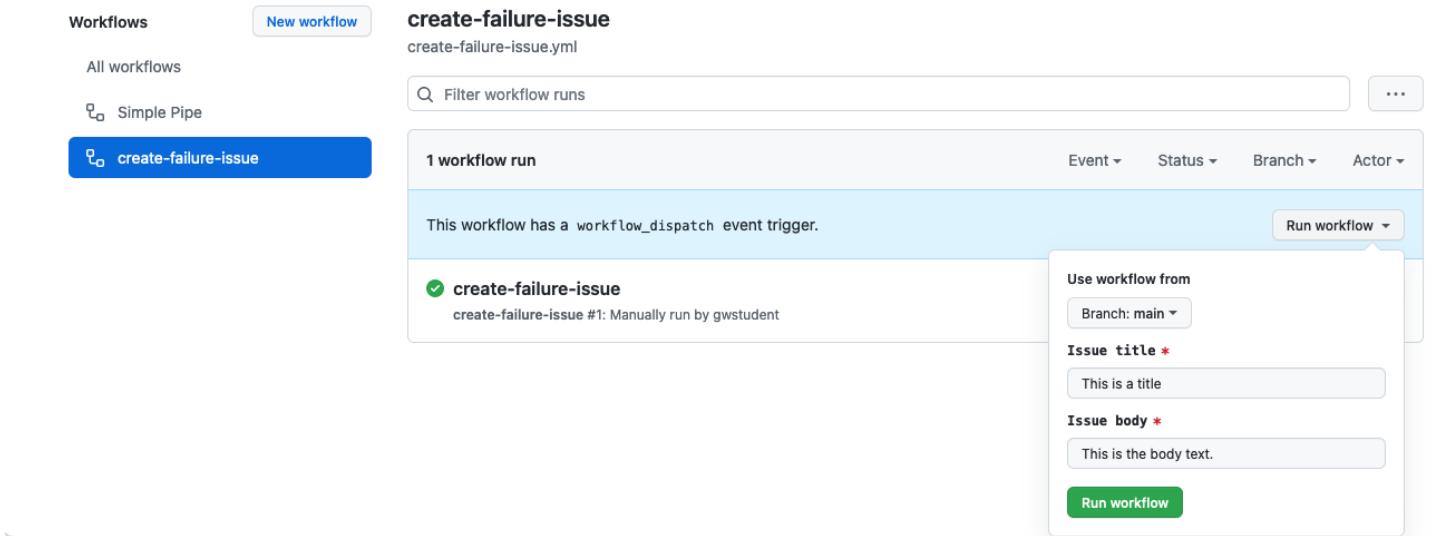
greetings-actions / .github / workflows / create-failure-issue.yml in main
1 # This is a basic workflow to help you get started with Actions

```

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



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".



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.

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.)

```

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/dispatch"
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.

The screenshot shows the GitHub Actions interface. On the left, there's a sidebar with 'Workflows' and a 'New workflow' button. Below that is a 'All workflows' button, which is highlighted in blue, and a list of workflows: 'Simple Pipe' and 'create-failure-issue'. The main area is titled 'All workflows' and shows 'Showing runs from all workflows'. It has a 'Filter workflow runs' input field. Below that is a table titled '15 workflow runs' with columns for 'Event', 'Status', 'Branch', and 'Actor'. The table contains two rows: one for 'create-failure-issue' (status: success, 15 seconds ago, 13s) and one for 'Simple Pipe' (status: failure, 1 minute ago, 49s). The 'Simple Pipe' row shows the commit details: 'Simple Pipe #14: Commit 1539282 pushed by 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 2s"]; B --> C["✓ create-issue-on-failure 1s"]; C --> A; D["✓ build 16s"]
```

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"]
```

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

Automated workflow failure issue for commit 153928267f2fc38a4e91b5bd00d61f033abd59d6 #4

[Open](#) [github-actions](#) (bot) opened this issue 3 minutes ago · 0 comments



github-actions (bot) commented 3 minutes ago

...

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

THE END - THANKS!