CI/CD and DevOps in 3 Weeks

Week 3

Revision 1.4 – 08/24/22

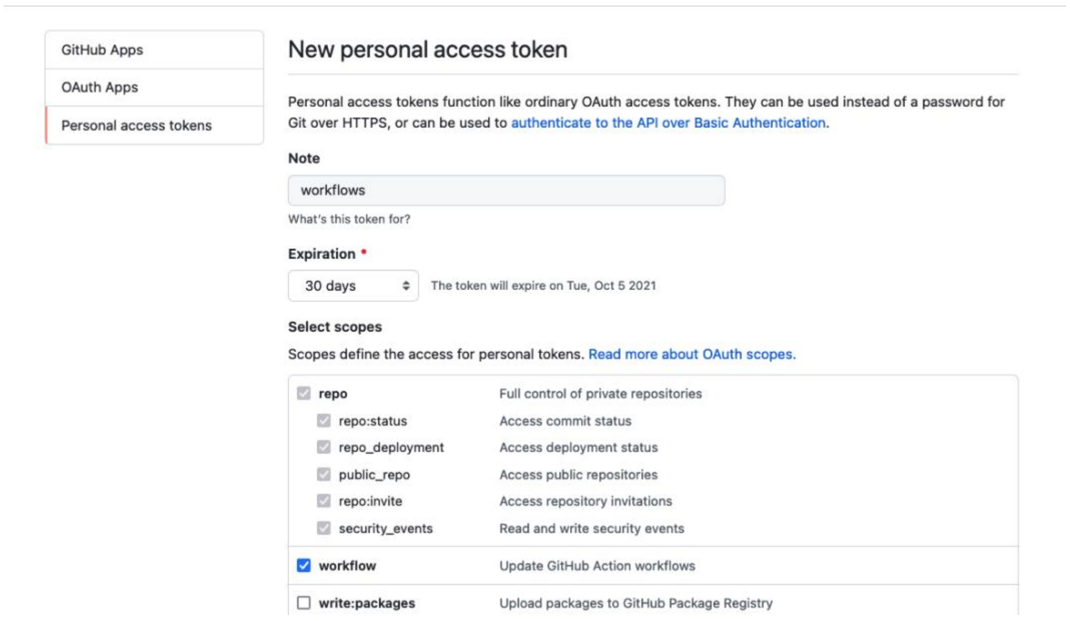
Tech Skills Transformations LLC / Brent Laster

Lab 10: Getting a Personal Access Token and enhancing the workflow-dispatch code

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". 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?

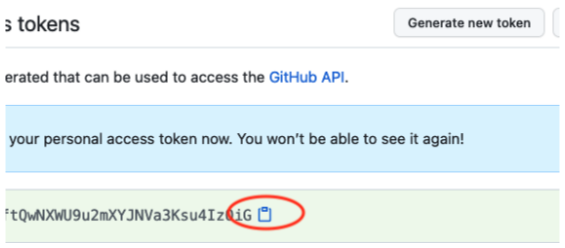
Expiration *: 30 days — The token will expire on Tue, Oct 5 2021

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

✓ repo	Full control of private repositories
✓ repo:status	Access commit status
✓ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
✓ repo:invite	Access repository invitations
✓ security_events	Read and write security events
✓ workflow	Update GitHub Action workflows
☐ write:packages	Upload packages to GitHub Package Registry

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

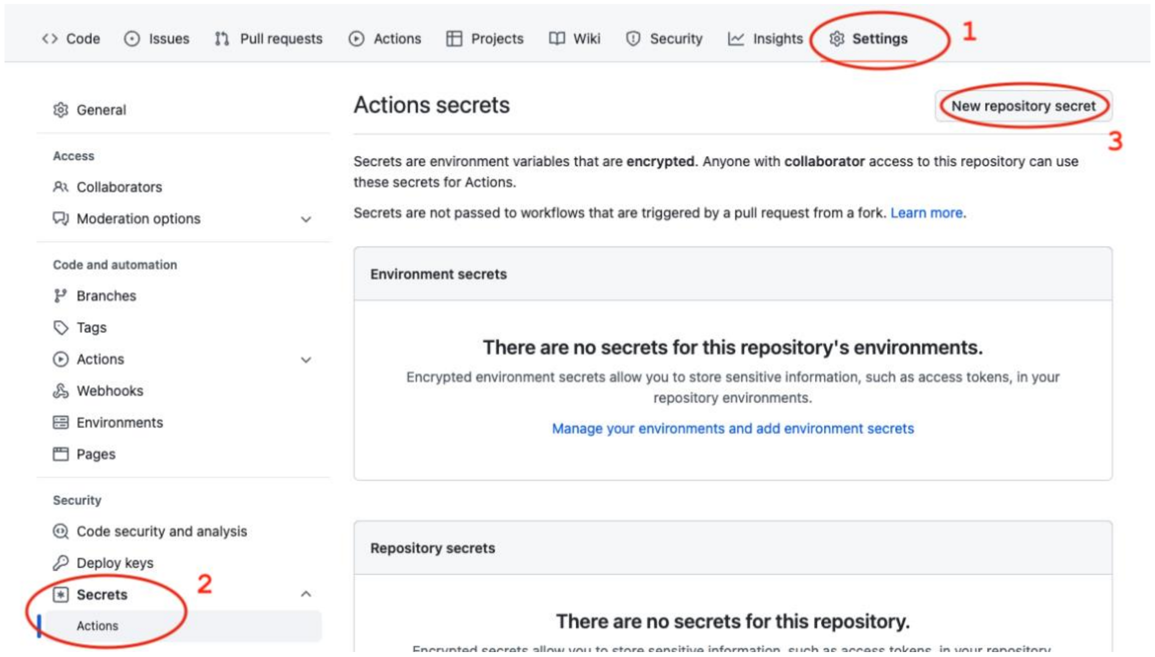


erated that can be used to access the GitHub API.

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

```
ftQwNXWU9u2mXYJNVa3Ksu4IzQiG
```

4. Now we'll create a new secret and store the PAT value in it. Go to the repository and in the top menu select "Settings". Then on the left-hand side, select "Secrets" and select "Actions". Now, click on the "New repository secret in the upper right to create a new secret for the action to use.



Actions secrets

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.

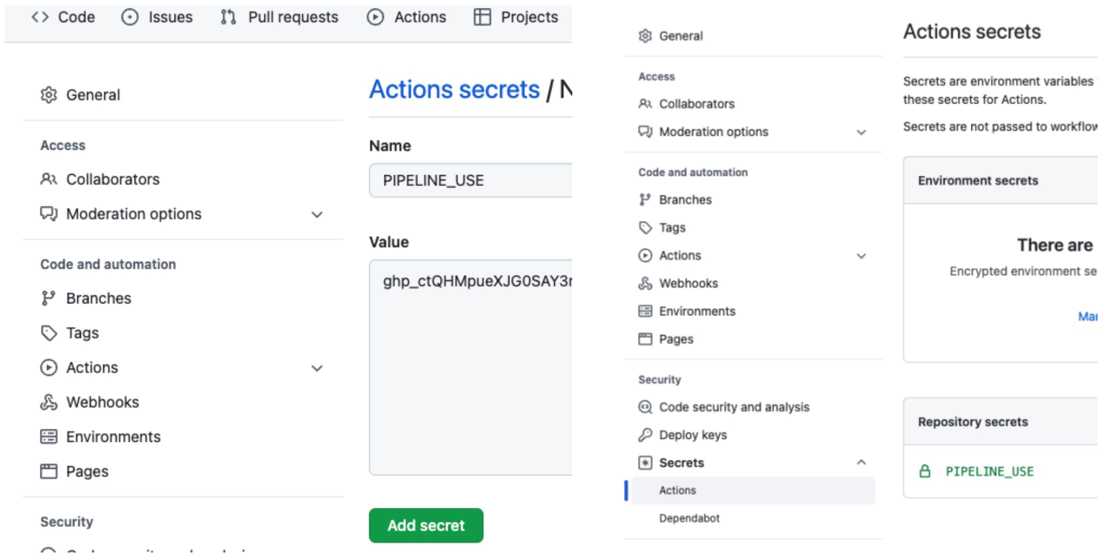
Encrypted environment secrets allow you to store sensitive information, such as access tokens, in your repository environments.

Manage your environments and add environment secrets

Repository secrets

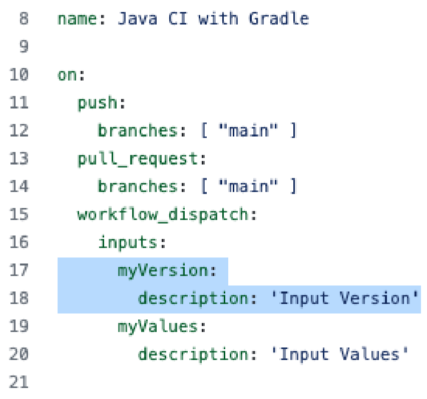
There are no secrets for this repository.

5. For the Name of the new secret, use PIPELINE_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.



6. Now let's update the *.github/workflows/pipeline.yml* workflow to allow providing a version input for the workflow_dispatch code. Go to the project and edit the pipeline.yml file. First, add an additional parameter to allow you to put in a version for the artifact in the workflow_dispatch code in the "on:" section. Add the two lines for a new input value as shown below.

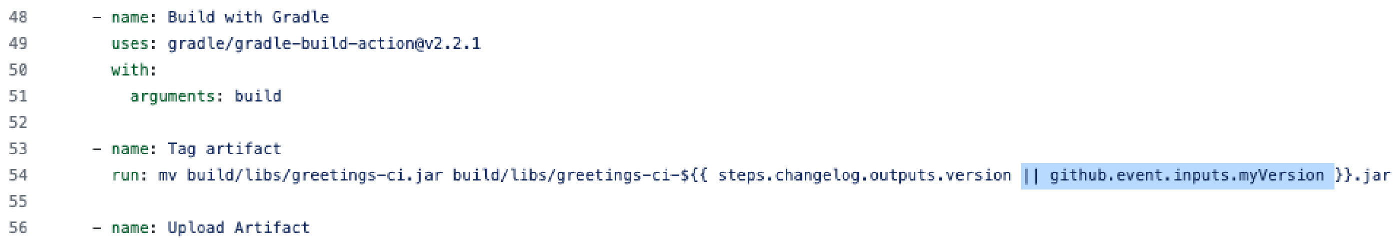
```
myVersion:
  description: 'Input Version'
```



```
name: Java CI with Gradle

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

```

7. Next, add the version input as an alternative to use in the tagging step if the changelog.outputs.version is empty. Make the update highlighted below.

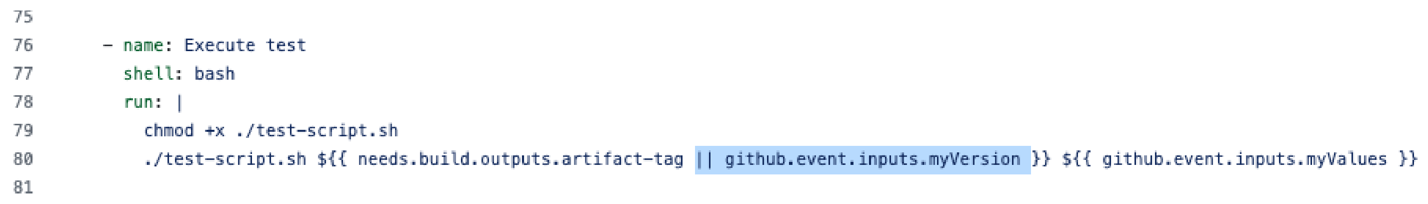
```
- name: Tag artifact
  run: mv build/libs/greetings-ci.jar build/libs/greetings-ci-${{ steps.changelog.outputs.version ||
github.event.inputs.myVersion }}.jar
```



```
- name: Build with Gradle
  uses: gradle/gradle-build-action@v2.2.1
  with:
    arguments: build

- name: Tag artifact
  run: mv build/libs/greetings-ci.jar build/libs/greetings-ci-${{ steps.changelog.outputs.version || github.event.inputs.myVersion }}.jar

- name: Upload Artifact
```

8. Next, we'll update the call to the testing script with a similar approach. If the changelog.outputs.version is empty, we'll try to use the myVersion input. Make the highlighted change below.

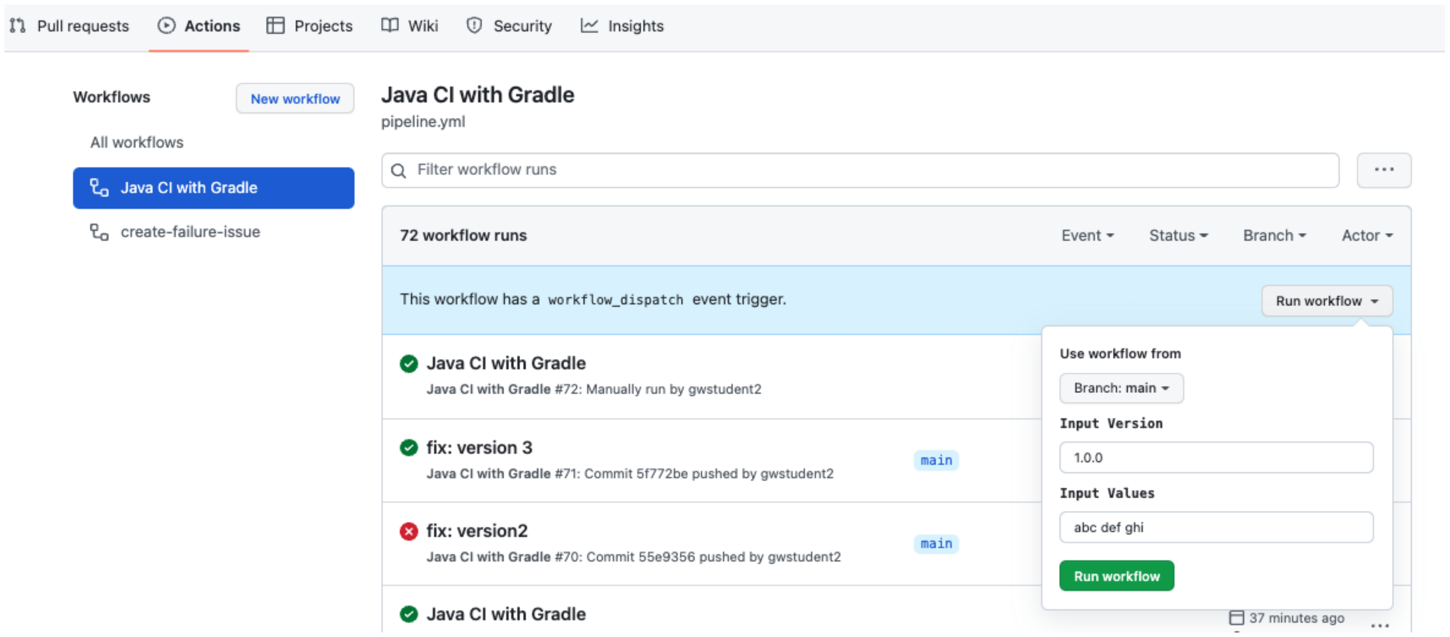
```
- name: Execute test
  shell: bash
  run: |
    chmod +x ./test-script.sh
    ./test-script.sh ${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }} ${{
github.event.inputs.myValues }}
```



```

    - name: Execute test
      shell: bash
      run: |
        chmod +x ./test-script.sh
        ./test-script.sh ${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }} ${{ github.event.inputs.myValues }}

```

9. Go ahead and commit the changes with a "fix: <message>" commit message.

10. You can now launch the workflow with the updated workflow_dispatch event if you want to try it out. Enter a semantic version number for the "Input Version" field.



Java CI with Gradle
pipeline.yml

Filter workflow runs

72 workflow runs — Event, Status, Branch, Actor

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

Java CI with Gradle
Java CI with Gradle #72: Manually run by gwstudent2

fix: version 3 — main
Java CI with Gradle #71: Commit 5f772be pushed by gwstudent2

fix: version2 — main
Java CI with Gradle #70: Commit 55e9356 pushed by gwstudent2

Java CI with Gradle — 37 minutes ago

Use workflow from
Branch: main
Input Version: 1.0.0
Input Values: abc def ghi
Run workflow

END OF LAB

Lab 11: Working with fast feedback and automatically reporting issues

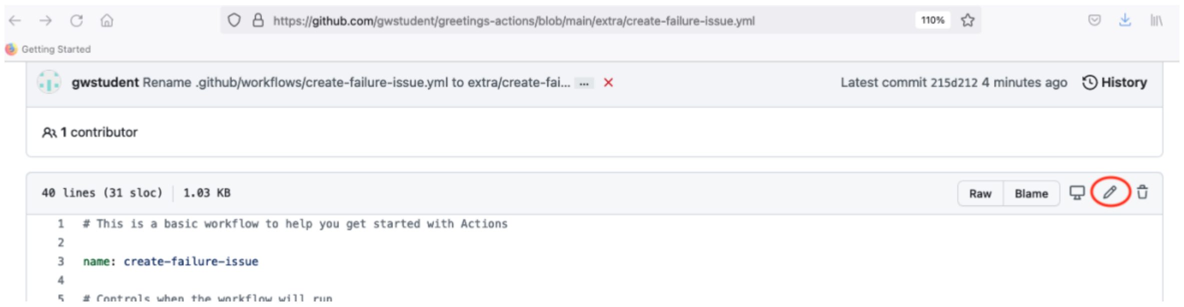
Purpose: Learning how to get fast feedback and automatic failure reporting in our pipeline

1. 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. 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.



https://github.com/gwstudent/greetings-actions/blob/main/extra/create-failure-issue.yml

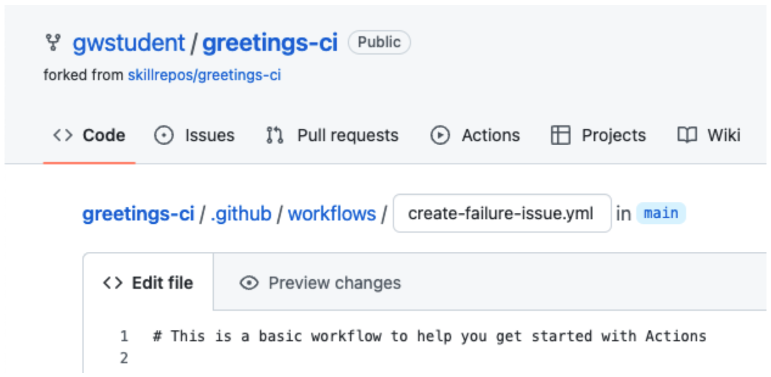
gwstudent Rename .github/workflows/create-failure-issue.yml to extra/create-fai... Latest commit 215d212 4 minutes ago History

1 contributor

40 lines (31 sloc) | 1.03 KB

```
# This is a basic workflow to help you get started with Actions

name: create-failure-issue

# Controls when the workflow will run
```

d. In the filename field, change the name of the 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". (Backspace over *extra/* and type in *.github/workflows/* where it was.)

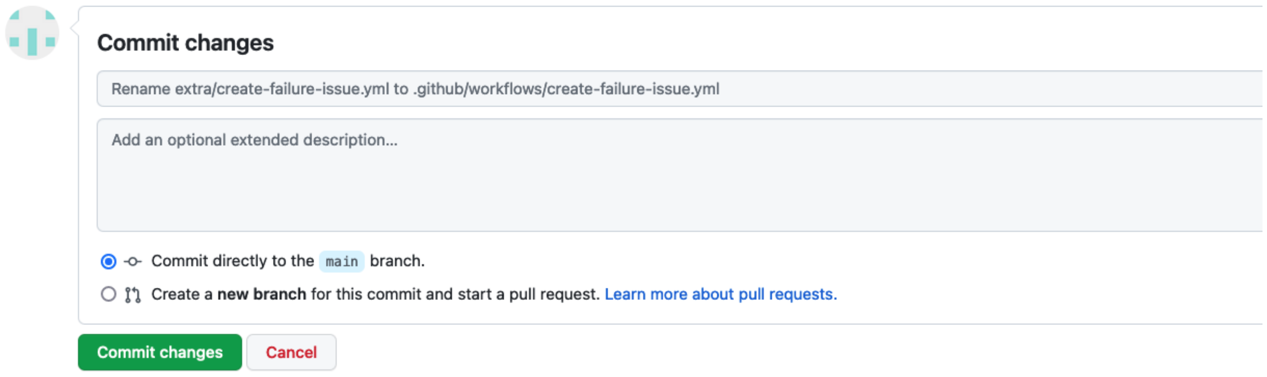


gwstudent / greetings-ci Public
forked from skillrepos/greetings-ci

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

```
# 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.



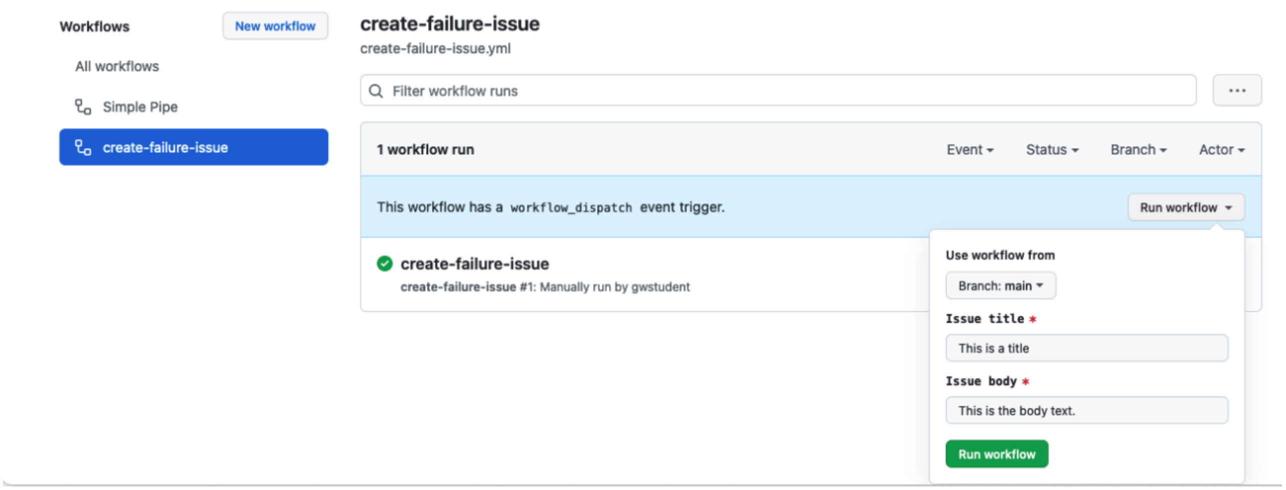
Commit changes

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

Add an optional extended description...

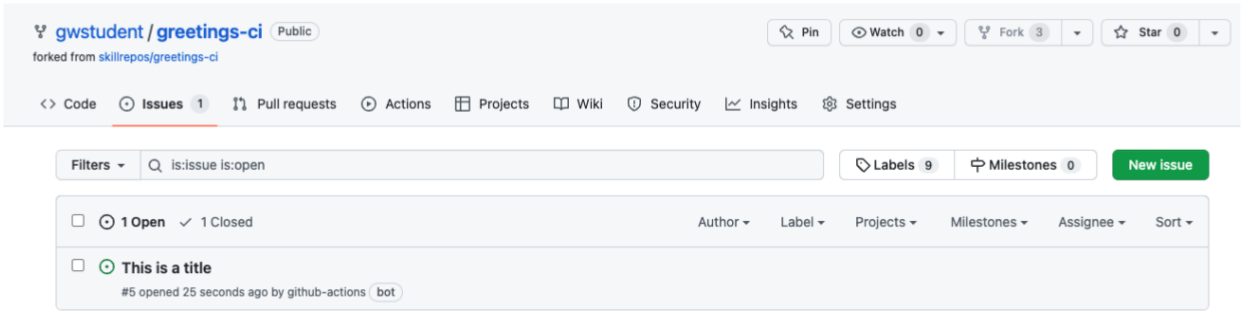
(●) Commit directly to the main branch.
( ) Create a **new branch** for this commit and start a pull request. Learn more about pull requests.

2. 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".



create-failure-issue
create-failure-issue.yml
1 workflow run
This workflow has a workflow_dispatch event trigger.
create-failure-issue
create-failure-issue #1: Manually run by gwstudent
Use workflow from
Branch: main
Issue title *
This is a title
Issue body *
This is the body text.

3. 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.

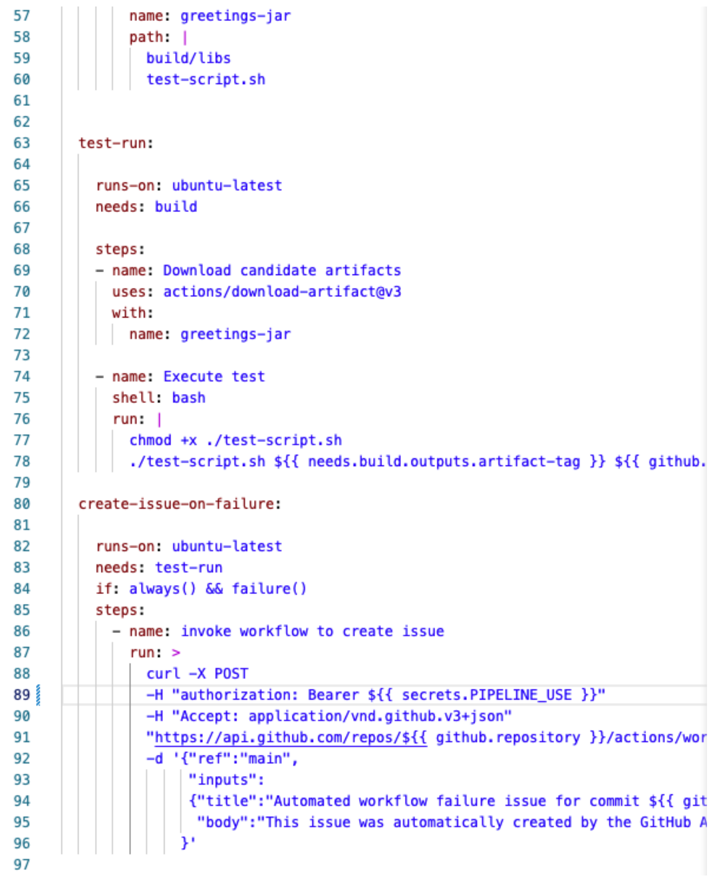


gwstudent / greetings-ci Public
forked from skillrepos/greetings-ci
is:issue is:open
1 Open ✓ 1 Closed
This is a title
#5 opened 25 seconds ago by github-actions bot

4. 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 pipeline.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.) The "create-issue-on-failure" job name should align with the "test-run" job name. (You may want to click on the "raw" link in the upper right to get a better version to copy and paste.) See screenshot further down.

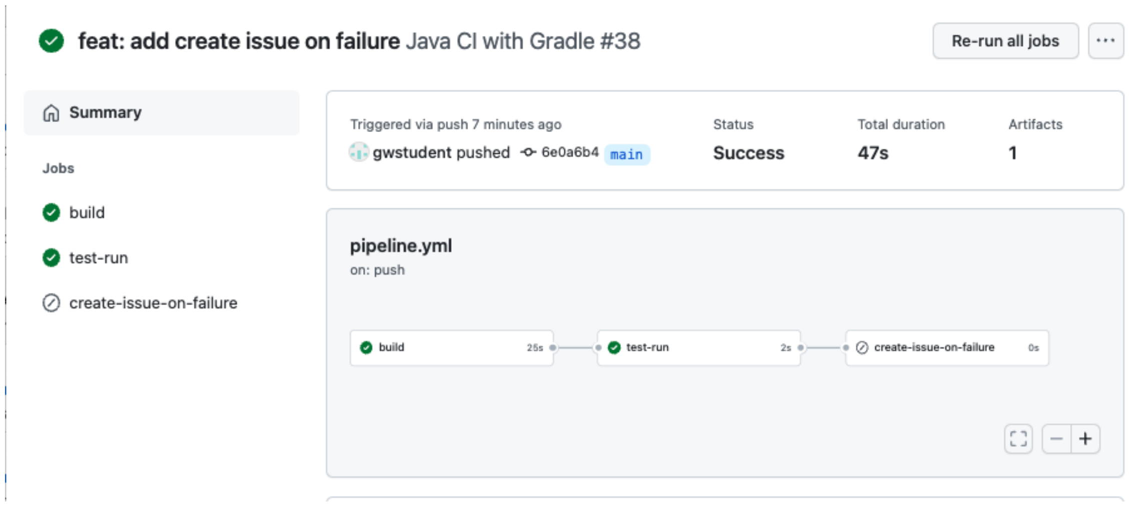
```
create-issue-on-failure:

  runs-on: ubuntu-latest
  needs: test-run
  if: always() && failure()
  steps:
    - name: invoke workflow to create issue
```

```
        run: >
          curl -X POST
          -H "authorization: Bearer ${{ secrets.PIPELINE_USE }}"
          -H "Accept: application/vnd.github.v3+json"
          "https://api.github.com/repos/${{ 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 }} **"}
              }'
```



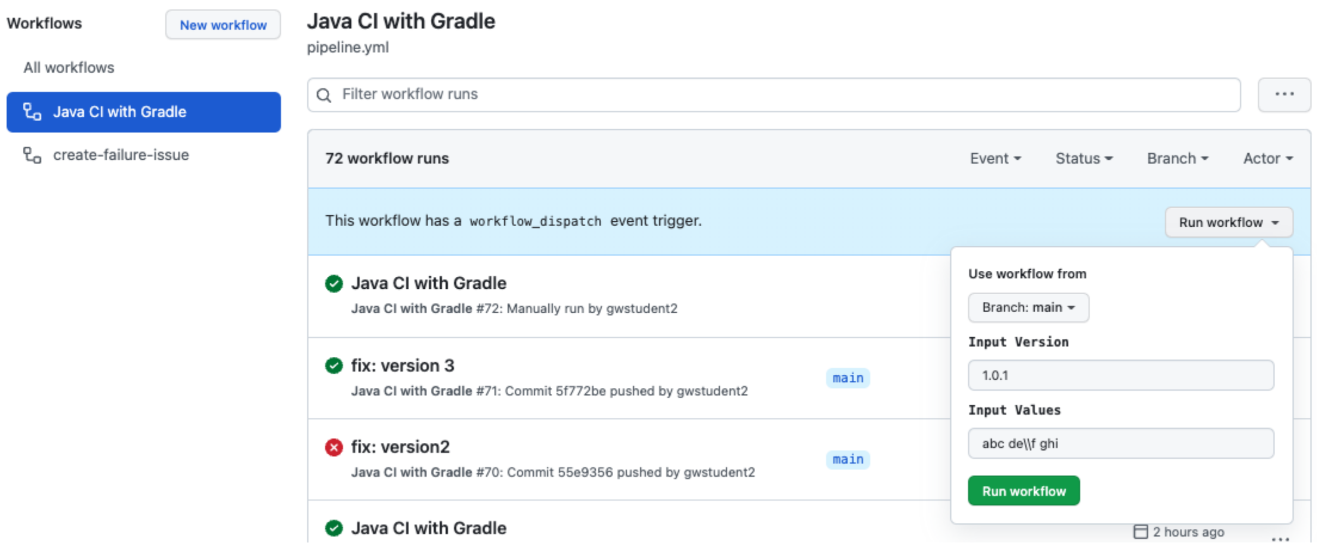
```
        name: greetings-jar
        path: |
          build/libs
          test-script.sh


  test-run:

    runs-on: ubuntu-latest
    needs: build

    steps:
    - name: Download candidate artifacts
      uses: actions/download-artifact@v3
      with:
        name: greetings-jar

    - name: Execute test
      shell: bash
      run: |
        chmod +x ./test-script.sh
        ./test-script.sh ${{ needs.build.outputs.artifact-tag }} ${{ github.

  create-issue-on-failure:

    runs-on: ubuntu-latest
    needs: test-run
    if: always() && failure()
    steps:
      - name: invoke workflow to create issue
        run: >
          curl -X POST
          -H "authorization: Bearer ${{ secrets.PIPELINE_USE }}"
          -H "Accept: application/vnd.github.v3+json"
          "https://api.github.com/repos/${{ github.repository }}/actions/wor
          -d '{"ref":"main",
               "inputs":
               {"title":"Automated workflow failure issue for commit ${{ git
                "body":"This issue was automatically created by the GitHub A
             }'
```

5. After this is committed and the workflow runs, you can look at the output for the run and you'll see that the "create-issue-on-failure" job was skipped. That makes sense because we have the checks in the code and there was no failure on previous jobs.

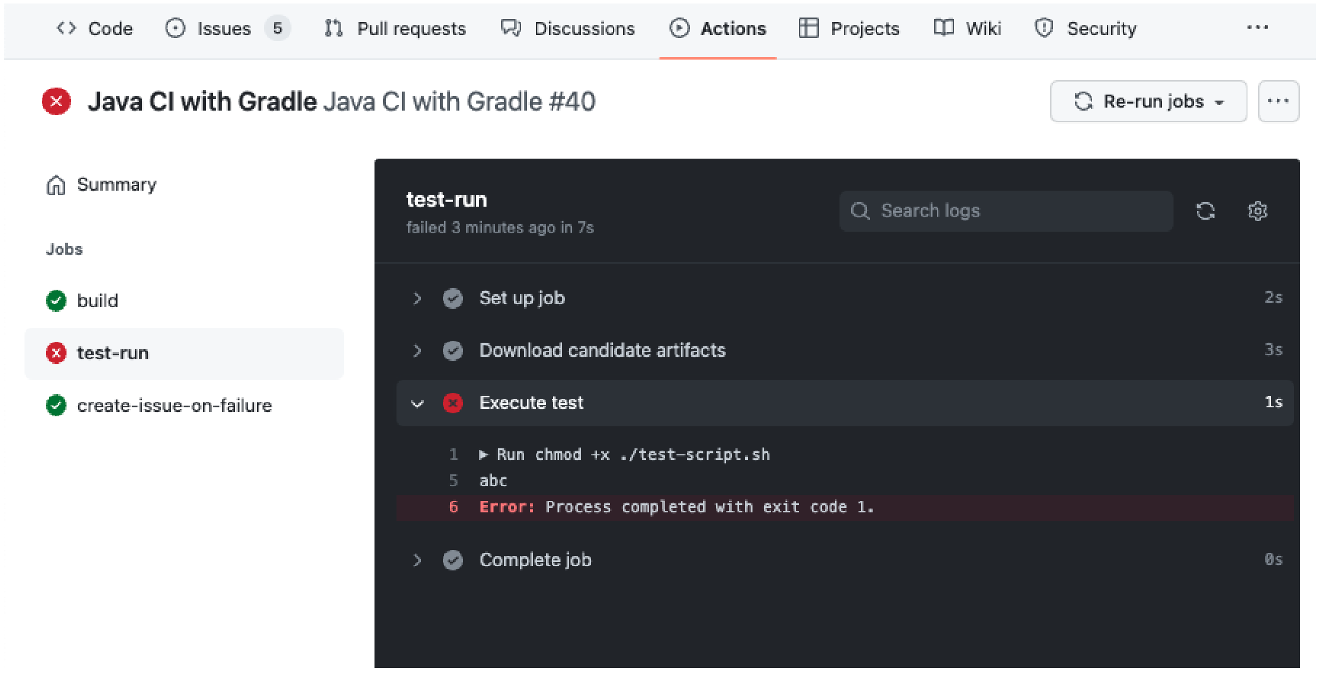

feat: add create issue on failure Java CI with Gradle #38
Re-run all jobs
Summary
Jobs
build
test-run
create-issue-on-failure
Triggered via push 7 minutes ago
gwstudent pushed 6e0a6b4 main
Status
Success
Total duration
47s
Artifacts
1
pipeline.yml
on: push
build 25s
test-run 2s
create-issue-on-failure 0s

6. To have this executed via the "if" statement, we need to have a failure. Let's try some different input with special characters that may not print out as expected. Go to the Actions menu, and then select our main "Java CI with Gradle" workflow. Click on the "Run workflow" button and enter text like below: (that's two backslashes between the "de" and "f"). As long as you have two backslashes somewhere, this should fail.

abc de\\f ghi

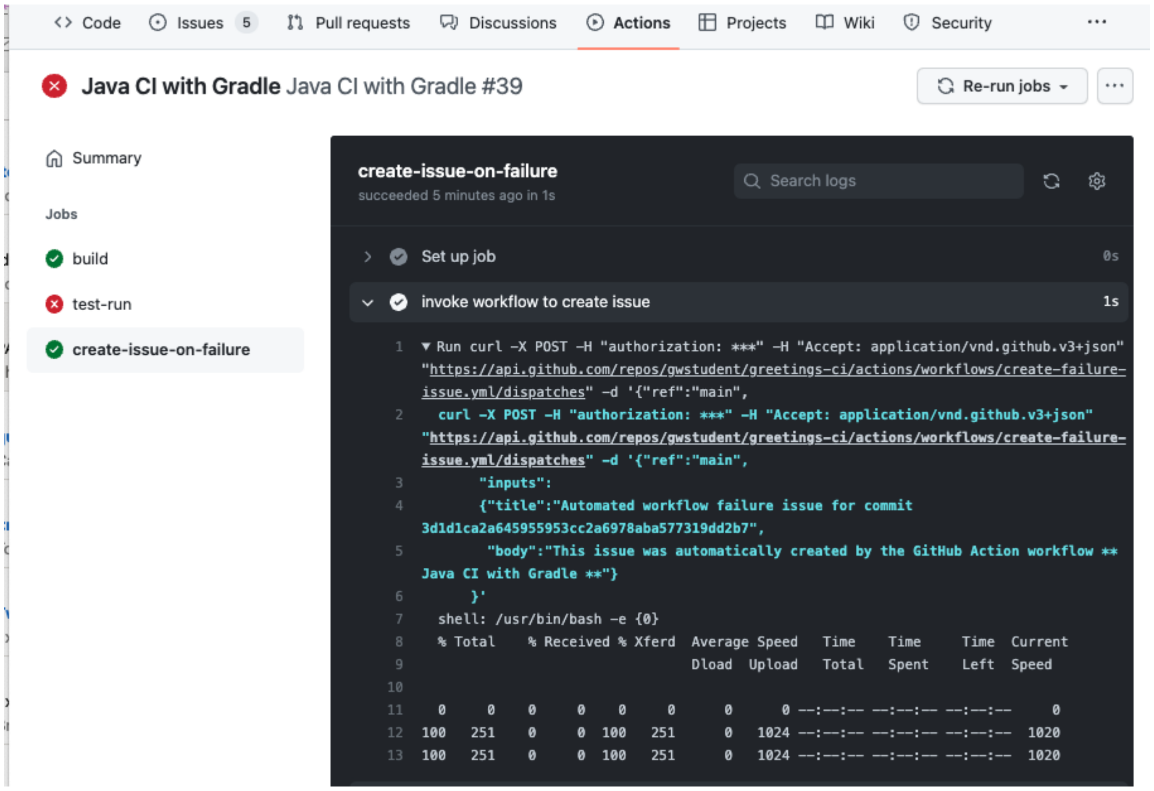


7. After the workflow run completes for this, there should be a failure in our testing. This will in turn, cause our other workflow to create an issue. You can verify the failure in testing by looking at the logs.

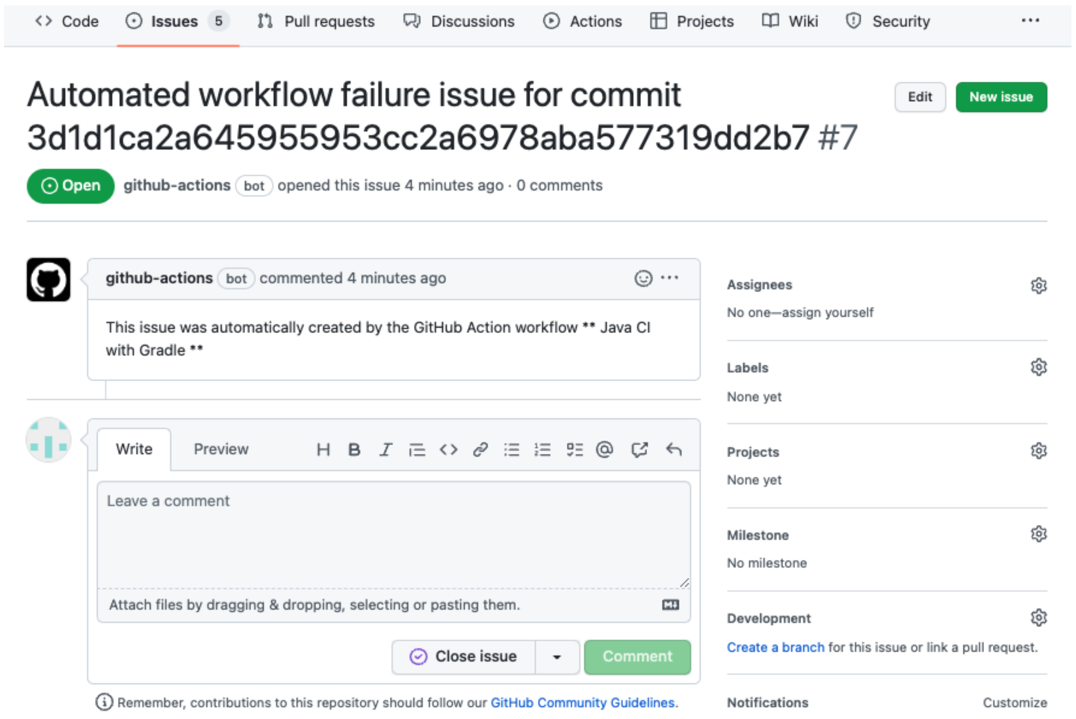


```
▶ Run chmod +x ./test-script.sh
abc
Error: Process completed with exit code 1.
```

You can also verify the new issue got created as a result of the failure through the logs of that job and by looking in the Issues menu at the top.



```
▼ Run curl -X POST -H "authorization: ***" -H "Accept: application/vnd.github.v3+json"
"https://api.github.com/repos/gwstudent/greetings-ci/actions/workflows/create-failure-
issue.yml/dispatches" -d '{"ref":"main",
  curl -X POST -H "authorization: ***" -H "Accept: application/vnd.github.v3+json"
"https://api.github.com/repos/gwstudent/greetings-ci/actions/workflows/create-failure-
issue.yml/dispatches" -d '{"ref":"main",
       "inputs":
       {"title":"Automated workflow failure issue for commit
3d1d1ca2a645955953cc2a6978aba577319dd2b7",
        "body":"This issue was automatically created by the GitHub Action workflow **
Java CI with Gradle **"}
      }'
  shell: /usr/bin/bash -e {0}
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100   251    0     0  100   251      0   1024 --:--:-- --:--:-- --:--:--  1020
100   251    0     0  100   251      0   1024 --:--:-- --:--:-- --:--:--  1020
```



Automated workflow failure issue for commit 3d1d1ca2a645955953cc2a6978aba577319dd2b7 #7

Open github-actions bot opened this issue 4 minutes ago · 0 comments

github-actions bot commented 4 minutes ago

This issue was automatically created by the GitHub Action workflow ** Java CI with Gradle **

Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Close issue
Comment
Remember, contributions to this repository should follow our GitHub Community Guidelines.
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Create a branch for this issue or link a pull request.
Notifications
Customize

END OF LAB

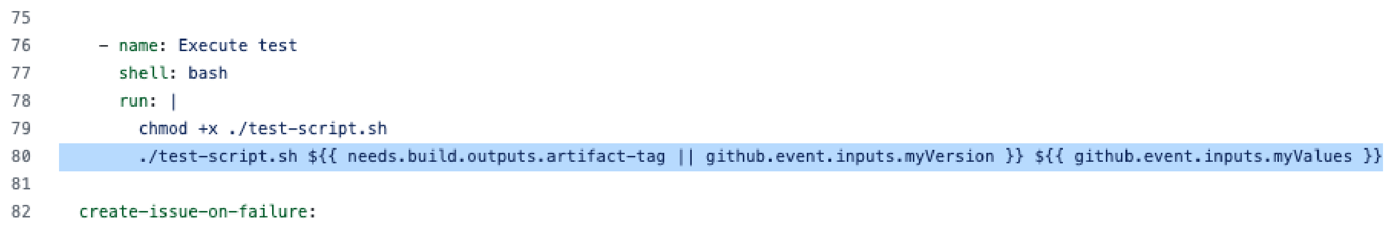
Lab 12 – Securing inputs

Purpose: In this lab, we'll look at how to plug a potential security hole with our inputs.

1. Switch to the pipeline.yml file in the .github/workflows directory and take a look at the "test-run" job and in particular, this line in the "Execute test" step:

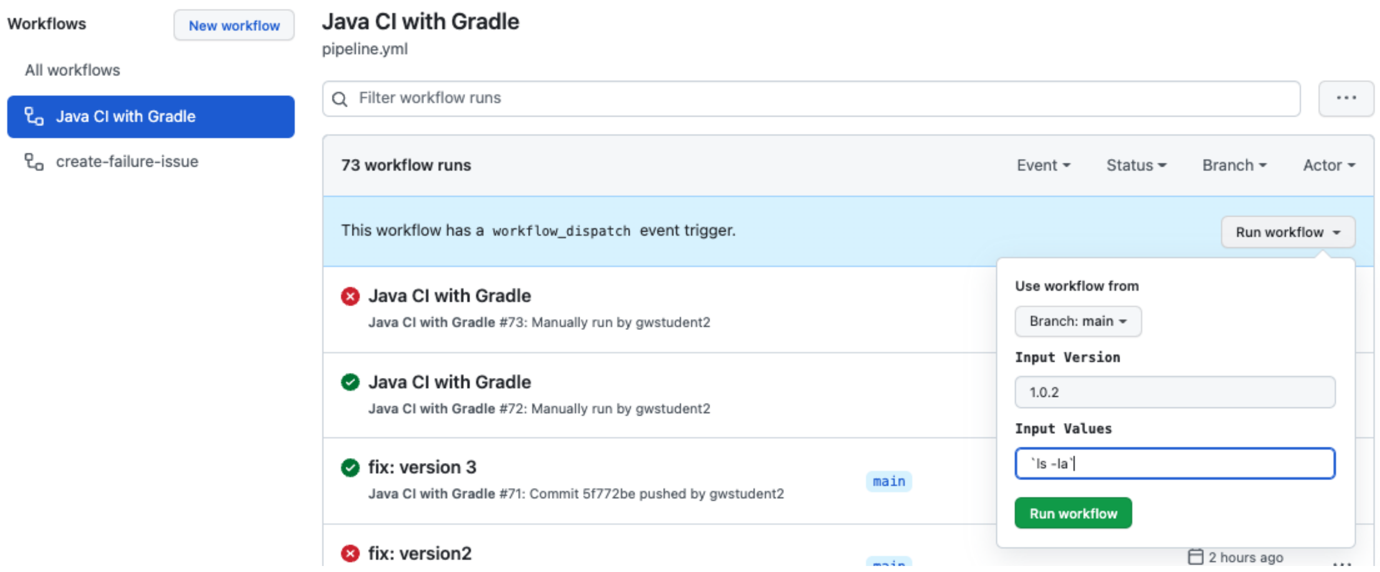
./test-script.sh ${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion

}} ${{ github.event.inputs.myValues }}



```
    - name: Execute test
      shell: bash
      run: |
        chmod +x ./test-script.sh
        ./test-script.sh ${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }} ${{ github.event.inputs.myValues }}

  create-issue-on-failure:
```

2. When we create our pipelines that execute code based on generic inputs, we have to be cognizant of potential security vulnerabilities such as injection attacks. This code is subject to such an attack. To demonstrate this, use the workflow_dispatch event for the workflow in the Actions menu, put in a version and pass in the following as the arguments in the arguments field (NOTE: That is two backquotes around ls -la) `ls -la`



Java CI with Gradle
pipeline.yml

73 workflow runs

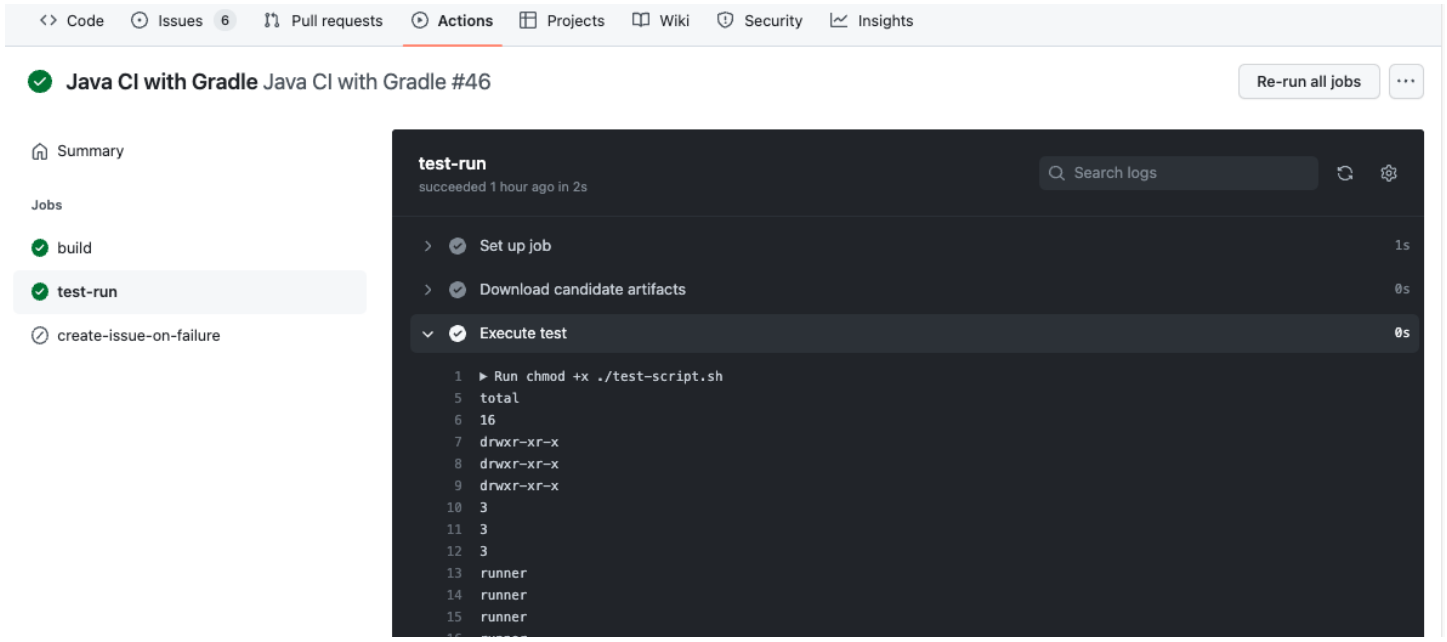
This workflow has a `workflow_dispatch` event trigger.

Input Version: 1.0.2

Input Values: `ls -la`

- Java CI with Gradle — Java CI with Gradle #73: Manually run by gwstudent2
- Java CI with Gradle — Java CI with Gradle #72: Manually run by gwstudent2
- fix: version 3 — Java CI with Gradle #71: Commit 5f772be pushed by gwstudent2 — main
- fix: version2 — 2 hours ago

3. After the run completes, take a look at the output of the step. Notice that it ran successfully but it has actually run the `ls -la` command directly on the runner system. The command was innocuous in this case, but this could have been a more destructive command.



Java CI with Gradle Java CI with Gradle #46

test-run
succeeded 1 hour ago in 2s

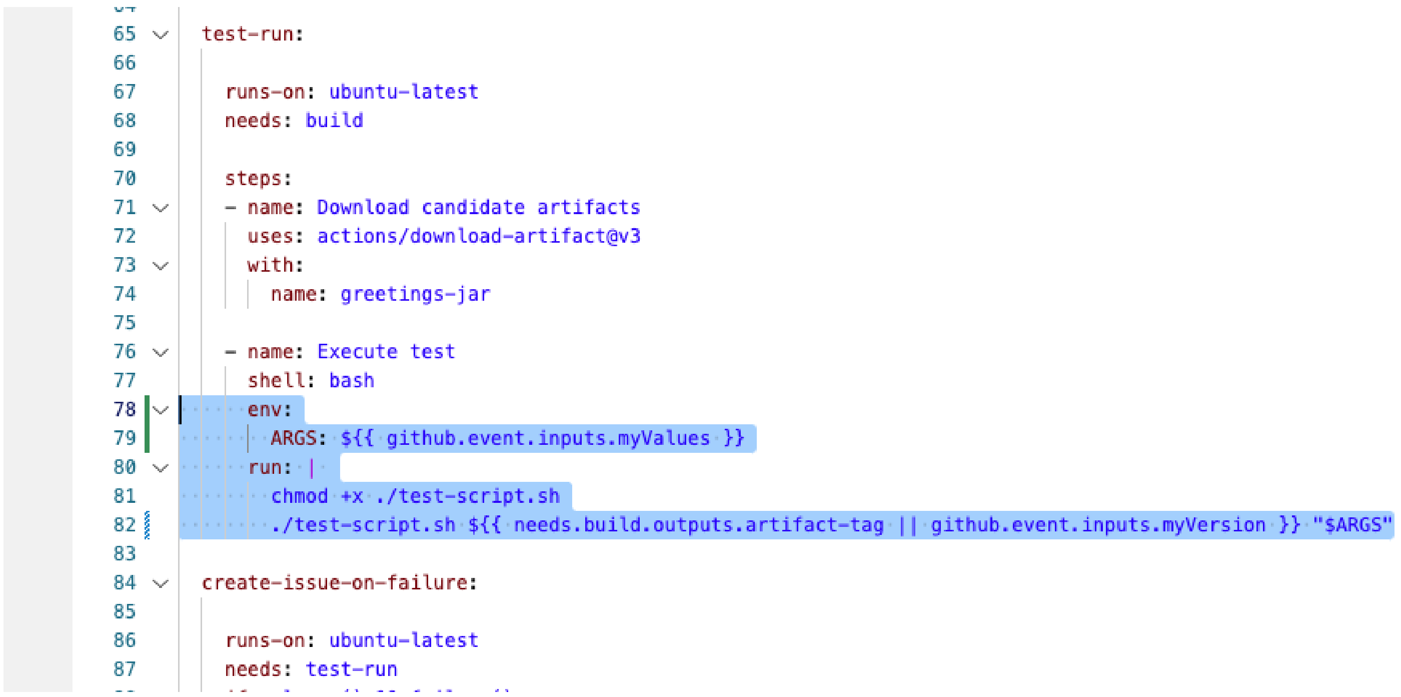
```
▶ Run chmod +x ./test-script.sh
total
16
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x
3
3
3
runner
runner
runner
```

4. Let's fix the command to not be able to execute the code in this way. We can do that by placing the output into an environment variable first and then passing that to the step. Edit the *pipeline.yaml* file and change the code to look like the following:

```
    env:
  ARGS: ${{ github.event.inputs.myValues }}
run: |
  chmod +x ./test-script.sh
```

./test-script.sh ${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion

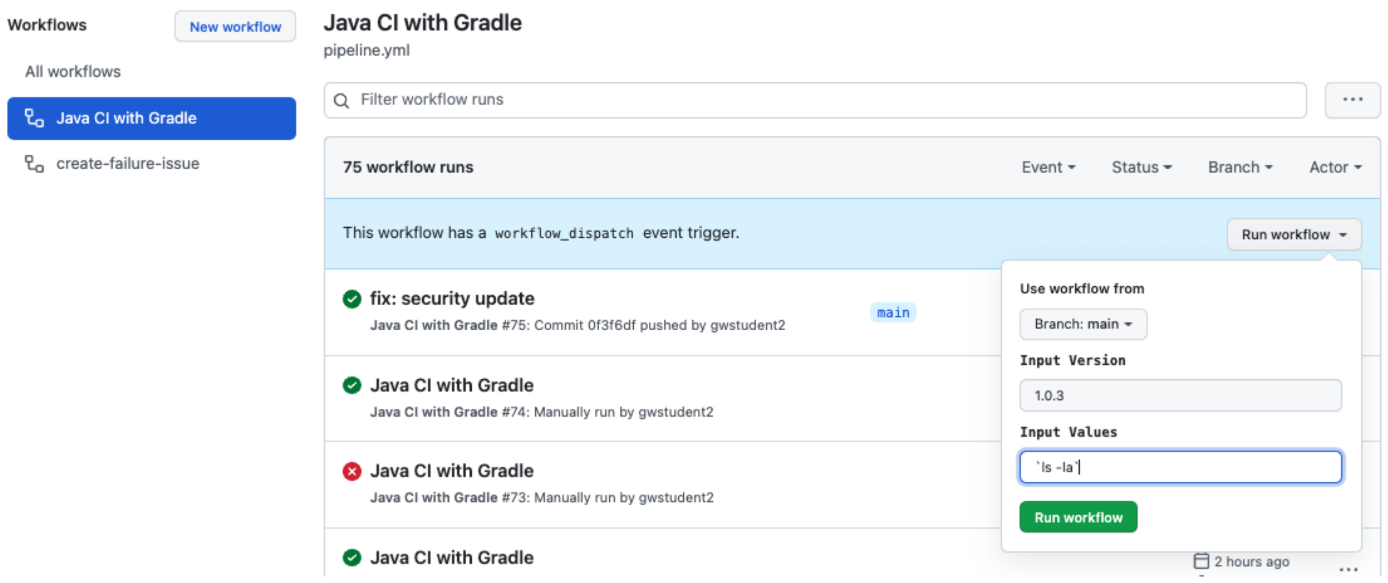
}} "$ARGS"



```
  test-run:

    runs-on: ubuntu-latest
    needs: build

    steps:
    - name: Download candidate artifacts
      uses: actions/download-artifact@v3
      with:
        name: greetings-jar

    - name: Execute test
      shell: bash
      env:
        ARGS: ${{ github.event.inputs.myValues }}
      run: |
        chmod +x ./test-script.sh
        ./test-script.sh ${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }} "$ARGS"

  create-issue-on-failure:

    runs-on: ubuntu-latest
    needs: test-run
```

5. Commit back the changes and wait till the action run for the push completes.

6. Now, you can execute the code again with the same arguments as before.



Java CI with Gradle
pipeline.yml

75 workflow runs

This workflow has a `workflow_dispatch` event trigger.

fix: security update — Java CI with Gradle #75: Commit 0f3f6df pushed by gwstudent2 — main

Java CI with Gradle — Java CI with Gradle #74: Manually run by gwstudent2

Java CI with Gradle — Java CI with Gradle #73: Manually run by gwstudent2

Java CI with Gradle — 2 hours ago

Use workflow from: Branch: main

Input Version: 1.0.3

Input Values: `ls -la`

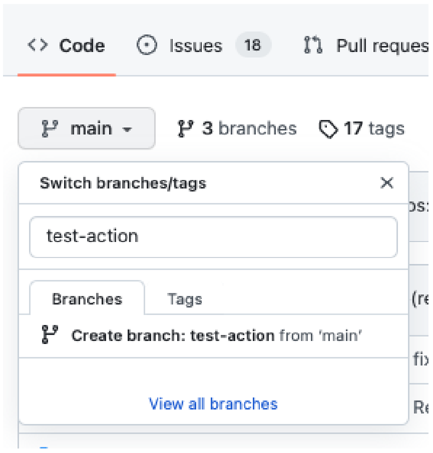
7. Notice that this time, the output did not run the commands, but just echoed them back out as desired.

END OF LAB

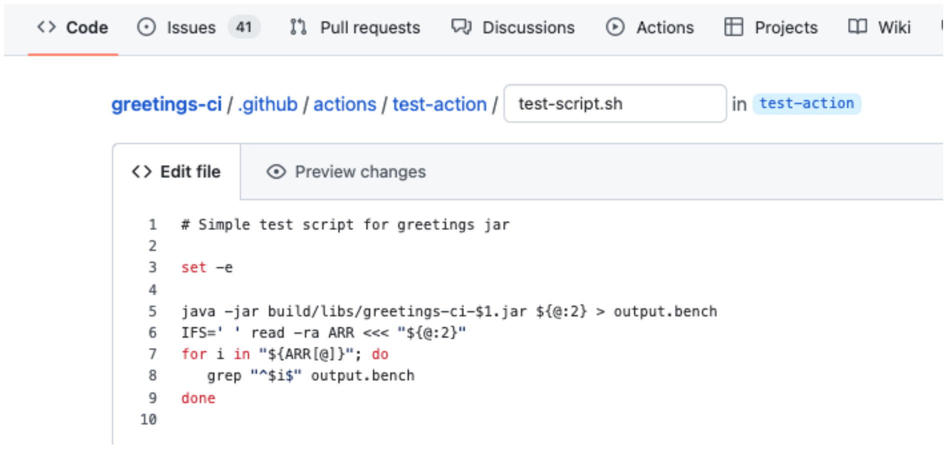
Lab 13 – Separating out jobs into a separate action

Purpose: In this lab, we'll look at how to separate our testing job into a separate action.

1. We're going to make our test script into a composite action. To do this, lets first create a new branch to work with called "test-action". In the "Code" tab, click on the branch dropdown that says "main". Then in the text area that says "Find or create a branch…", enter the text "test-action". Then click on the **"Create branch: test-action from 'main'"** link.



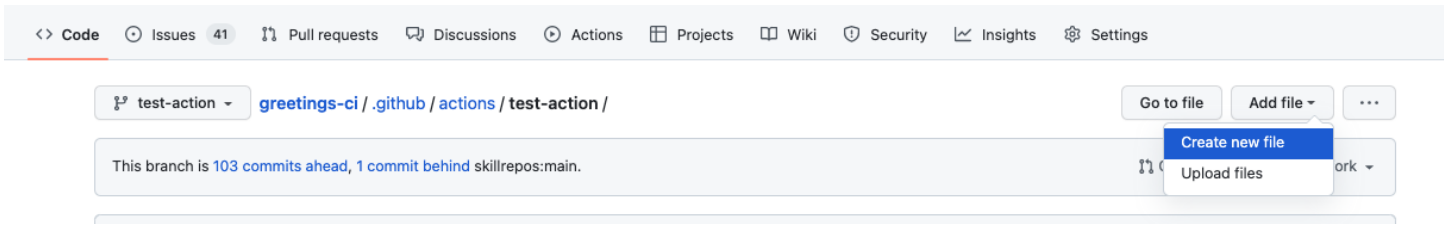
2. You should now be on the "test-action" branch. The "test-script.sh" file will be the basis for our new composite action. So, let's move it to a separate local area for this action. Select the test-script.sh file, edit it, and then add ".github/actions/test-action" to the path as shown below.



```
# Simple test script for greetings jar

set -e

java -jar build/libs/greetings-ci-$1.jar ${@:2} > output.bench
IFS=' ' read -ra ARR <<< "${@:2}"
for i in "${ARR[@]}"; do
   grep "^$i$" output.bench
done
```

3. Click on the green button to commit your changes. Notice that no workflows were kicked off because we don't have events defined in our workflow related to the "test-action" branch.

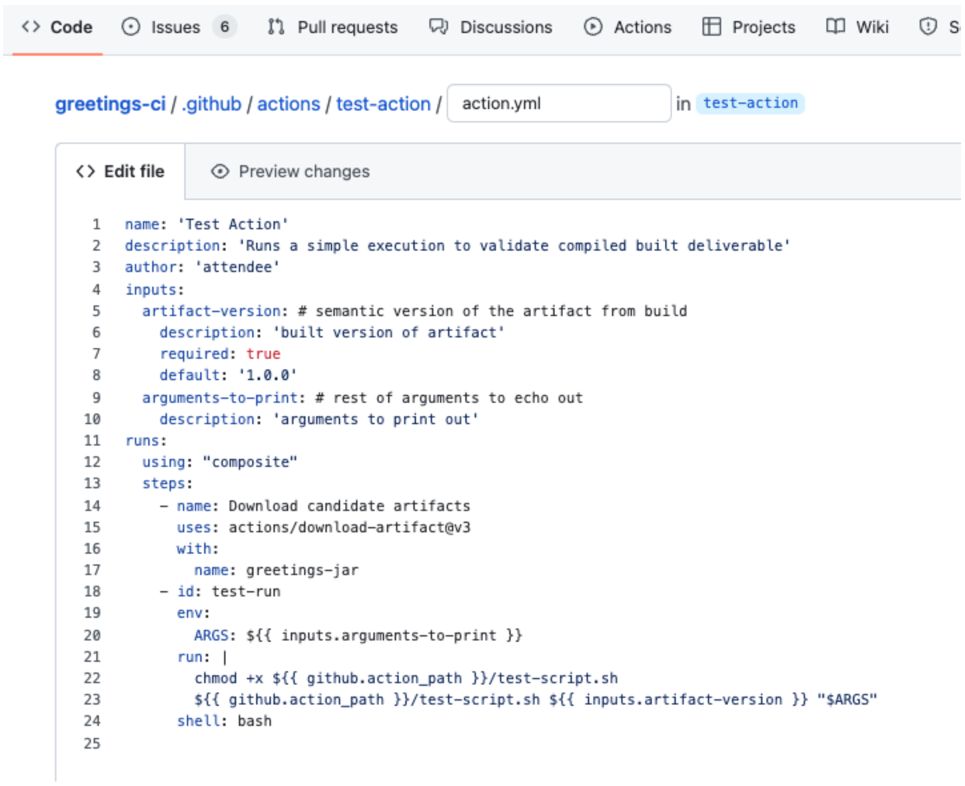
4. Now, let's create the action.yml file for our test action. You will need to create a new file in the path "greetings-ci/.github/actions/test-action" directory by going there, clicking on "Add file" and then clicking on "Create new file"



This branch is 103 commits ahead, 1 commit behind skillrepos:main.

5. Name the new file "action.yml". For the file contents, you can either copy and paste from below or from the file at https://raw.githubusercontent.com/skillrepos/greetings-ci/main/extra/action.yml Commit the file when done.

```
name: 'Test Action'
description: 'Runs a simple execution to validate compiled built deliverable'
author: 'attendee'
inputs:
  artifact-version: # semantic version of the artifact from build
    description: 'built version of artifact'
    required: true
    default: '1.0.0'
  arguments-to-print: # rest of arguments to echo out
    description: 'arguments to print out'
runs:
  using: "composite"
  steps:
    - name: Download candidate artifacts
      uses: actions/download-artifact@v3
      with:
        name: greetings-jar
    - id: test-run
      env:
        ARGS: ${{ inputs.arguments-to-print }}
      run: |
        chmod +x ${{ github.action_path }}/test-script.sh
        ${{ github.action_path }}/test-script.sh ${{ inputs.artifact-version }} "$ARGS"
      shell: bash
```



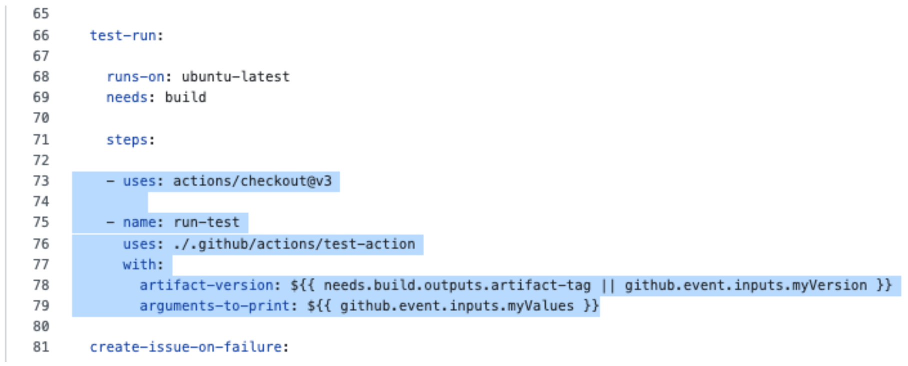
```
name: 'Test Action'
description: 'Runs a simple execution to validate compiled built deliverable'
author: 'attendee'
inputs:
  artifact-version: # semantic version of the artifact from build
    description: 'built version of artifact'
    required: true
    default: '1.0.0'
  arguments-to-print: # rest of arguments to echo out
    description: 'arguments to print out'
runs:
  using: "composite"
  steps:
    - name: Download candidate artifacts
      uses: actions/download-artifact@v3
      with:
        name: greetings-jar
    - id: test-run
      env:
        ARGS: ${{ inputs.arguments-to-print }}
      run: |
        chmod +x ${{ github.action_path }}/test-script.sh
        ${{ github.action_path }}/test-script.sh ${{ inputs.artifact-version }} "$ARGS"
      shell: bash
```

6. This is all we need for our basic composite action. Notice that we've essentially copied over a couple of steps into our composite action that were in the original workflow file. So, we can go back and modify the original

workflow file to use our new action. Still in the "test-action" branch, edit the file "greetings-ci/.github/workflows/pipeline.yaml".

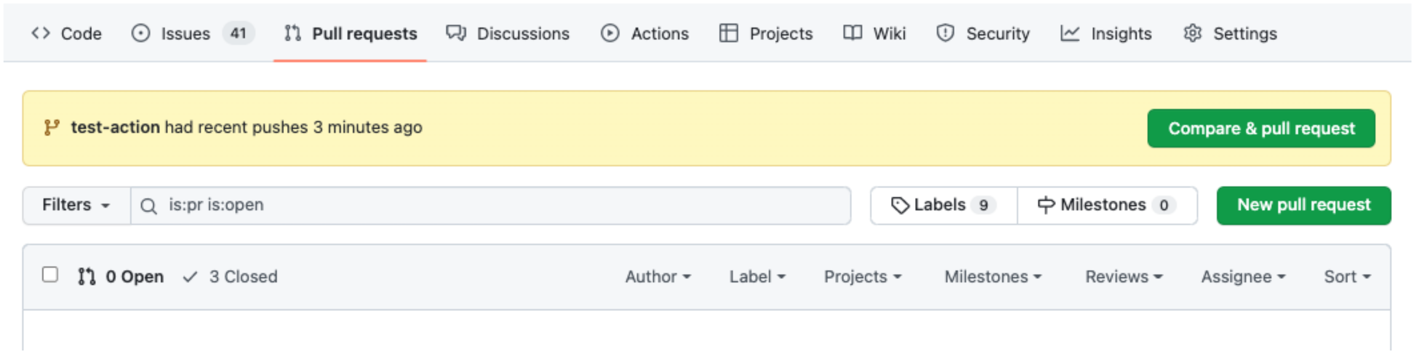
Replace the current steps of test-run, with the new set as shown below. Notice that we need to add a checkout action here to have the necessary pieces from our test-action directory present for the action to get to. Then we just call our new action passing in the parameters. Commit the file when done.

```
- uses: actions/checkout@v3

- name: run-test
  uses: ./.github/actions/test-action
  with:
    artifact-version: ${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
    arguments-to-print: ${{ github.event.inputs.myValues }}
```

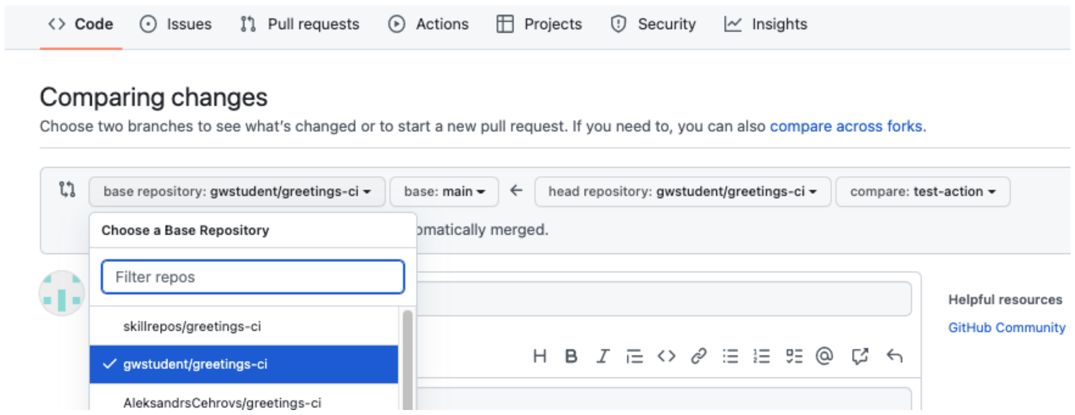


```
  test-run:

    runs-on: ubuntu-latest
    needs: build

    steps:

    - uses: actions/checkout@v3

    - name: run-test
      uses: ./.github/actions/test-action
      with:
        artifact-version: ${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
        arguments-to-print: ${{ github.event.inputs.myValues }}

  create-issue-on-failure:
```

7. Finally, let's merge in the "test-action" branch to the "main" branch. Click on the top-level "Pull requests" menu. You should see a yellow bar with text that indicates the "test-action" branch had recent pushes. Click on the green "Compare & pull request" button.



8. As we've done before, change the "base" portion to be the current repo. After this, it should show that you can merge from the "test-action" branch to the "main" branch. Fill in an appropriate comment and then click the green "Create pull request" button.



Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base repository: gwstudent/greetings-ci | base: main | head repository: gwstudent/greetings-ci | compare: test-action

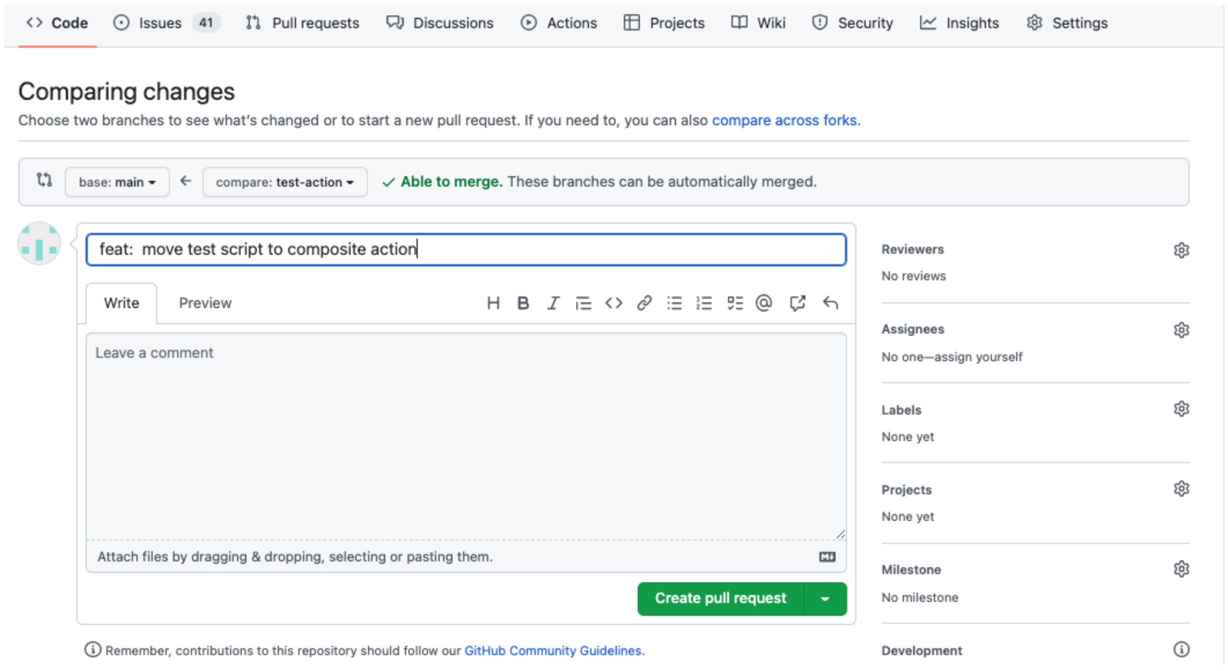
Choose a Base Repository

Filter repos

skillrepos/greetings-ci
✓ gwstudent/greetings-ci
AleksandrsCehrovs/greetings-ci

...omatically merged.

Helpful resources
GitHub Community



Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base: main ← compare: test-action ✓ **Able to merge.** These branches can be automatically merged.

feat: move test script to composite action

Write | Preview

Leave a comment

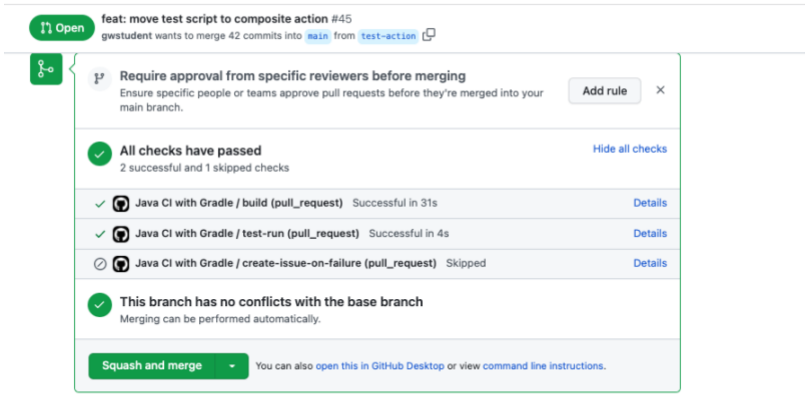
Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Reviewers: No reviews
Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone: No milestone
Development

9. With the Pull Request created, the automated merge checks should run and succeed. After that, you can click on the "Squash and merge" button to complete the merge. Confirm when asked. The merge should complete and the Pull Request should be closed.



feat: move test script to composite action #45
gwstudent wants to merge 42 commits into `main` from `test-action`

Require approval from specific reviewers before merging
Ensure specific people or teams approve pull requests before they're merged into your main branch.
Add rule

All checks have passed
2 successful and 1 skipped checks
Hide all checks

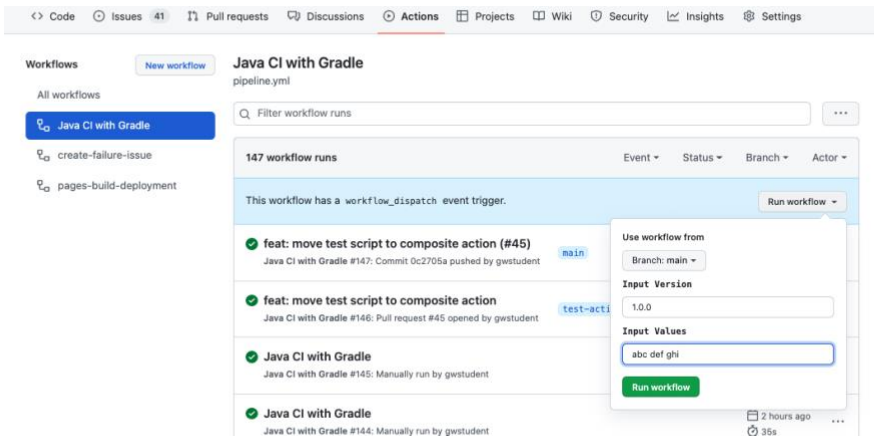
✓ Java CI with Gradle / build (pull_request) Successful in 31s Details
✓ Java CI with Gradle / test-run (pull_request) Successful in 4s Details
⊘ Java CI with Gradle / create-issue-on-failure (pull_request) Skipped Details

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

Squash and merge

You can also open this in GitHub Desktop or view command line instructions.

10. A workflow run will have occurred because of the merge. But if you want to try out the merged code with the action more fully, you can do a manual workflow run as before.



Java CI with Gradle
pipeline.yml

Filter workflow runs

147 workflow runs — Event, Status, Branch, Actor

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

- feat: move test script to composite action (#45) — Java CI with Gradle #147: Commit 0c2705a pushed by gwstudent — main
- feat: move test script to composite action — Java CI with Gradle #146: Pull request #45 opened by gwstudent — test-acti
- Java CI with Gradle — Java CI with Gradle #145: Manually run by gwstudent
- Java CI with Gradle — Java CI with Gradle #144: Manually run by gwstudent — 2 hours ago, 35s

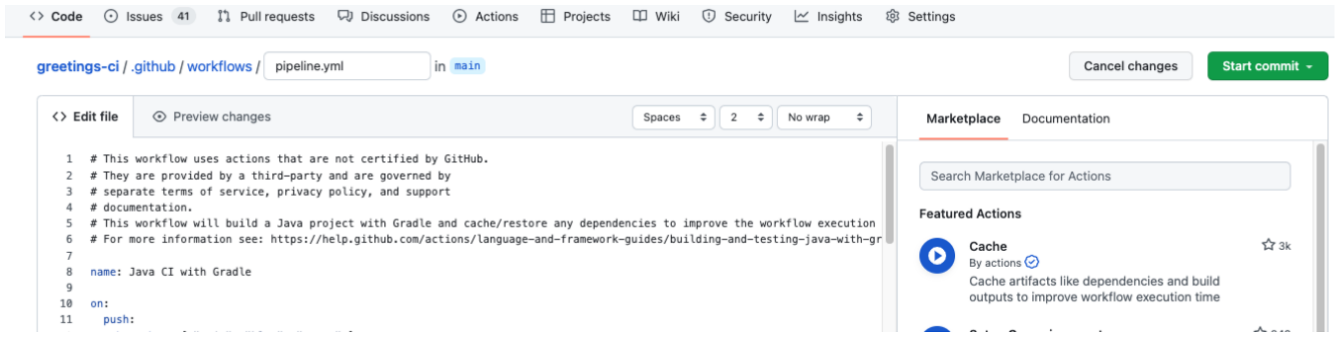
Use workflow from
Branch: main
Input Version: 1.0.0
Input Values: abc def ghi
Run workflow

END OF LAB

Lab 14 – Adding Environments and Releases

Purpose: In this lab, we'll look at how to add staging (blue, green) and production environments and releases.

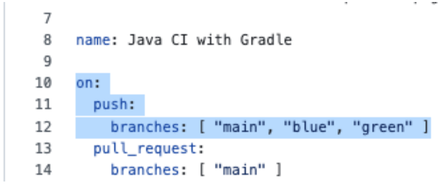
1. Let's add some deploy jobs to our pipeline.yaml file. Edit the .github/workflows/pipeline.yaml file. For simplicity, we can just do this in the main branch.



```
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.
# This workflow will build a Java project with Gradle and cache/restore any dependencies to improve the workflow execution
# For more information see: https://help.github.com/actions/language-and-framework-guides/building-and-testing-java-with-gr

name: Java CI with Gradle

on:
  push:
```

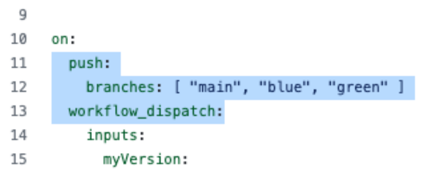
2. We can illustrate blue/green deployment with new branches such as "blue" and "green". So, let's modify the "on:" section first to run the workflow on a push to any of these. Modify the on: push: command to be like the following.

```
on:
  push:
    branches: [ "main", "blue", "green" ]
```



```

name: Java CI with Gradle

on:
  push:
    branches: [ "main", "blue", "green" ]
  pull_request:
    branches: [ "main" ]
```

3. You can also remove the "pull_request" portion.



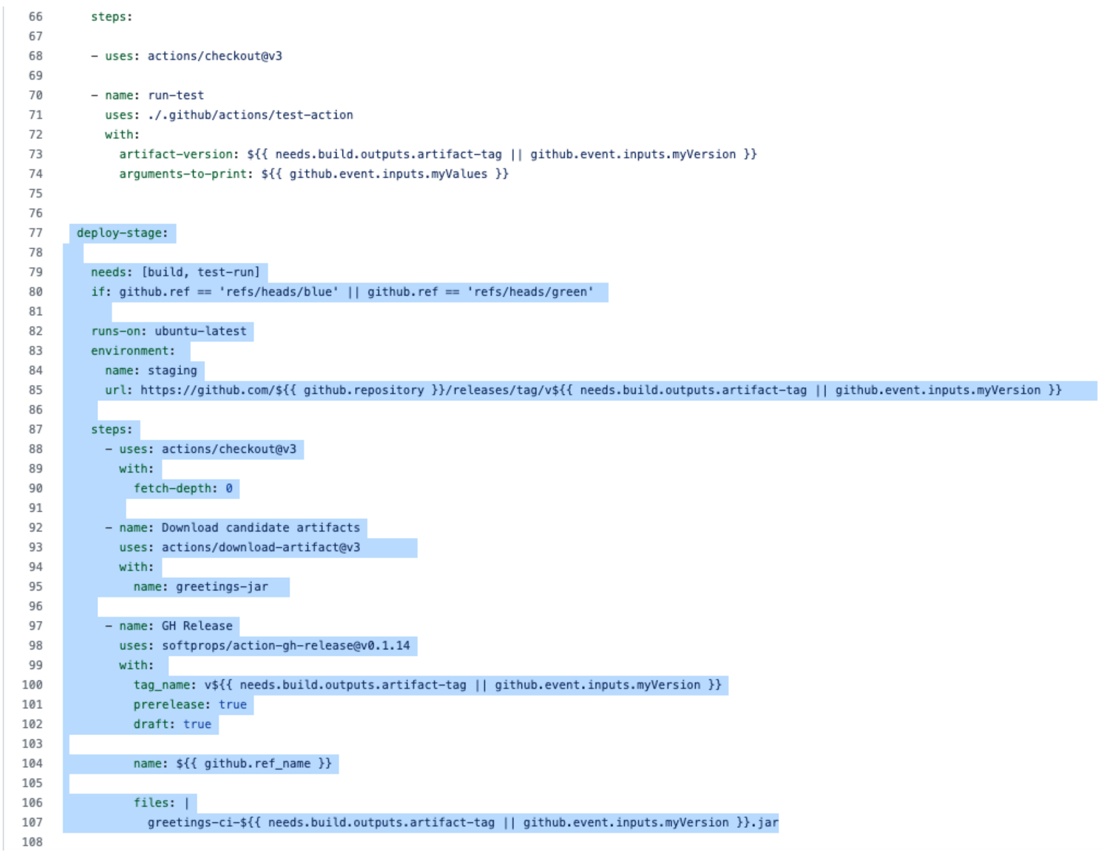
```

on:
  push:
    branches: [ "main", "blue", "green" ]
  workflow_dispatch:
    inputs:
      myVersion:
```

4. Now, let's add the job for deploying a "stage" environment/release. This job can be inserted between the "test-run" job and the "create-issue-on-failure" job. The code for this job is already done for you and can be copied from the file https://raw.githubusercontent.com/skillrepos/greetings-ci/main/extra/deploy-stage.txt Just copy and paste.

(Note: You may need to add a space or two at the front of the first line of output once pasted to get it to line up correctly.)

This code essentially does the following:

- Waits for the build and test jobs to complete (line 79)
- Checks to see if the branch being pushed to is "blue" or "green" (line 80)
- Establishes an environment called "staging" (line 83)
- Sets the associated URL for the environment to the releases page (line 85)
- Checkouts the source code (line 87-90)
- Downloads the jar we built (line 92-95)
- Calls a GitHub Action to create a release that: (line 97-105)
    - is based on the tag we got from the build
    - is set as a draft and prerelease
    - includes the jar file we've built



```
66      steps:
67
68      - uses: actions/checkout@v3
69
70      - name: run-test
71        uses: ./.github/actions/test-action
72        with:
73          artifact-version: ${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
74          arguments-to-print: ${{ github.event.inputs.myValues }}
75
76
77    deploy-stage:
78
79      needs: [build, test-run]
80      if: github.ref == 'refs/heads/blue' || github.ref == 'refs/heads/green'
81
82      runs-on: ubuntu-latest
83      environment:
84        name: staging
85        url: https://github.com/${{ github.repository }}/releases/tag/v${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
86
87      steps:
88        - uses: actions/checkout@v3
89          with:
90            fetch-depth: 0
91
92        - name: Download candidate artifacts
93          uses: actions/download-artifact@v3
94          with:
95            name: greetings-jar
96
97        - name: GH Release
98          uses: softprops/action-gh-release@v0.1.14
99          with:
100           tag_name: v${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
101           prerelease: true
102           draft: true
103
104           name: ${{ github.ref_name }}
105
106           files: |
107             greetings-ci-${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}.jar
108
```

5. Now, let's add the job for deploying a "prod" (production) environment/release from a pull-request being merged into "main". This job can be inserted between the "deploy-stage" job and the "create-issue-on-failure" job. The code for this job is already done for you and can be copied from the file

https://raw.githubusercontent.com/skillrepos/greetings-ci/main/extra/deploy-prod.txt Just copy and paste.

(Note: You may need to add a space or two at the front of the first line of output once pasted to get it to line up correctly.)

This code essentially does the following:

- Waits for the build and test jobs to complete (line 114)

- Checks to see if we got here on the main branch (line 115)

- Establishes an environment called "production" (line 119)

- Sets the associated URL for the environment to the releases page (line 120)

- Checkouts the source code (line 123-125)

- Downloads the jar we built (line 127-130)

- Calls a GitHub Action to create a release that: (line 132-140)

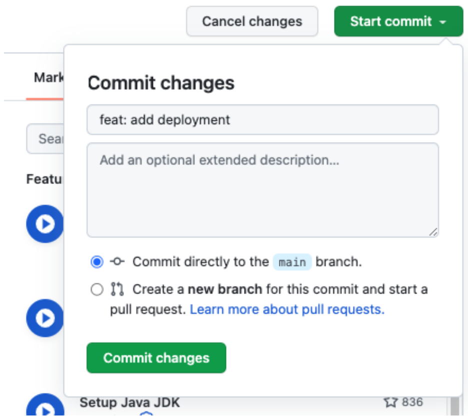
    - is based on the tag we got from the build

    - is named as "Production"

    - includes the jar file we've built and the CHANGELOG

```
          files: |
            greetings-ci-${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}.jar




  deploy-prod:

    needs: [build, test-run]
    if: github.ref == 'refs/heads/main'

    runs-on: ubuntu-latest
    environment:
      name: production
      url: https://github.com/${{ github.repository }}/releases/tag/v${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
    steps:

      - uses: actions/checkout@v3
        with:
          fetch-depth: 0

      - name: Download candidate artifacts
        uses: actions/download-artifact@v3
        with:
          name: greetings-jar

      - name: GH Release
        uses: softprops/action-gh-release@v0.1.14
        with:
          tag_name: v${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
          generate_release_notes: true
          name: Production
          files: |
            CHANGELOG.md
            greetings-ci-${{ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}.jar


  create-issue-on-failure:
```

6. Go ahead and commit your changes to the main branch. You can include a "feat" conventional commit message.



feat: add deployment

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.

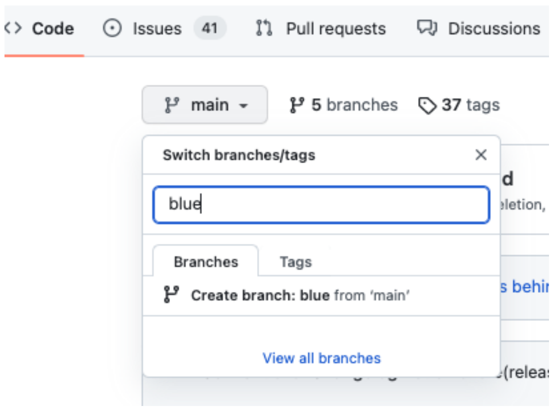
7. This will kick off a new run of the workflow and will create an initial production deployment because of a change in main.

END OF LAB

Lab 15 – Exercising the entire workflow

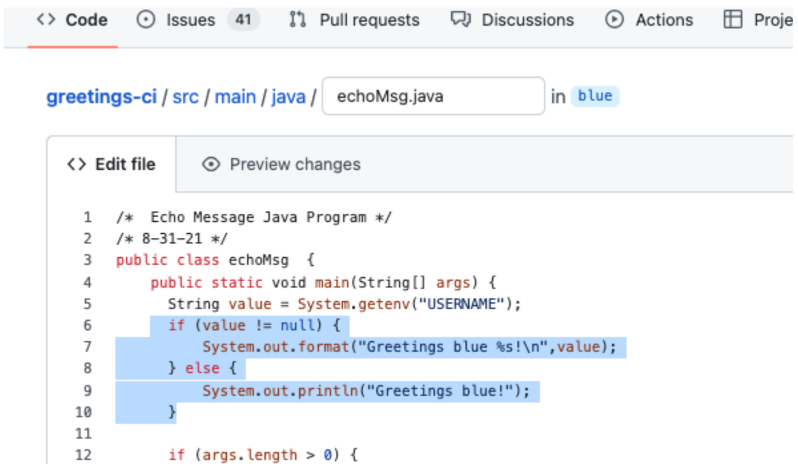
Purpose: In this lab, we'll see how to make a change in source code and have it processed through the pipeline.

1. In the example of using a "blue/green" environment, let's create a branch called "blue" from the "main" branch to make some changes on. Do this just as you've done before.



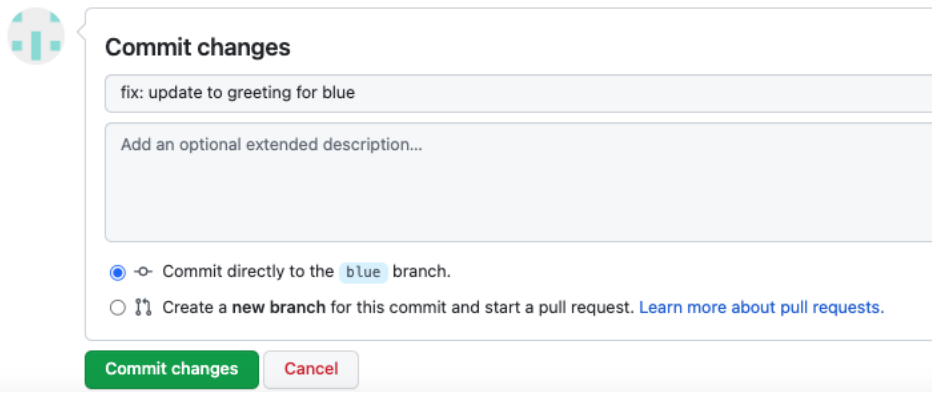
2. In the "blue" branch, edit the file src/main/java/echoMsg.java. Make a simple, non-breaking change like adding "blue" to the lines that print out "Greetings". See text and figure below.

```
if (value != null) {
    System.out.format("Greetings blue %s!\n",value);
} else {
    System.out.println("Greetings blue!");
}
```



```
/*  Echo Message Java Program */
/* 8-31-21 */
public class echoMsg  {
    public static void main(String[] args) {
      String value = System.getenv("USERNAME");
      if (value != null) {
          System.out.format("Greetings blue %s!\n",value);
      } else {
          System.out.println("Greetings blue!");
      }

      if (args.length > 0) {
```

3. Commit the changes with an appropriate "fix: " conventional commit message.



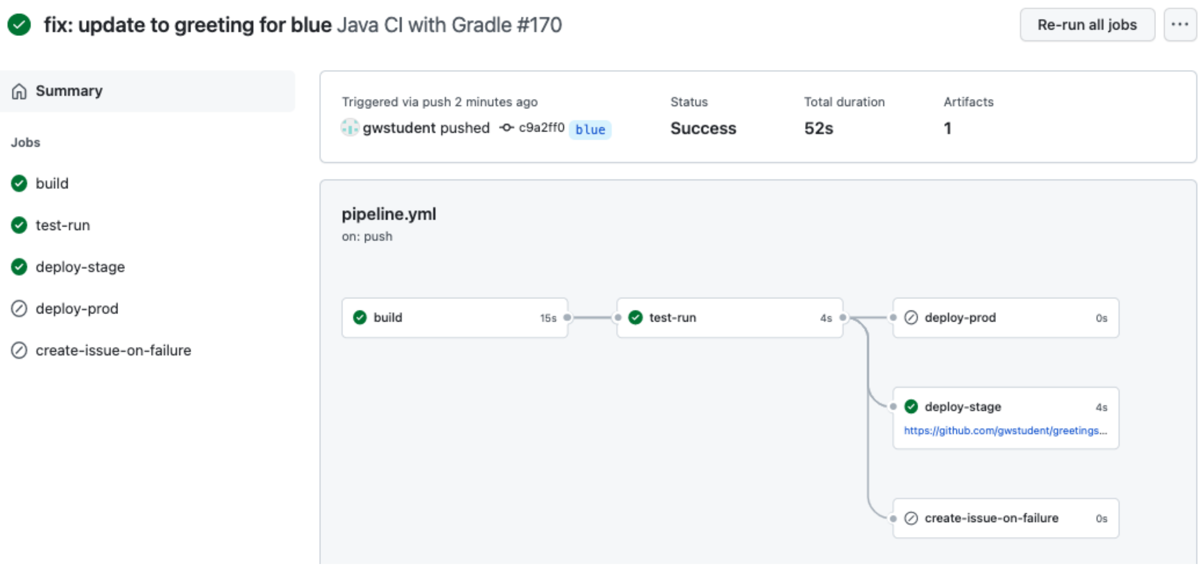
fix: update to greeting for blue

Add an optional extended description...

[✓ Commit directly to the blue branch.]

Create a **new branch** for this commit and start a pull request. Learn more about pull requests.

4. After the workflow run completes, you can click on the run and look at the job graph. You should be able to see that it executed the build and test pieces and then deployed it to the stage environment.



fix: update to greeting for blue Java CI with Gradle #170

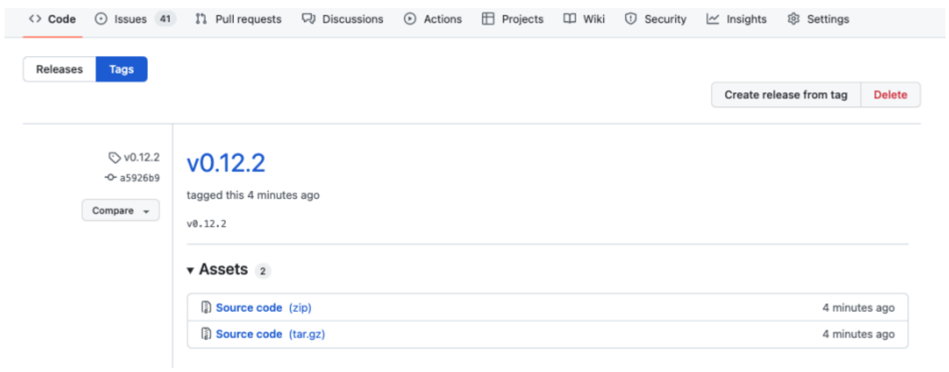
Triggered via push 2 minutes ago
gwstudent pushed c9a2ff0 blue

Status	Total duration	Artifacts
Success	52s	1

pipeline.yml
on: push

Jobs: build (15s), test-run (4s), deploy-prod (0s), deploy-stage (4s) https://github.com/gwstudent/greetings..., create-issue-on-failure (0s)

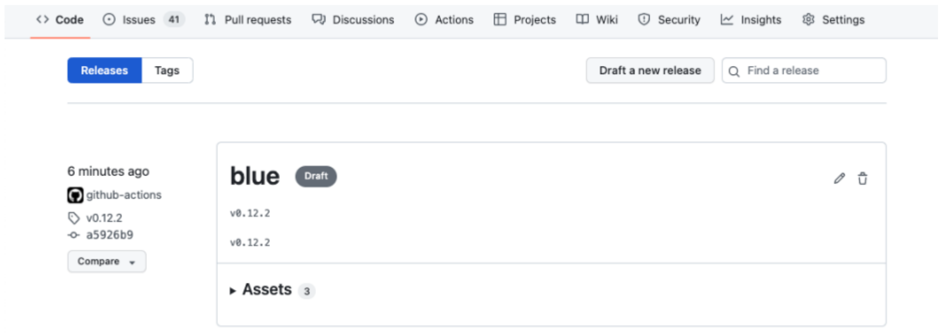
5. Now, click on the link in the "deploy-stage" box. This will take you to the tagged version of the source repo.



v0.12.2
tagged this 4 minutes ago
v0.12.2

Assets 2
Source code (zip) 4 minutes ago
Source code (tar.gz) 4 minutes ago

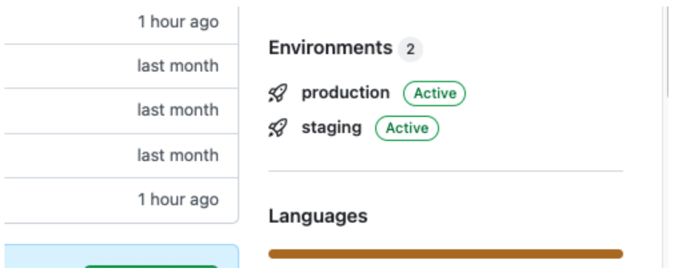
6. If you click on the "Releases" item next to "Tags", you can see the draft release that was created.

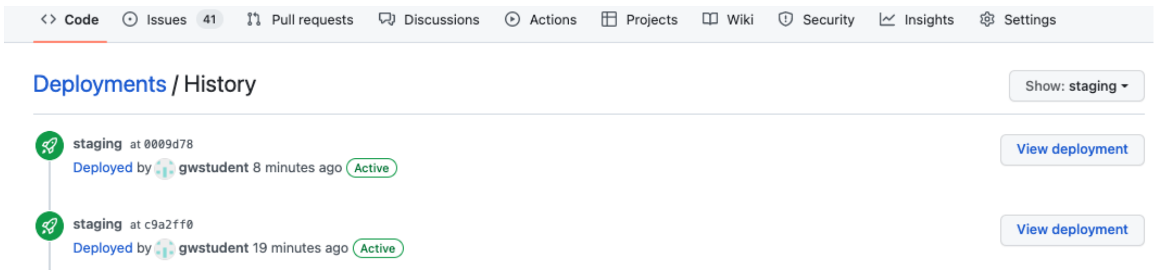


blue Draft
v0.12.2
v0.12.2
Assets 3

6 minutes ago
github-actions
v0.12.2
a5926b9

7. And, if you click on the main code page, in the lower right, you'll be able to see a new "Staging" environment. You can click on that to see a list of recent deployments there.

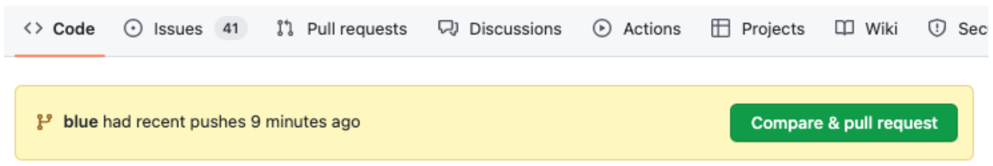




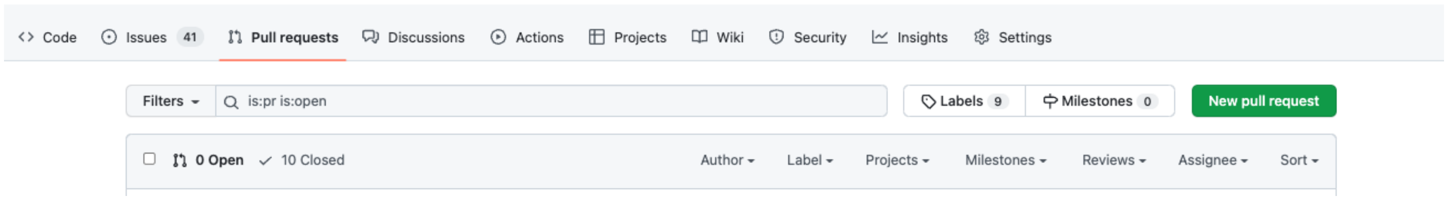
staging at 0009d78
Deployed by gwstudent 8 minutes ago Active

staging at c9a2ff0
Deployed by gwstudent 19 minutes ago Active

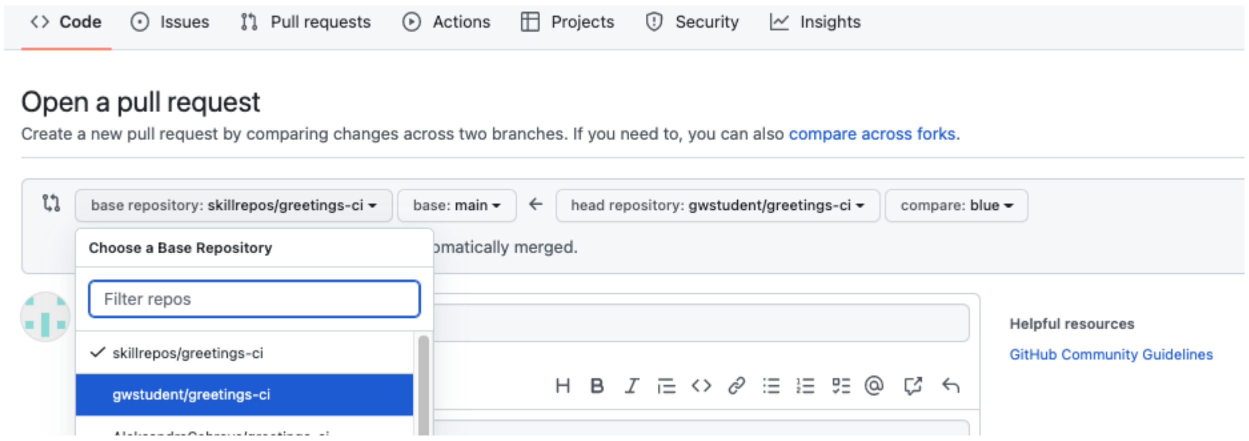
8. Since everything built ok, we can deploy this change to the production environment. To merge the changes, we can just create a pull request to main and merge it. In the "Code" page for your repository, there may be a yellow bar that says "blue had recent pushes…" If so, click on the big green "Compare & pull request" button.



If not, got to "Pull requests" and then click on "New pull request" and "Create pull request".



9. Change the dropdown at the top to select the same repository so you are merging the "blue" branch into the "main" branch. Add a conventional commit message like "fix: greet blue". Then proceed to "Create pull request" by clicking the other green button.



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base repository: skillrepos/greetings-ci ▾ | base: main ▾ | ← | head repository: gwstudent/greetings-ci ▾ | compare: blue ▾

Choose a Base Repository

Filter repos

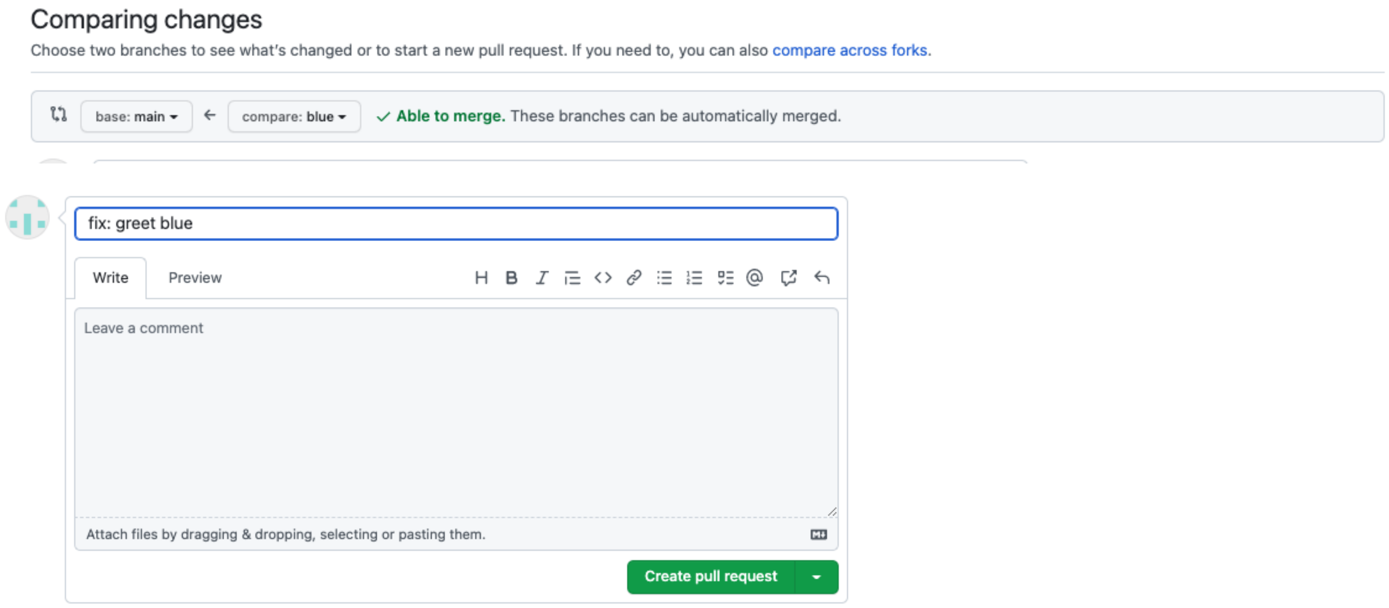
✓ skillrepos/greetings-ci

gwstudent/greetings-ci

omatically merged.

Helpful resources

GitHub Community Guidelines



Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

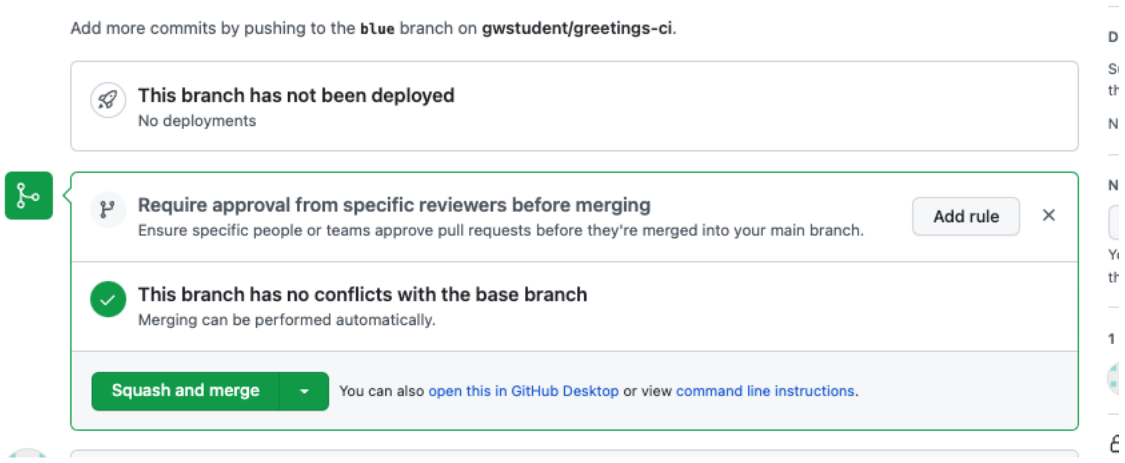
base: main ▾ | ← | compare: blue ▾ | ✓ **Able to merge.** These branches can be automatically merged.

fix: greet blue

Leave a comment

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

10. At this point, you can go ahead and click the "Squash and merge" button when available and confirm.



Add more commits by pushing to the **blue** branch on **gwstudent/greetings-ci**.

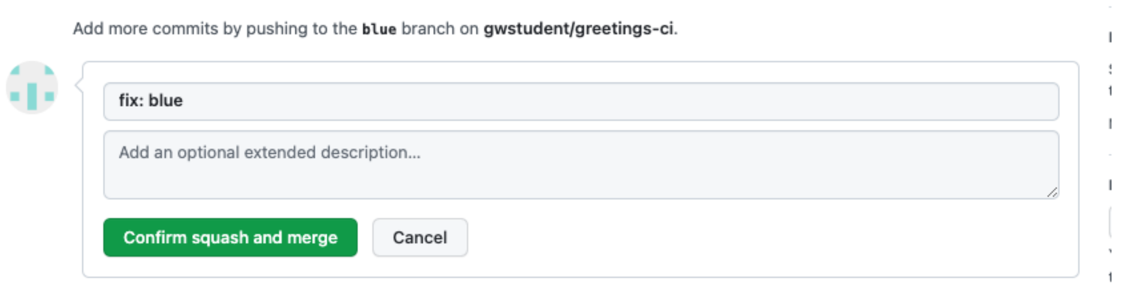
This branch has not been deployed
No deployments

Require approval from specific reviewers before merging
Ensure specific people or teams approve pull requests before they're merged into your main branch.

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

You can also open this in GitHub Desktop or view command line instructions.

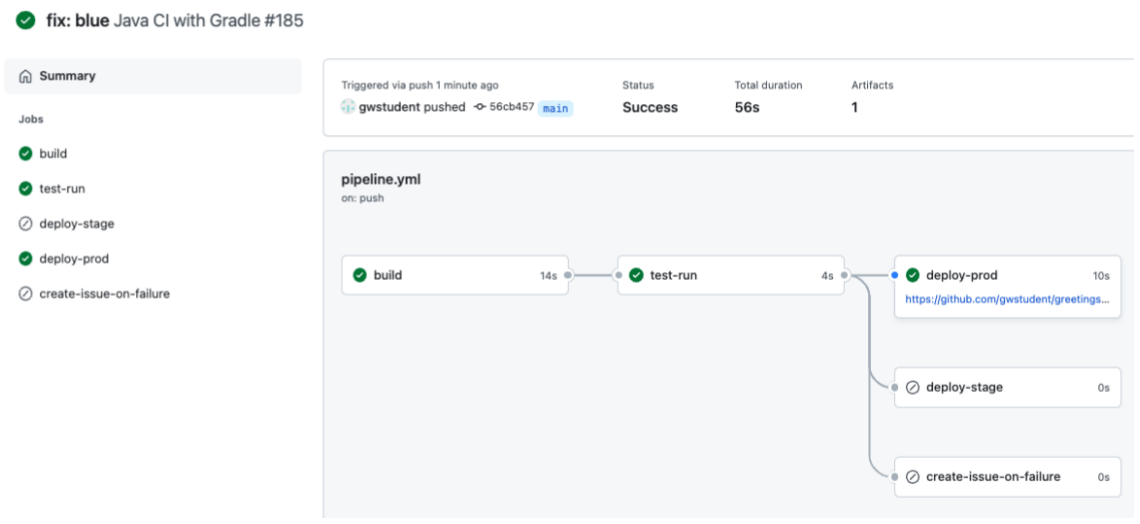
11. You can edit the main comment to have something like "fix: blue" in it and do what you want with the other commit messages. Then go ahead and click the "Confirm squash and merge" button.


Add more commits by pushing to the `blue` branch on **gwstudent/greetings-ci**.

fix: blue

Add an optional extended description...

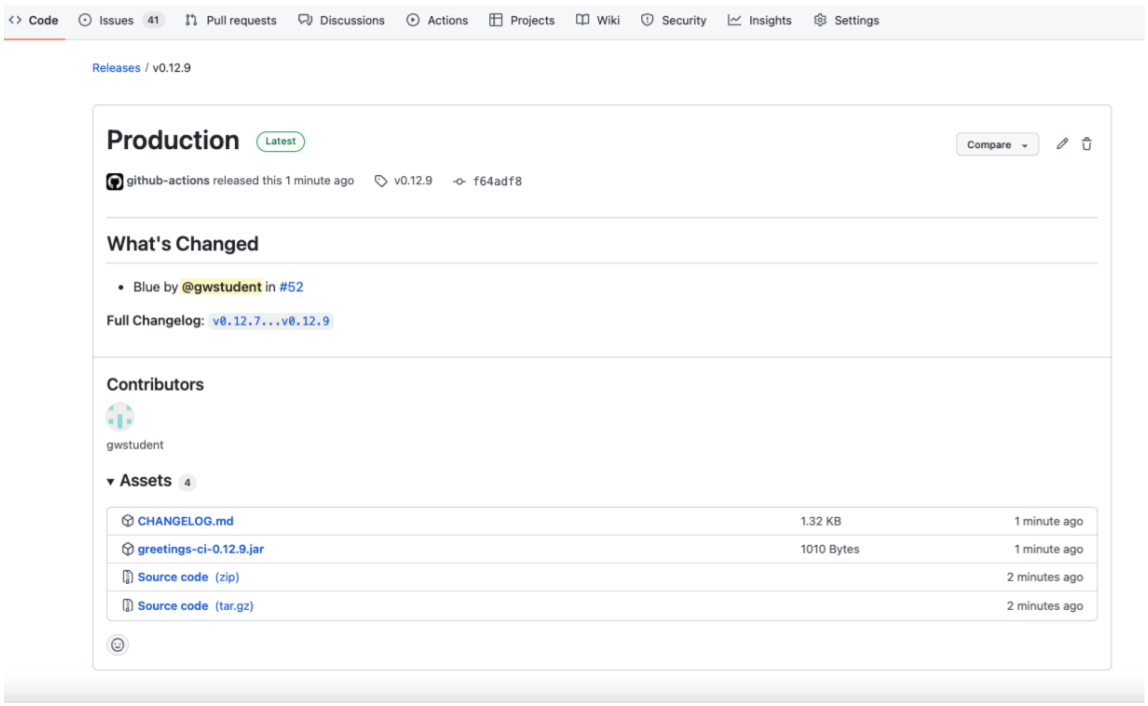
12. This should kick off another run of the action workflow in main. Because it runs in main, it should kick off the deploy-prod job.


fix: blue Java CI with Gradle #185

Triggered via push 1 minute ago: gwstudent pushed 56cb457 main
Status: Success
Total duration: 56s
Artifacts: 1

pipeline.yml
on: push

13. After this completes, you can click on the link in the "deploy-prod" box to see the release it created.



Releases / v0.12.9

Production

github-actions released this 1 minute ago v0.12.9 f64adf8

What's Changed

- Blue by @gwstudent in #52

Full Changelog: `v0.12.7...v0.12.9`

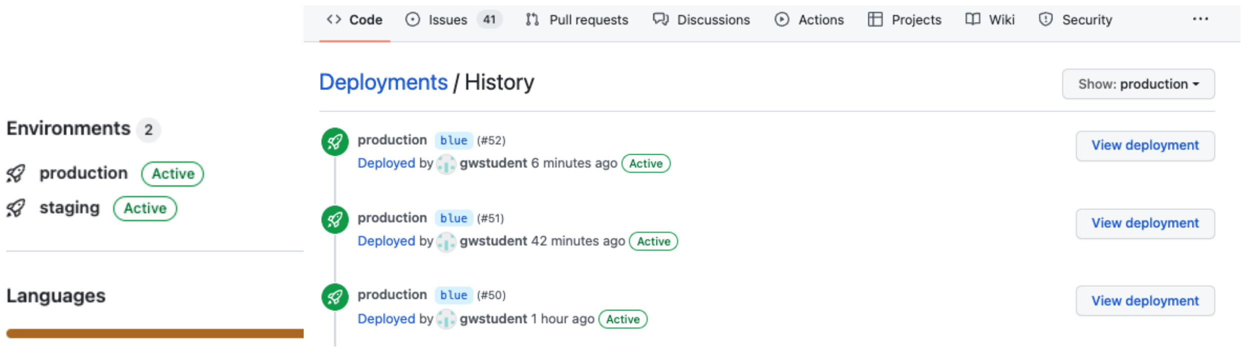
Contributors

gwstudent

Assets 4

CHANGELOG.md	1.32 KB	1 minute ago
greetings-ci-0.12.9.jar	1010 Bytes	1 minute ago
Source code (zip)		2 minutes ago
Source code (tar.gz)		2 minutes ago

14. You can also now see a Production environment available from the main repo page. You can click on it and see the deployments to production. Clicking on "View deployment" will take you to the same kind of page as the previous step did.



Deployments / History

Environments 2

production Active
staging Active

production blue (#52)
Deployed by gwstudent 6 minutes ago Active

production blue (#51)
Deployed by gwstudent 42 minutes ago Active

production blue (#50)
Deployed by gwstudent 1 hour ago Active

15. If you want, you can repeat the same exercise with a "green" branch to see how it works the same.

END OF LAB