

# Introduction to GitHub Actions

Revision 2.1 – 12/3/21

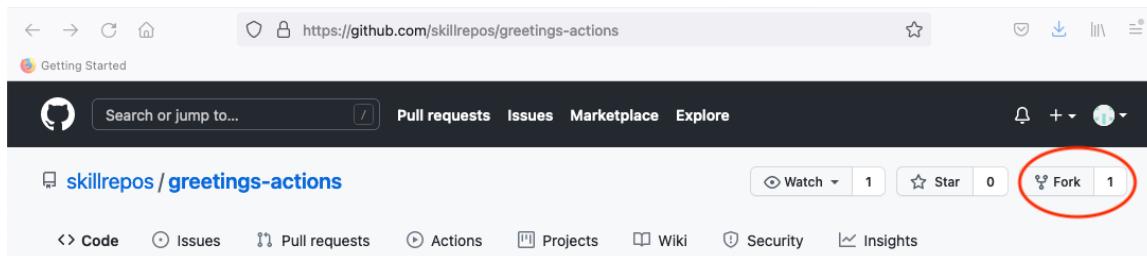
Tech Skills Transformations LLC / Brent Laster

*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 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 'skillrepos/greetings-actions'. At the top, there's a navigation bar with links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the navigation is a search bar and a 'Getting Started' section. The main header shows the repository name 'skillrepos / greetings-actions'. To the right of the repository name are buttons for 'Watch' (1), 'Star' (0), and 'Fork' (1). The 'Fork' button is highlighted with a red circle. Below the header, there are tabs for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', and 'Insights'. The 'Code' tab is selected. On the left, there's a sidebar with a 'main' branch dropdown, a '1 branch' indicator, and a '0 tags' indicator. The main content area displays a list of commits. The first commit is by 'Brent Laster' and is titled 'update files for labs'. It was made 1 hour ago and has 4 commits. Below this, there are other commits for 'extra', 'gradle/wrapper', 'src/main/java', 'gradlew', and 'gradlew.bat', all made 6 days ago. To the right of the commits, there's an 'About' section with a description: 'Simple hello world type of program for use with learning GitHub Actions'. There's also a 'Releases' section indicating 'No releases published'.

3. We have a simple java source file in `greetings-actions/src/main/java/echoMsg.java`, a Gradle build file in `build.gradle` and a simple GitHub Action to build it at `extra/simple-pipe.yml` (<https://github.com/skillrepos/greetings-actions/blob/main/extra/simple-pipe.yml>). Take a look at that file by selecting it in the browser.

skillrepos / greetings-actions

Code Issues Pull requests Actions Projects Wiki Security Insights

main greetings-actions / extra / simple-pipe.yml

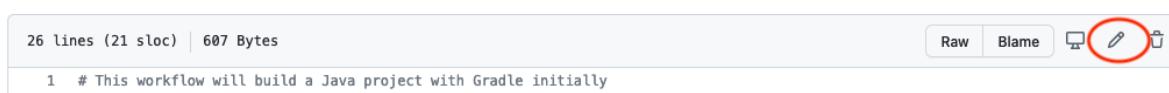
Brent Laster update files for labs Latest commit c175a1c 2 hours ago History

0 contributors

26 lines (21 sloc) | 607 Bytes Raw Blame

```
1 # This workflow will build a Java project with Gradle initially
```

- To make this a workflow that can be executed, we need to put this file in a subdirectory named `.github/workflows` in the current project. First, click on the pencil icon to edit the file.



- Next in the path/name area near the top, click in the text box, backspace over the "extra/" directory and enter `.github/workflows`. Afterwards, the path should look like the figure below. Once you are done, scroll down to the bottom and click the green "Commit changes" button. You can commit them directly to the main branch.

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

Edit file Preview changes

**Commit changes**

Rename extra/simple-pipe.yml to .github/workflows/simple-pipe.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

6. Now, click on the "Actions" tab at the top of the project and you'll be able to see the new workflow being run.

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 this, a sidebar on the left lists 'Workflows' and 'New workflow'. A blue button labeled 'All workflows' is selected. Under 'All workflows', there's a section for 'Simple Pipe' with a single workflow run listed. The run details are: '1 workflow run', 'Event: Push', 'Status: In progress', 'Branch: main', and 'Actor: gwstudent'. The commit message is 'Rename extra/simple-pipe.yml to .github/w...'. A timestamp indicates it was 23 seconds ago.

This occurred because we did a push to main (the commit) and we had this section in the gradle.yml file that described our workflow.

```
6  on:
7    push:
8      branches: [ main ]
9    pull_request:
10   branches: [ main ]
```

7. Shortly, the workflow will have completed and should have completed successfully. That will be indicated by a green circle with a check mark in it. Click on the commit message next to the circle "Rename extra/simple-pipe.yml to..." to view more about the jobs in the workflow. You should see a screen similar to the second one below.

The screenshot shows the '1 workflow run' details for the 'Simple Pipe' workflow. The run is successful ('Status: Success') and completed in 24s. The commit message is 'Rename extra/simple-pipe.yml to .github/w...'. A green checkmark icon is next to the commit message, indicating success. The 'Event' dropdown is set to 'Push'.

The screenshot shows the workflow summary for 'Simple Pipe #1'. It displays the 'simple-pipe.yml' job, which was triggered via a push 8 minutes ago by gwstudent. The job status is 'Success' with a total duration of 24s. The job details show a single step named 'build' that completed successfully in 13s.

8. Click on either of the "build" links with the green circle checkmark next to it to see details for the single job in our workflow.

The screenshot shows a GitHub Actions workflow named "Rename extra/simple-pipe.yml to .github/workflows/simple-pipe.yml Simple Pipe #1". It was triggered via push 9 minutes ago by gwstudent pushed to main branch. The status is Success with a total duration of 24s. There are no artifacts. A single job named "simple-pipe.yml" is listed, which runs on push. It contains one step named "build" with a green checkmark icon, indicating success, and a duration of 13s.

9. From the next screen you can see the set of steps in the build job. Click on any of them to expand and see the logs from execution of that step.

The screenshot shows the detailed view of the "build" step from the previous workflow. The step succeeded 11 minutes ago in 13s. The logs show the following sequence of events:

- > Set up job
- > Run actions/checkout@v2
- > Set up JDK 1.8
- > Grant execute permission for gradlew
- > Build with Gradle

Under the "Build with Gradle" section, the logs show the command "Run ./gradlew build" and the output of the Gradle 4.10 welcome message:

```

1  Run ./gradlew build
8  Downloading https://services.gradle.org/distributions/gradle-4.10
9  .....
10
11  Welcome to Gradle 4.10!
12
13  Here are the highlights of this release:
14  - Incremental Java compilation by default
15  - Periodic Gradle caches cleanup
16  - Gradle Kotlin DSL 1.0-RC3
17  - Nested included builds
18  - SNAPSHOT plugin versions in the `plugins {}` block
19
20  For more details see https://docs.gradle.org/4.10/release-notes.html
21

```

10. You can also get a link to any line in the log. Hover over a line number and then right-click to copy the link, open it in a new browser, etc.

## Lab 2 – Learning more about Actions

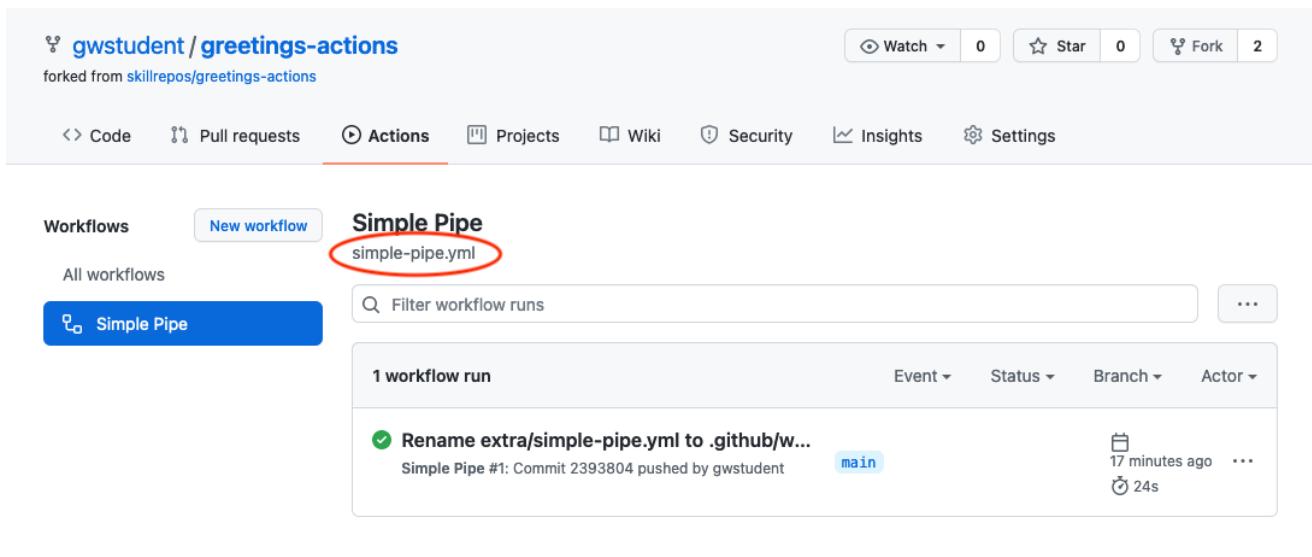
**Purpose:** In this lab, we'll see how to get more information about Actions and how to update our workflow to use others.

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 gradle.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 page for 'gwstudent/greetings-actions'. The 'Actions' tab is selected. Under 'Workflows', the 'Simple Pipe' workflow is selected, indicated by a blue bar. The 'simple-pipe.yml' file is highlighted with a red circle. A list of workflow runs is shown, with one run for 'Rename extra/simple-pipe.yml to .github/w...' completed 17 minutes ago.

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



The screenshot shows the GitHub file editor for 'simple-pipe.yml'. The file content is displayed, and the top right corner has a red circle around the edit icon (pencil).

```
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 ]
```

3. You'll now see the file open up in the editor, but also to the right, you should see a new pane with references to GitHub actions. We're going to add a job to our workflow to upload an artifact. Let's find actions related to uploading.

In the "Search Marketplace for Actions" box on the upper right, enter "Upload" and see what's returned.

Next, click on the "Upload a Build Artifact" item. Take a look at the page that comes up from that. Let's look at the full listing on the Actions Marketplace. Click on the "View full Marketplace listing".

**Marketplace / Search results**

- Veracode Upload And Scan** By veracode (13) ★ 13  
Upload files to veracode and start a static scan
- Cloud Storage Uploader** By google-github-actions (51) ★ 51  
Upload files or folders to GCS buckets
- 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  
Run tfsec against terraform code base and upload the sarif output to the github repo
- twine-upload** By yaananth (1) ★ 1  
Upload to twine

**Marketplace / Search results / Upload a Build Artifact**

**Upload a Build Artifact**  
By actions (1.1k) v2.2.4 ★ 1.1k

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: v2.2.4
- name: Upload a Build Artifact
  uses: actions/upload-artifact@v2.2.4
  with:
    # Artifact name
    name: # optional, default is artifact
    # A file, directory or wildcard pattern that describes the path:
    path: # The desired behavior if no files are found using this path:
    # Available Options:
    warn: Output a warning but do not fail the action
    error: Fail the action with an error message
    ignore: Do not output any warnings or errors, the action will continue even if there are errors
```

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

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

**GitHub Action Checkout**  
v2.3.4 (Latest version)

[Use latest version](#)

**Checkout V2**

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.

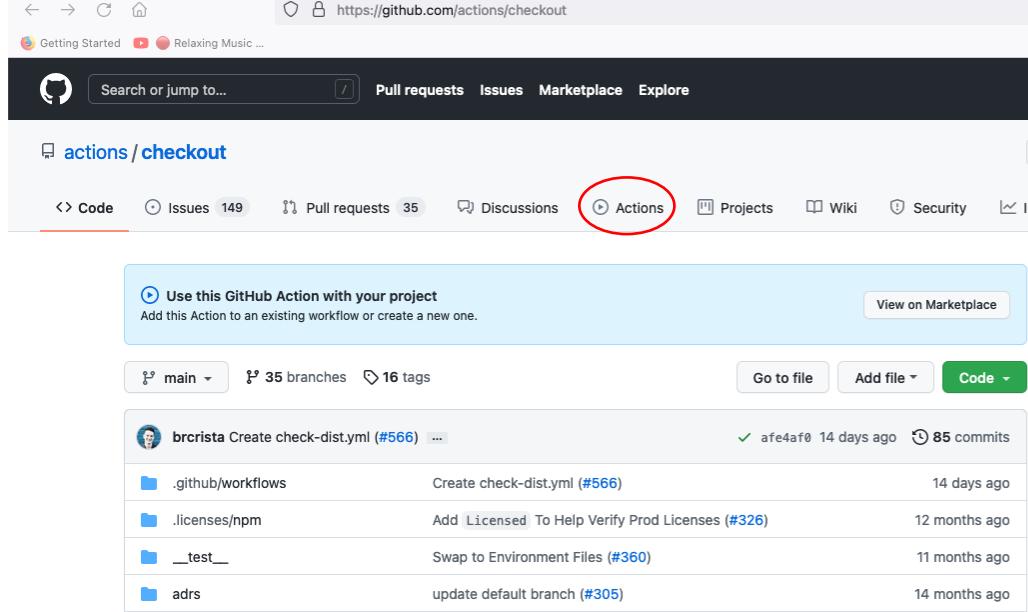
**What's new**

- Improved performance
  - Fetches only a single commit by default

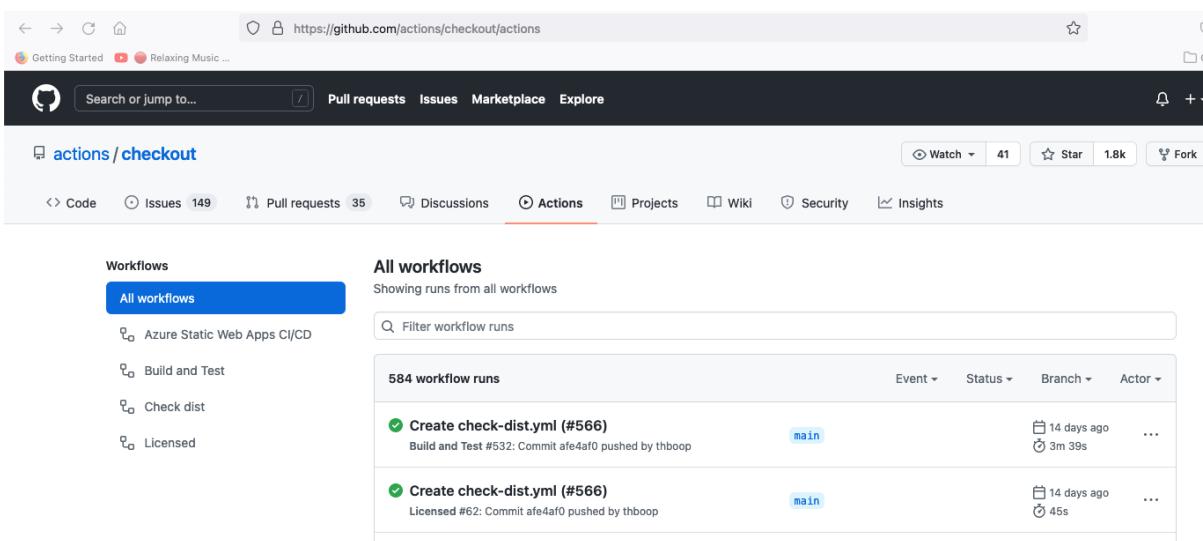
**Links**

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

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



The screenshot shows the GitHub Actions checkout repository page. The Actions tab is highlighted with a red circle. The page displays a list of branches (main, 35 branches, 16 tags), a commit history, and sections for About, Releases, and Insights.

The screenshot shows the GitHub Actions checkout repository page with the Actions tab selected. The left sidebar shows 'Workflows' with 'All workflows' selected. The main area shows 'All workflows' with 584 workflow runs listed, each with details like event, status, branch, and actor.

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 32-36) for how this should look afterwards. (Your line numbers may be different.)

```
- name: Upload Artifact
  uses: actions/upload-artifact@v2
  with:
    name: greetings-jar
    path: build/libs
```

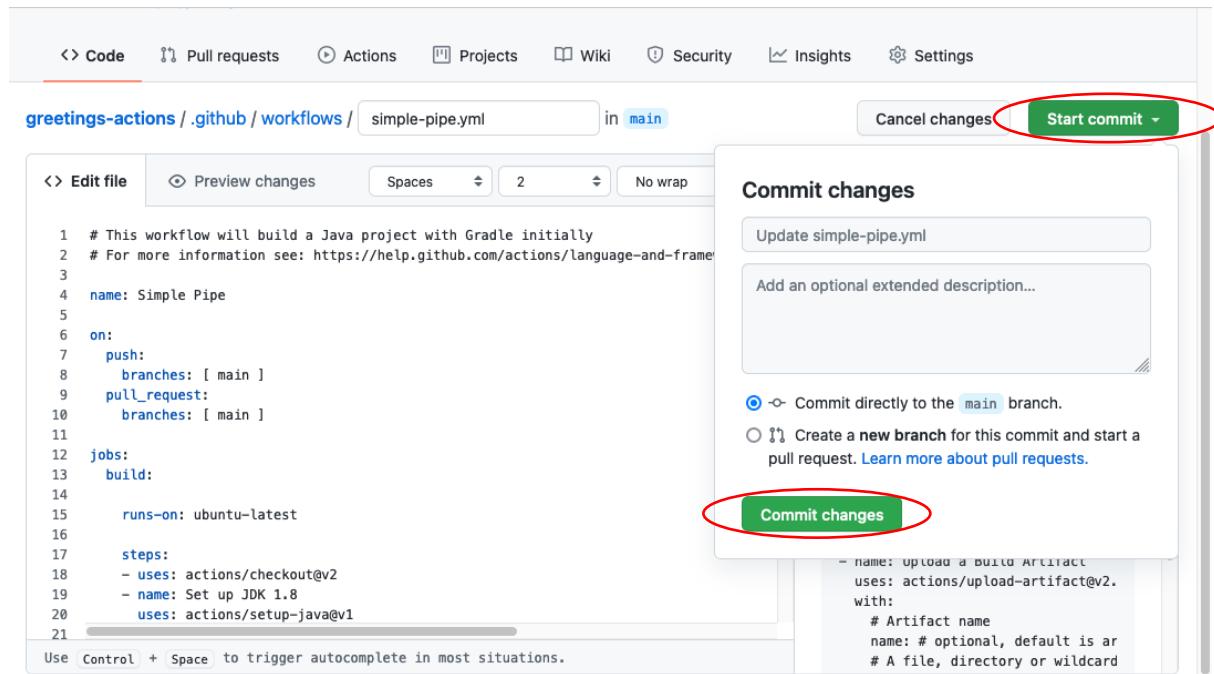
<> Edit file <> Preview changes

```

18   build:
19
20     runs-on: ubuntu-latest
21
22     steps:
23       - uses: actions/checkout@v2
24       - name: Set up JDK 1.8
25         uses: actions/setup-java@v1
26         with:
27           java-version: 1.8
28       - name: Grant execute permission for gradlew
29         run: chmod +x gradlew
30       - name: Build with Gradle
31         run: ./gradlew build
32       - name: Upload Artifact
33         uses: actions/upload-artifact@v2
34         with:
35           name: greetings-jar
36           path: build/libs
37

```

- 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" at the bottom. You can download the artifact from there. Click on the name of the artifact to try this.

The screenshot shows a GitHub Actions run summary for a workflow named 'simple-pipe.yml'. The run was triggered via push 2 minutes ago by gwstudent pushed to branch main. The status is Success, total duration is 26s, and there is 1 artifact. The 'simple-pipe.yml' file has a single job named 'build' which completed successfully in 13s. The 'Artifacts' section shows a single artifact named 'greetings-jar' produced during runtime, which is a Java jar file (1006 Bytes). The artifact name is circled in red.

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

#### **test-run:**

```

runs-on: ubuntu-latest
needs: build

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

```

**Edit file** Preview changes Spaces 2 No wrap

```

26      run: ./gradlew build
27      - name: Upload Artifact
28          uses: actions/upload-artifact@v2
29          with:
30              name: greetings-jar
31              path: build/libs
32
33      test-run:
34
35      runs-on: ubuntu-latest
36      needs: build
37
38      steps:
39          - name: Download candidate artifacts
40              uses: actions/download-artifact@v2
41              with:
42                  name: greetings-jar
43                  shell: bash
44                  run: |
45                      java -jar greetings-actions.jar ${{ github.event.inputs.myValues }}
```

- 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.

Code Pull requests Actions Projects Wiki Security Insights Settings

**Update simple-pipe.yml Simple Pipe #3** Cancel workflow ...

**Summary**

Triggered via push 17 seconds ago gwstudent pushed → 4c12b4e main Status In progress Total duration — Artifacts —

**Jobs**

**build**

**simple-pipe.yml**  
on: push

**build** 7s → **test-run**

## Lab 3: Adding your own action

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

1. First, we'll fork the repo for a simple action that displays a count of the arguments passed in to 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 page for 'skillrepos/arg-count-action'. At the top right, there is a 'Fork' button with a value of '1'. Below the header, there are tabs for 'Code', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. A prominent blue banner at the top says 'Publish this Action to Marketplace'. On the right side, there's an 'About' section with the text 'Simple GitHub Action demo'. Below it is a 'Releases' section showing 8 tags and a link to 'Create a new release'. Under 'Packages', it says 'No packages published'. The main content area shows a list of files: 'action.yml' (updated 2 days ago), 'count-args.sh' (initial commit 3 days ago), and a commit from 'techupskills' (Delete iterate-args.sh, 1 hour ago). There are also buttons for 'Go to file', 'Add file', and 'Code'.

2. In your fork of the repository, take a 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 }}

```

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

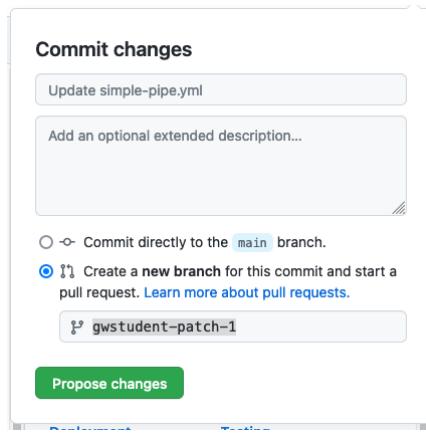
```

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

```

In this case, we call our custom action (<your github repo/arg-count-action>), using the latest from the main branch.

- Let's use a pull request to merge this change. Click on the green "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, update the comment if you want and then click on the "Create pull request" button. You'll then see it run through the jobs in our workflow as prechecks for merging.

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

**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](#).

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.

[Code](#) [Pull requests 1](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

**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**

**Summary**

Triggered via pull request 14 minutes ago Status Failure Total duration 45s Artifacts 1

Jobs

- ✓ build
- ✗ count-args
- ✓ test-run

**simple-pipe.yml**  
on: pull\_request

```

graph LR
    build[build] -- "14s" --> testRun[test-run]
    testRun -- "5s" --> countArgs[count-args]
    countArgs -- "33s" --> end(( ))

```

**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 and **switch to the patch branch that you created for the pull request.**

forked from [skillrepos/greetings-actions](#)

Code Pull requests 1 Actions Projects Wiki Security Insights

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

Switch branches/tags

Branches Tags

✓ main (default)

brentlaster-patch-1

project with Gradle initially  
://help.github.com/actions/language-and-framework-guides/b

```

3
4 name: Simple Pipe
5
6 on:
7   push:
8     branches: [ main ]

```

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.

```

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".

The screenshot shows the GitHub interface for committing changes. On the left, there is a code editor for the file `simple-pipe.yml`. The code contains two workflow definitions: `with:` and `count-args:`. The `count-args:` section is highlighted with a blue background. On the right, a modal window titled "Commit changes" is open. It has a text input field containing "Update simple-pipe.yml" and another field containing "Fix GitHub userid". Below these fields are two radio buttons: one selected (blue outline) labeled "Commit directly to the gwstudent-patch-2 branch." and one unselected (gray outline) labeled "Create a new branch for this commit and start a pull request." A large green "Commit changes" button is at the bottom of the modal. At the very bottom of the page, there is a note: "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](#).

The screenshot shows the GitHub Actions check results for the `gwstudent-patch-2` branch. It includes a summary of successful checks, a note about no conflicts with the base branch, and a "Merge pull request" button.

**All checks have passed**  
3 successful checks

**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](#).

The screenshot shows the "Merge pull request" dialog. It lists the changes being merged: "Merge pull request #2 from gwstudent/gwstudent-patch-2" and "Update simple-pipe.yml". At the bottom are "Confirm merge" and "Cancel" buttons.

The screenshot shows the confirmation dialog after the pull request was merged. It says "Pull request successfully merged and closed" and "You're all set—the gwstudent-patch-2 branch can be safely deleted." A "Delete branch" button is visible.

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 "All workflows" page. It displays 7 workflow runs for the "Simple Pipe" workflow. One run is highlighted: "Merge pull request #2 from gwstudent/gwstudent-..." with status "main", completed 43 seconds ago, and took 40s.

**All workflows**  
Showing runs from all workflows

**7 workflow runs**

Event	Status	Branch	Actor
✓ Merge pull request #2 from gwstudent/gwstudent-...	main	43 seconds ago	...
Simple Pipe #7: Commit 274fa3a pushed by gwstudent		40s	

## Lab 4: Exploring logs

**Purpose:** In this lab, we'll take a closer look at the different options for getting information from logs.

1. If not already there, switch back to the Actions tab. To the right of the list of workflows is a search box. Let's execute a simple search - note that only certain keywords are provided and not a complete log search. Let's search for the workflow runs that were done for the branch that you used for the Pull Request in the last lab. Enter "**branch:<patch-branch-name>**" (no spaces) in the search box and hit enter.

3 workflow run results			
Event	Status	Branch	Actor
<span style="color: green;">✓</span> Update simple-pipe.yml	Success	gwstudent-patch-2	22 minutes ago
<span style="color: red;">✗</span> Update simple-pipe.yml	Failure	gwstudent-patch-2	1 hour ago

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

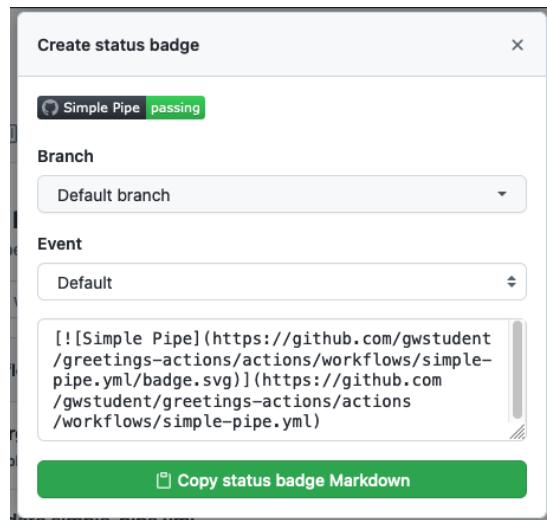
7 workflow runs			
Event	Status	Branch	Actor
<span style="color: green;">✓</span> Merge pull request #2 from gwstudent/gwstudent-...	Success	main	3 ago
<span style="color: green;">✓</span> Update simple-pipe.yml	Success	gwstud	3 ago

Filter by branch  
 main  
 Find a branch  
 gwstudent-patch-1  
 gwstudent-patch-2

3. Select the "Simple Pipe" workflow. You'll now have a box with "..." beside the search box that has some additional options. Click on the "..." beside the search box to see some them. They include disabling the workflow and setting up a "badge" for the workflow. Let's go ahead and set up a badge now to show success/failure for running the workflow. Click on the entry for "Create status badge".

The screenshot shows the GitHub Workflows interface for a repository named 'simple-pipe.yml'. At the top, there's a 'New workflow' button. Below it, a search bar says 'Filter workflow runs'. The main area shows '7 workflow runs' for the 'Simple Pipe' workflow. One run is highlighted with a green checkmark and the text 'Merge pull request #2 from gwstudent/gwstudent...'. In the top right, there are buttons for 'Event', 'Status', 'Create status badge' (which is circled in red), 'Disable workflow', and a timestamp '22 minutes ago'.

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.

The screenshot shows the GitHub README editor. At the top, it says 'Help people interested in this repository understand your project by adding a README.' and has a green 'Add a README' button. Below is a text editor for 'greetings-actions / README.md' in the 'main' branch. The editor shows the following code:

```
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/actions/workflows/simple-pipe.yml)
```

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

A screenshot of a GitHub repository page for 'greetings-actions / README.md'. At the top, there's a 'main' dropdown, the repository name, and a 'Go to file' button. Below the header, it shows a green checkmark icon next to 'gwstudent Create README.md ✓'. To the right, it says 'Latest commit 33e7dd4 11 hours ago' and has a 'History' link. Underneath, it says '1 contributor'. At the bottom, it shows '3 lines (3 sloc) | 294 Bytes' and buttons for 'Raw', 'Blame', and icons for copy/paste.

## greetings-actions

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

- 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.

Then click on the box with the 3 dots in the upper right. Here we have options to "View workflow file" (look at the actual yaml file for the workflow), "View workflow runs" (show the runs as the previous screen) and "Create status badge" (what we did in the preceding steps) and delete the logs.

A screenshot of the GitHub Actions interface for a workflow named 'Create README.md Simple Pipe #8'. The 'Actions' tab is selected. In the main area, there's a 'Summary' section with a green checkmark icon and the text 'Triggered via push 2 minutes ago' followed by 'gwstudent pushed → 05235e3 main'. To the right, it shows 'Status Success', 'Total duration 41s', and 'Artifacts 1'. A context menu is open over this summary bar, listing 'Re-run jobs', 'View workflow file', 'View workflow runs', 'Create status badge', and 'Delete all logs'. Below the summary, there's a 'simple-pipe.yml' section with 'on: push' and a job graph. The graph shows three jobs: 'build', 'test-run', and another unnamed job. The 'build' job took 16s and has an arrow pointing to the unnamed job, which then points to the 'test-run' job, which took 4s. The unnamed job is shown with a timestamp of '16s'.

- 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

Search logs ⚙

3s

0s

1s

> Set up job

> Download candidate artifacts

> Run java -jar greetings-actions.jar

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.

> Complete job 0s

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

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

Getting Started

https://pipelines.actions.githubusercontent.com/SV7ZIYPW2Eh3tn9Plq8HPOymvfPoeUAYn... 🌐 ⭐

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

- 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

```

```

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:

```

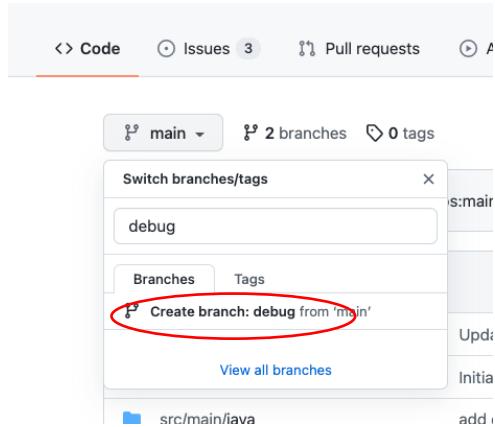
13. Commit your changes to the main branch. Since we added the additional "on" section, if you look at the Actions page and select the workflow, you'll see that after the merge we now have a note that "This workflow has a workflow\_dispatch event trigger." And there is a "Run workflow" button that we 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.

Workflow Run	Event	Status	Branch	Actor
Update simple-pipe.yml	Event	Success	main	gwstudent
Create README.md	Event	Success	main	gwstudent
Merge pull request #2 from gwstudent/gwstudent-p...	Event	Pending	main	gwstudent

## 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 yaml file. **Change the references to "main" in the "on" section at the top to "debug".** Also, add a new job to surface some debug context. Add in the lines below after the "jobs:" line. Pay attention to indenting again. A screenshot of how everything should look and lines up is further down. (For convenience, the text for the info job is also in a file in extra/info.txt.)

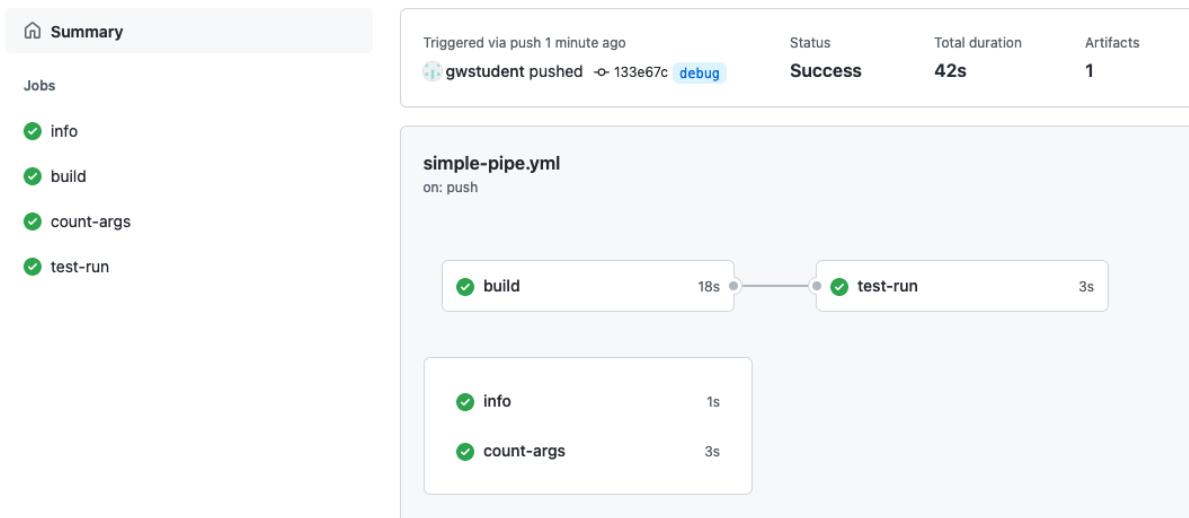
```
info:  
  runs-on: ubuntu-latest  
  
steps:  
  - name: Print warning message  
    run:  
      echo "::warning::This version is for debugging only."  
  - name: Dump context for runner  
    env:  
      RUNNER_CONTEXT: ${{ toJSON(runner) }}  
    run:  
      echo "::debug::Runner context is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

```
< > Edit file | Preview changes

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

- 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 take a look at the logs.

 Update simple-pipe.yml Simple Pipe #11



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

The screenshot shows the GitHub Actions pipeline interface. On the left, there's a sidebar with a 'Summary' section and a list of jobs: 'info' (green checkmark), 'build', 'count-args', and 'test-run'. The 'info' job is expanded, showing its steps: 'Set up job' (0s), 'Print warning message' (1s), 'Dump context for runner' (0s), and 'Complete job' (0s). The 'Print warning message' step contains the following log output:

```

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

```

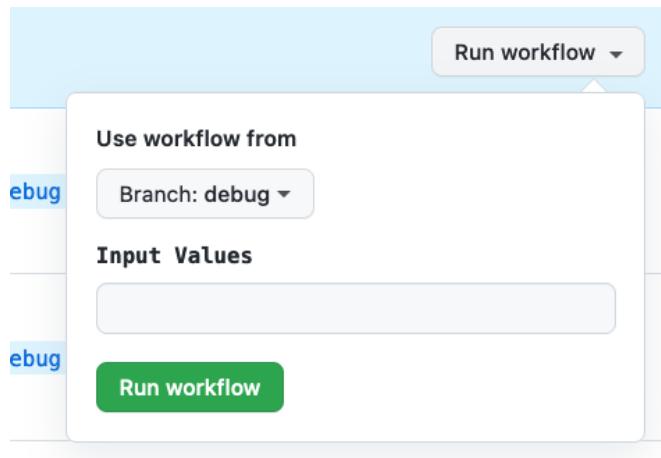
5. Notice that while we can see both commands that echo our custom "warning" and "debug" messages - only the output of the warning message actually is displayed, not the output of the debug message. This is because we need to turn on debugging info in our logs. We do that by enabling two secrets - one for `ACTIONS_RUNNER_DEBUG` and one for `ACTIONS_STEP_DEBUG`.
6. 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". 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 'gwstudent / greetings-actions'. At the top, there are buttons for 'Watch' (0), 'Star' (0), 'Fork' (2), and 'Settings' (1, circled in red). Below the navigation bar, there's a sidebar with links: Options, Manage access, Security & analysis, Branches, Webhooks, Notifications, Integrations, Deploy keys, Actions, Environments, and Secrets (2, circled in red). The main content area is titled 'Actions secrets' (3, circled in red). It contains information about environment variables and a link to 'New repository secret'. Below this is a section for 'Environment secrets' with a message: 'There are no secrets for this repository's environments.' and a link to 'Manage your environments and add environment secrets'. At the bottom, there's a section for 'Repository secrets' with a similar message: 'There are no secrets for this repository.' and a link to 'Manage your repository secrets'.

7. Now create each of the new secrets and set their value to "true".

The screenshot shows the GitHub Actions secrets creation interface. On the left, there's 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 is titled "Actions secrets / New secret". It has three fields: "Name" (set to "ACTIONS\_STEP\_DEBUG"), "Value" (set to "true"), and a "Add secret" button at the bottom.

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

Dump context for runner
0s

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. Click on "Generate new token". 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.

**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 \***

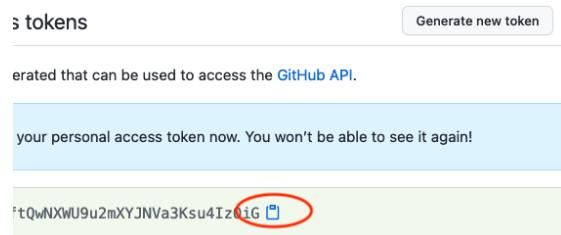
30 days The token will expire on Tue, Oct 5 2021

**Select scopes**

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

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

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



- 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 and in the top menu and select "Settings". Then on the left-hand side, select "Secrets". Now, click on the "New repository secret in the upper right to create a new secret for the action to use.

gwstudent / greetings-actions  
forked from skillrepos/greetings-actions

Code Issues 3 Pull requests Actions Projects Wiki Security Insights Settings

Options Manage access Security & analysis Branches Webhooks Notifications Integrations Deploy keys Actions Environments Secrets

**Actions secrets** New repository secret

Secrets are environment variables that are **encrypted**. Anyone with collaborator access to this repository can use these secrets for Actions.

Secrets are not passed to workflows that are triggered by a pull request from a fork. [Learn more](#).

**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.

- 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.

The screenshot shows the 'Actions secrets / New secret' page. On the left, there's a form with 'Name' set to 'WORKFLOW\_USE' and 'Value' set to 'ghp\_BvOLiwI0ftQwNXWU9u2mXYJNVa'. On the right, under 'Repository secrets', three secrets are listed: 'ACTIONS\_RUNNER\_DEBUG', 'ACTIONS\_STEP\_DEBUG', and 'WORKFLOW\_USE' (which is the one just created). At the bottom is a green 'Add secret' button.

- 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 'Settings' page for a GitHub repository. In the 'Features' section, the 'Issues' checkbox is checked and highlighted with a red circle. Below it, a tooltip explains that issues integrate lightweight task tracking into the repository. At the bottom of the section is a blue button labeled 'Get organized with issue templates' and a green 'Set up templates' button.

- 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.
  - 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\_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 bod is simply a REST call (using the GITHUB\_TOKEN) to create a new issue.
- c. Click the pencil icon to edit it.

```

40 lines (31 sloc) | 1.03 KB
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, 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".

**gwstudent / greetings-actions**  
forked from [skillrepos/greetings-actions](#)

< Code Issues 12 Pull requests Discussions Actions

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

< Edit file Preview changes

```

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.

**Commit changes**

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

Add an optional extended description...

⌘-o Commit directly to the `main` branch.  
 ⌘-n Create a new branch for this commit and start a pull request. [Learn more about pull requests](#).

**Commit changes** **Cancel**

- 8. 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 Actions interface for the 'create-failure-issue' workflow. It displays one workflow run that has completed successfully. A modal window is overlaid, allowing the user to re-run the workflow with different parameters: 'Issue title' is set to 'This is a title' and 'Issue body' is set to 'This is the body text'. The 'Run workflow' button is visible at the bottom of the modal.

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

The screenshot shows the GitHub repository 'gwstudent/greetings-actions'. The 'Issues' tab is selected. A single issue is listed, titled 'This is a title', which was opened 2 minutes ago by 'github-actions[bot]'. The issue is currently open.

- 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} **"}
}'

```

11. 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.)

The screenshot shows a GitHub code editor interface. The top bar displays the repository path 'greetings-actions / .github / workflows / simple-pipe.yml' and the branch 'in main'. Below the header are buttons for 'Edit file', 'Preview changes', and various code settings like 'Spaces', '2', and 'No wrap'. The main content area shows the YAML configuration for the workflow. Line 65 is highlighted with a blue background and contains the command 'exit 1'. The configuration includes a 'count-args' job and a 'create-issue-on-failure' job.

```

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.

12. 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 All workflows

Simple Pipe  
create-failure-issue

All workflow runs Filter workflow runs

15 workflow runs Event ▾ Status ▾ Branch ▾ Actor ▾

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

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

Update simple-pipe.yml Simple Pipe #14 Re-run jobs ▾ ...

**Summary**

Jobs

- build
- count-args
- test-run
- create-issue-on-failure

Triggered via push 3 minutes ago gwstudent pushed → 1539282 main Status Failure Total duration 49s Artifacts 1

**simple-pipe.yml**  
on: push

```

graph LR
    A["✖ count-args 3s"] --> B["✔ test-run"]
    B --> C["✔ build 16s"]
    C --> D["✔ create-issue-on-failure 1s"]
  
```

**Annotations**  
1 error

✖ count-args  
Process completed with exit code 1.

**Artifacts**  
Produced during runtime

Name	Size
greetings-jar	1006 Bytes

create-failure-issue create-failure-issue #2 Re-run jobs ▾ ...

**Summary**

Jobs

- create\_issue\_on\_failure

Manually triggered 3 minutes ago gwstudent → 1539282 Status Success Total duration 13s Artifacts -

**create-failure-issue.yml**  
on: workflow\_dispatch

```

graph LR
    A["✔ create_issue_on_failure 1s"]
  
```

You can also see the new ticket that was opened with the text sent to it.

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

## Automated workflow failure issue for commit 153928267f2fc38a4e91b5bd00d61f033abd59d6 #4

**Open** github-actions bot opened this issue 3 minutes ago · 0 comments

github-actions bot commented 3 minutes ago

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

THE END - THANKS!