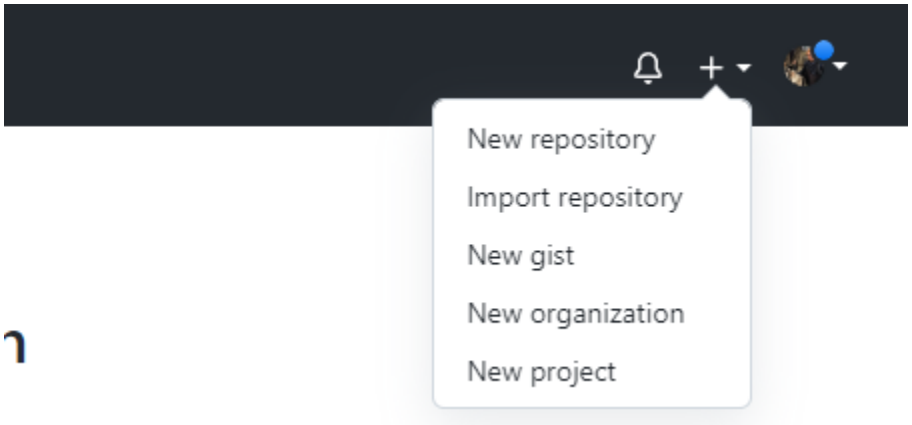
Exercise #1: GitHub Overview

1) Open “github.com” in a browser and click on “sign up” to create a new account or “sign in” if you already have an account.

2) Once you are logged in, it will take you to the main page where you will see your recent repositories to the left, and a feed with activity depending on the repositories that you follow. On the right you will see some GitHub adds and updates.

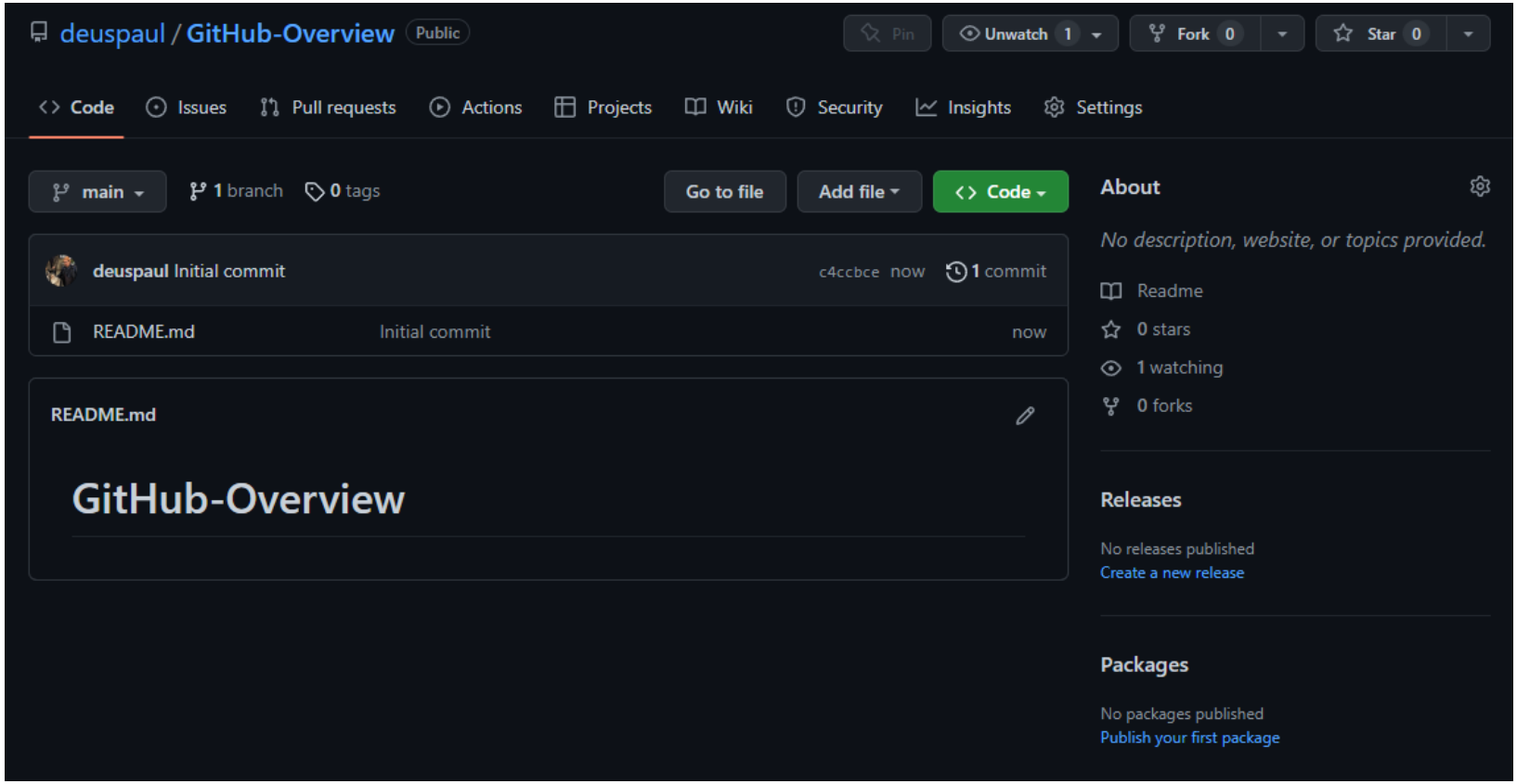
3) On the top part you have a search bar, a menu with a link to pull requests, Issues, Marketplace and Explore. The Marketplace is one of the links we will be using in this course, as that is where we will find some GitHub Actions that we can integrate into our projects.

4) At the top right you have notifications, a “+” button to create a new repository where you can host your code to collaborate with others, a gist which is basically just a snippet of code that you can share, an organization in case you want to set up a GitHub account for your team, or a project which allows you to plan and track work



5) Select “New repository”, in “Repository Name” input “GitHub Overview” and check the “Add a README file” checkbox so that your repository can be initialized with this “README” file. Leave all other settings with their default value and click on “Create repository”

6) Once your repository has been created, you will be taken to your repository:



7) At the top left you will see the “namespace” / “repository name”, its visibility (Public) and to the right the buttons to watch/fork and star. Watch is used to customize notifications for this repositors, fork is used to create a copy of this repository that you can freely make changes to as well as to push back into the parent repository where it was forked from, and star is used to add the repository to favorites.

8) Next, we have a menu that contains a link to the code, which is the main page of the repository. Here you will see everything related to the files of the repository, along with the branches, tags, a button to create/upload a file, as well as a button with the links to clone the repository with ssh, https, download, etc..

9) Next we have a button that takes you to a list of issues, followed by one that takes you to the pull requests. Actions which is what we will use the most in this lab. Projects to setup project tracking tools such as boards, wiki, to save documentation or information about the project. Security, which we will also use to setup code-scanning and dependabot, insights, which provides details about issues and pull requests. Finally, settings, which we will also use several times in the other labs of this lesson.

10) Click on “Actions”, since we have not setup a workflow, it will show GitHub actions that we can integrate into our repository. If you click on the “configure” button within the suggested “simple workflow”, it will help you set up a workflow in an editor.

deuspaul / GitHub-Overview Public

Pin | Unwatch 1 | Fork 0 | Star 0

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

GitHub-Overview / .github / workflows / blank.yml in main

Cancel changes | Start commit

Edit new file | Preview | Spaces | 2 | No wrap

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

name: CI

# Controls when the workflow will run
on:
  # Triggers the workflow on push or pull request events but only for the "main" branch
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

# A workflow run is made up of one or more jobs that can run sequentially or in parallel
jobs:
  # This workflow contains a single job called "build"
  build:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # Steps represent a sequence of tasks that will be executed as part of the job
```

Use Control + Space to trigger autocomplete in most situations.

Marketplace | Documentation

Search Marketplace for Actions

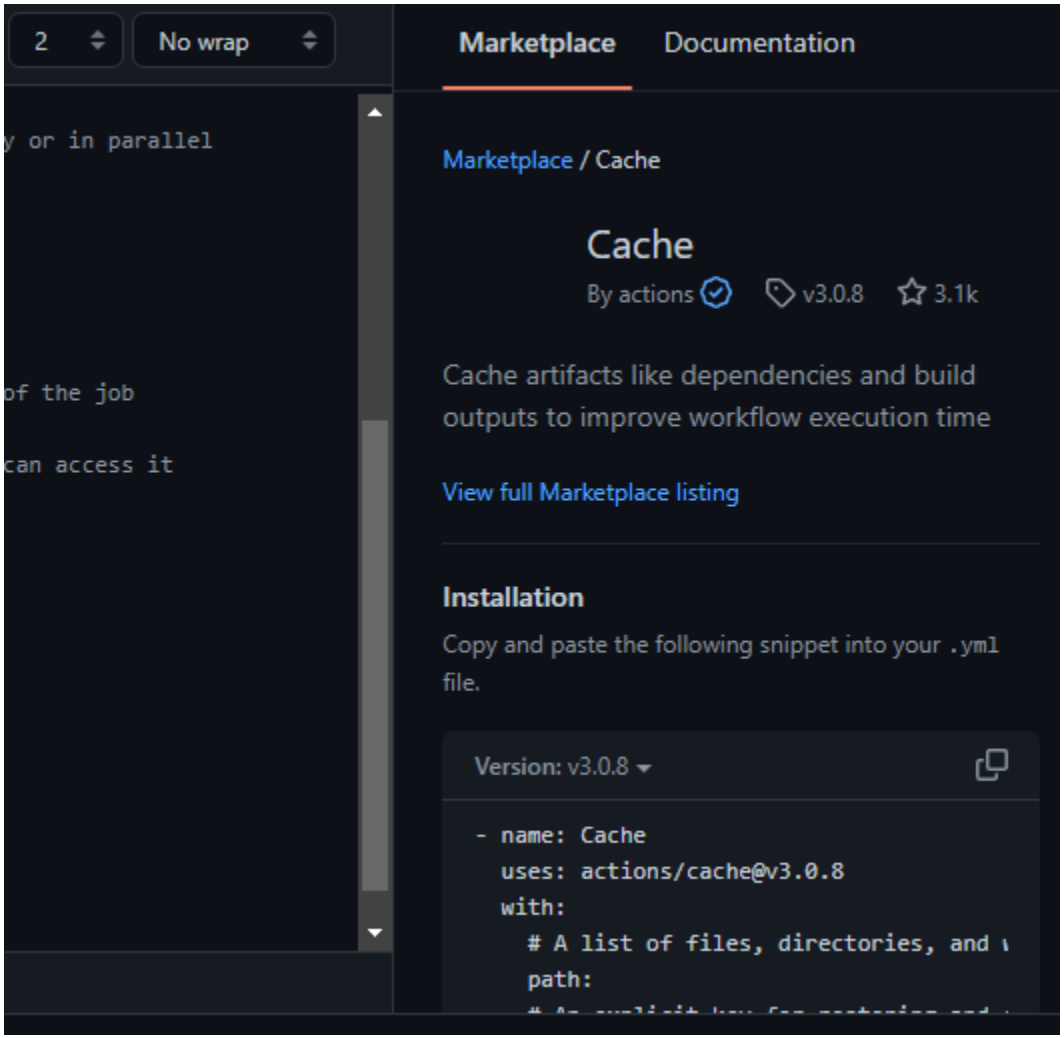
Featured Actions

Cache 3.1k
By actions
Cache artifacts like dependencies and build outputs to improve workflow execution time

Upload a Build Artifact 1.8k
By actions
Upload a build artifact that can be used by subsequent workflow steps

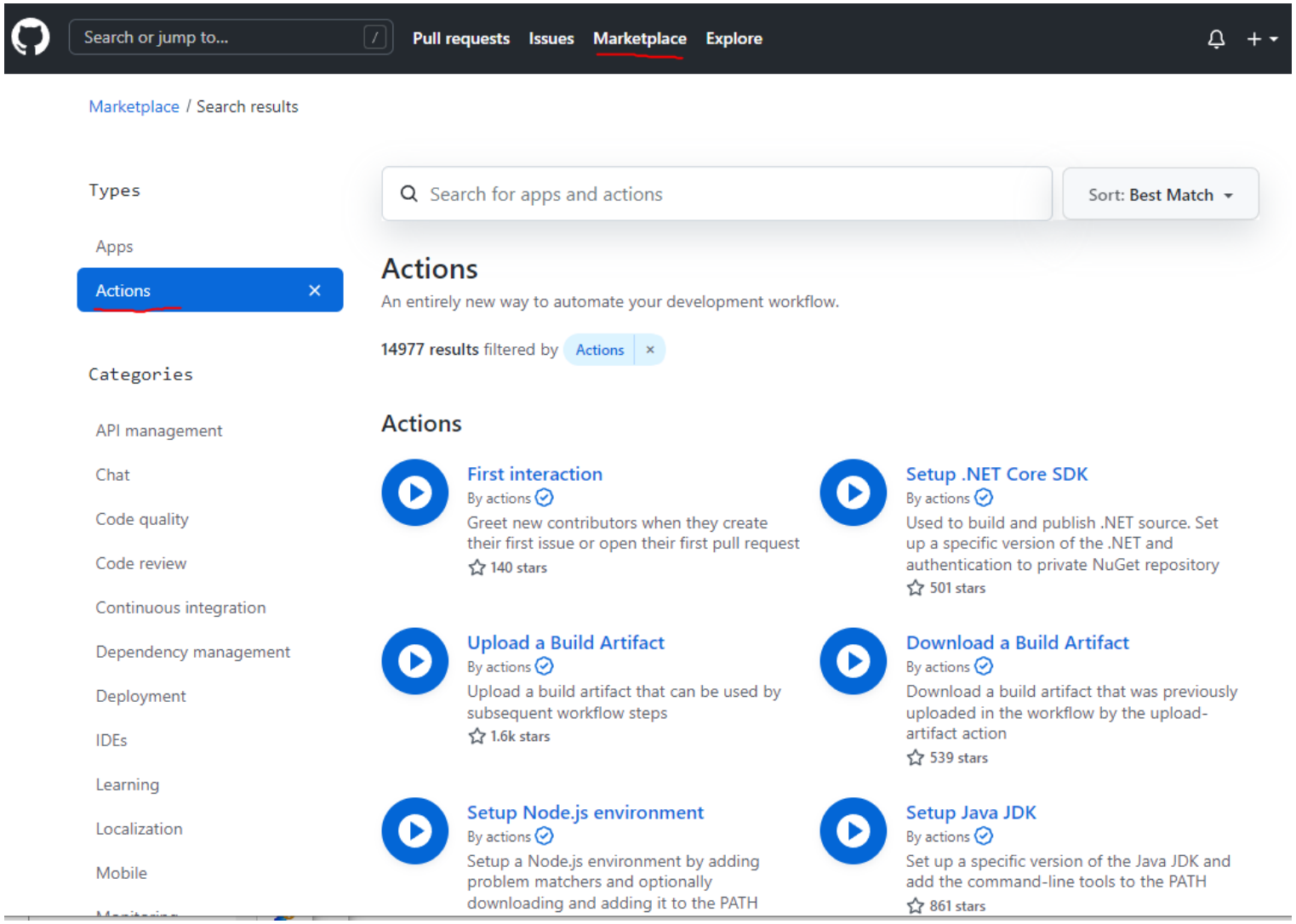
Setup Go environment 867
By actions
Setup a Go environment and add it to the PATH

11) To the right, you will notice some action from the Marketplace that you can integrate into your workflow. Upon clicking on them, you should be able to see the instructions about their usage.



12) For now, its not necessary to click on the green “start commit” button, as this was just an overview.

13) Another place where you can locate GitHub actions is directly in the marketplace. To access it, click on the “Marketplace” link at the very top. You can filter the results by clicking on Types > Actions



14) Upon clicking on one of the actions, you will be taken to information page about this action, which provides you with instructions on how it can be used along with different versions of the GitHub action:

Marketplace / Actions / First interaction

GitHub Action

First interaction

v1.1.0 Latest version

Use latest version

Choose a version

v1.1.0

v1.0.0
v1.0.0

First Interaction

An action for filtering pull requests and issues from first-time contributors.

created by **actions**.

Learn more about verified Actions.

Usage

See action.yml

```
steps:
- uses: actions/first-interaction@v1
  with:
    repo-token: ${{ secrets.GITHUB_TOKEN }}
    issue-message: '# Message with markdown.\nThis is the message that will be displayed on
    pr-message: 'Message that will be displayed on users' first pr. Look, a `code block` fo
```

Stars

Star 467

Contributors

Categories

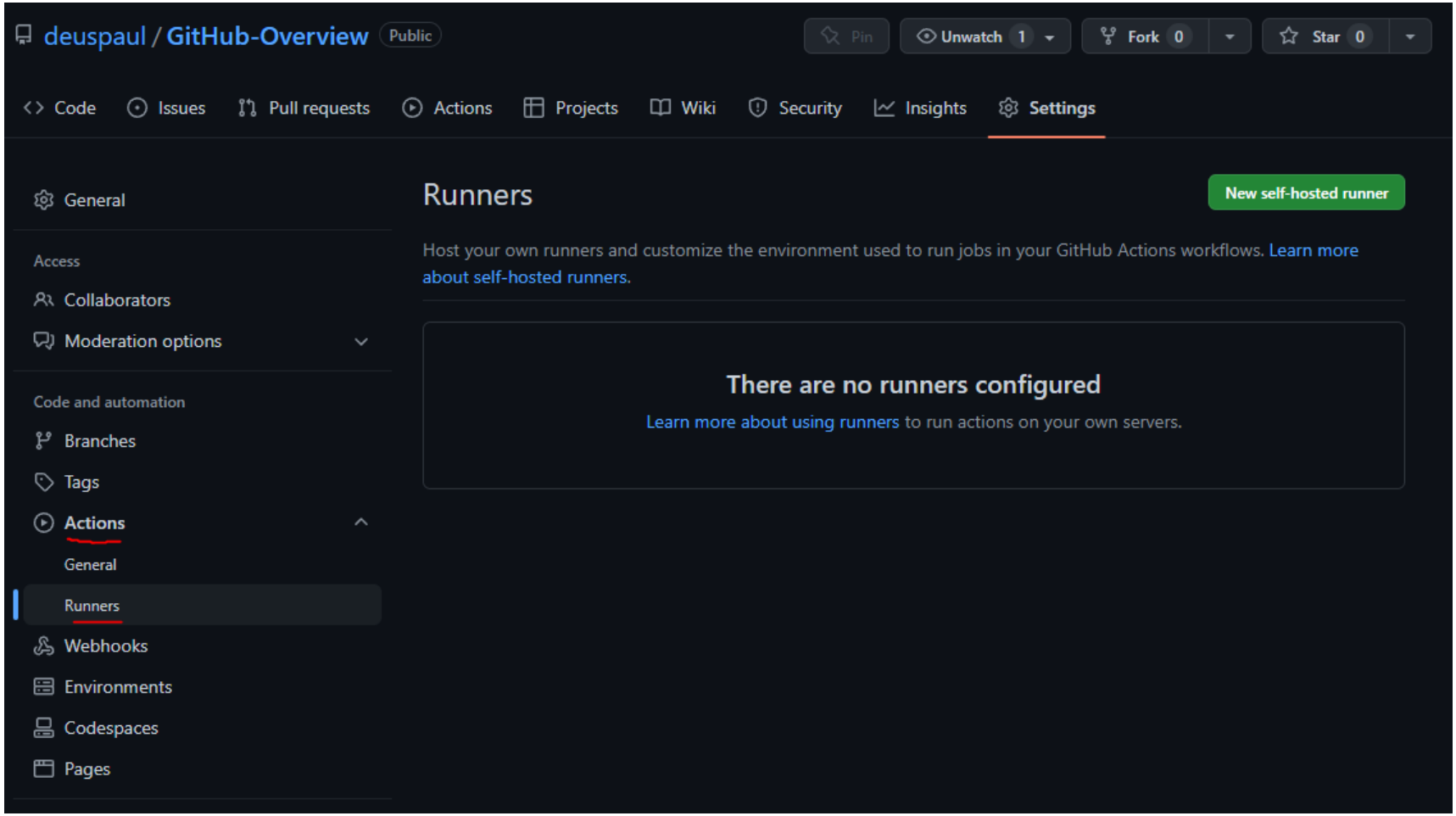
Utilities

License

Links

Exercise #2: Setup a runner and a workflow

1) Go back to the repository we created in the previous exercise and click on the “settings” tab. Then within the “Code and automation” section from the left menu, click on “Actions” > “runners” and then click on the green “New self-hosted runner” button at the top right.:



2) Select your Operating System in “Runner image” and your architecture (x64/arm)

3) Follow the steps listed below architecture to download and install the runner (Hint: you can copy the commands by clicking on them)

4) Example for Windows:
Open Terminal or Cmd/Powershell **as administrator** and create a folder called “actions-runner” in the drive root and open that

directory:

```
PS C:\Users\CircuitZero> mkdir c:\actions-runner


    Directory: C:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         9/20/2022   9:45 AM                actions-run
                                                  ner


PS C:\Users\CircuitZero> cd c:\actions-runner
PS C:\actions-runner>
```

5) Download the latest runner package:

```
PS C:\actions-runner> Invoke-WebRequest -Uri https://github.com/actions/runner/releases/download/v2.296
.2/actions-runner-win-x64-2.296.2.zip -OutFile actions-runner-win-x64-2.296.2.zip
PS C:\actions-runner> ls


    Directory: C:\actions-runner


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         9/20/2022   9:57 AM       72506232 actions-runner-win-x64-2.296.2.zip
```

6) Check the integrity of the downloaded file to make sure its legitimate (if it has been compromised it will display the message saying "Computed checksum did not match"):

```
PS C:\actions-runner> if((Get-FileHash -Path actions-runner-win-x64-2.296.2.zip -Algorithm SHA256).Hash
.ToUpper() -ne '96d03cf54dbfe2e016bd2aa5a08ffbd2a803b1899b0ae3eedf4bd18e370f14a4'.ToUpper()){ throw 'Co
mputed checksum did not match' }
PS C:\actions-runner>
```

7) Extract the file contents to the current location:

```
PS C:\actions-runner> Add-Type -AssemblyName System.IO.Compression.FileSystem ; [System.IO.Compression.
ZipFile]::ExtractToDirectory("$PWD/actions-runner-win-x64-2.296.2.zip", "$PWD")
PS C:\actions-runner> ls


    Directory: C:\actions-runner


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         9/20/2022  10:01 AM                bin
d-----         9/20/2022  10:01 AM                externals
-a----         9/20/2022   9:57 AM       72506232 actions-runner-win-x64-2.296.2.zip
-a----          9/8/2022   5:39 PM           1225 config.cmd
-a----          9/8/2022   5:39 PM           1539 run-helper.cmd.template
-a----          9/8/2022   5:39 PM           2146 run-helper.sh.template
-a----          9/8/2022   5:39 PM           1106 run.cmd


PS C:\actions-runner>
```

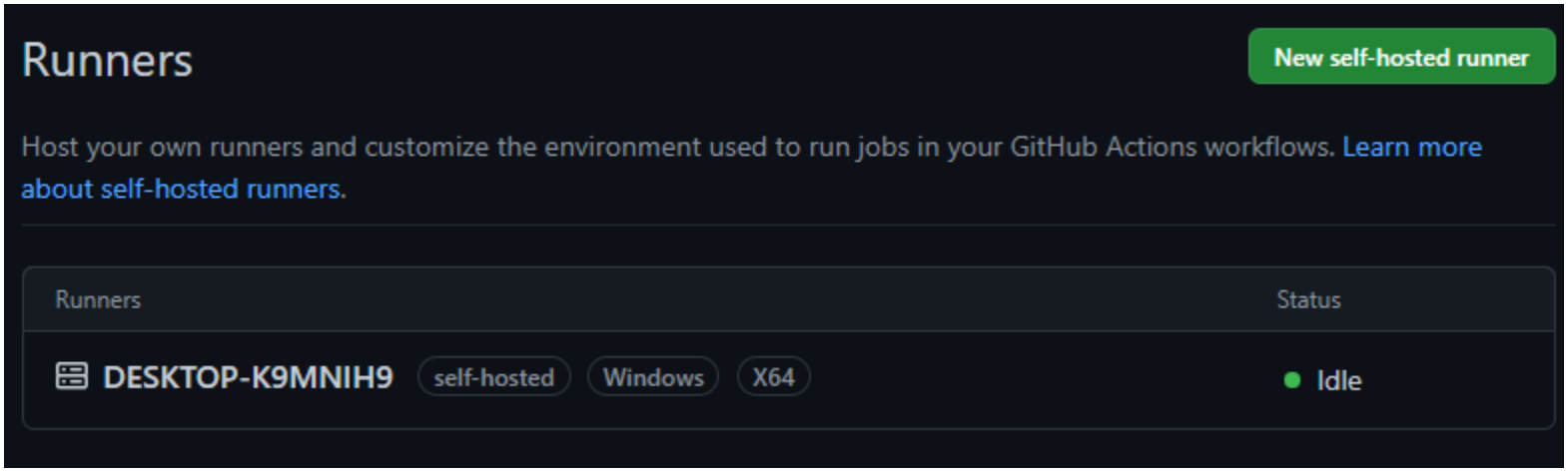
8) Register the runner to your github repository with the first command in the "Configure" section. You can leave the default values for name, labels and work folder or change them

when prompted if you would like to run it as a service select “y”, and select the default account to use for the service (NT Authority\Network service):

Administrator: Windows PowerShell

```
|      | |  _| | __| |_| | | | | '_ \    / _ \ / __| __| |/ _ \| '_ \/ __|     |
|      | |_| | | |_|   _  | |_| | |_) |  / ___ \ (__| |_| | (_) | | | \__ \     |
|       \____|_|\__|_| |_|\__,_|_.__/  /_/   \_\___|\__|_|\___/|_| |_|___/     |
|                                                                               |
|                       Self-hosted runner registration                         |
|                                                                               |
--------------------------------------------------------------------------------

# Authentication


√ Connected to GitHub

# Runner Registration

Enter the name of the runner group to add this runner to: [press Enter for Default]

Enter the name of runner: [press Enter for DESKTOP-K9MNIH9]

This runner will have the following labels: 'self-hosted', 'Windows', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]

√ Runner successfully added
√ Runner connection is good

# Runner settings

Enter name of work folder: [press Enter for _work]

√ Settings Saved.

Would you like to run the runner as service? (Y/N) [press Enter for N] y
User account to use for the service [press Enter for NT AUTHORITY\NETWORK SERVICE]
Granting file permissions to 'NT AUTHORITY\NETWORK SERVICE'.
Service actions.runner.deuspaul-GitHub-Overview.DESKTOP-K9MNIH9 successfully installed
Service actions.runner.deuspaul-GitHub-Overview.DESKTOP-K9MNIH9 successfully set recovery option
Service actions.runner.deuspaul-GitHub-Overview.DESKTOP-K9MNIH9 successfully set to delayed auto star
t
Service actions.runner.deuspaul-GitHub-Overview.DESKTOP-K9MNIH9 successfully configured
Waiting for service to start...
Service actions.runner.deuspaul-GitHub-Overview.DESKTOP-K9MNIH9 started successfully
```

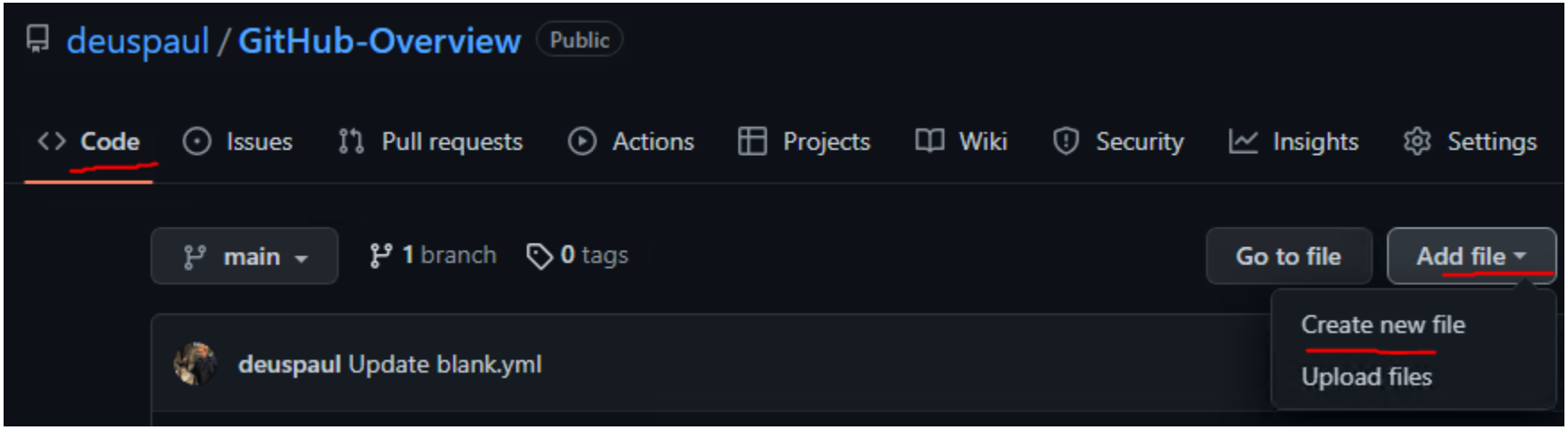
9) Once you have completed the previous step, your runner should appear as online/idle



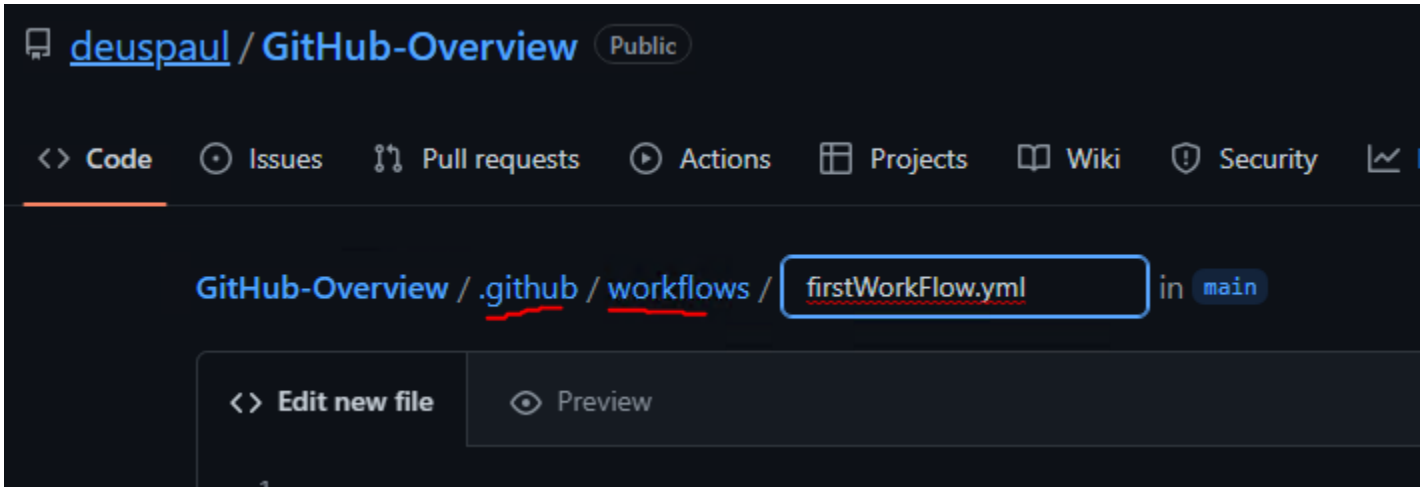
10) If it appears as offline, you may need to run the "./run.cmd" command

```
PS C:\actions-runner> .\run.cmd
        1 file(s) copied.

√ Connected to GitHub
```

11) Now we are ready to test our runner. Click on “Code”, then on the “Add file” button and select “Create new file”



12) At the top part, type “.github/”, then “workflows/”, and then “firstWorkFlow.yml”:



Add the following code to the file, mind the 2 spaces between each indent:

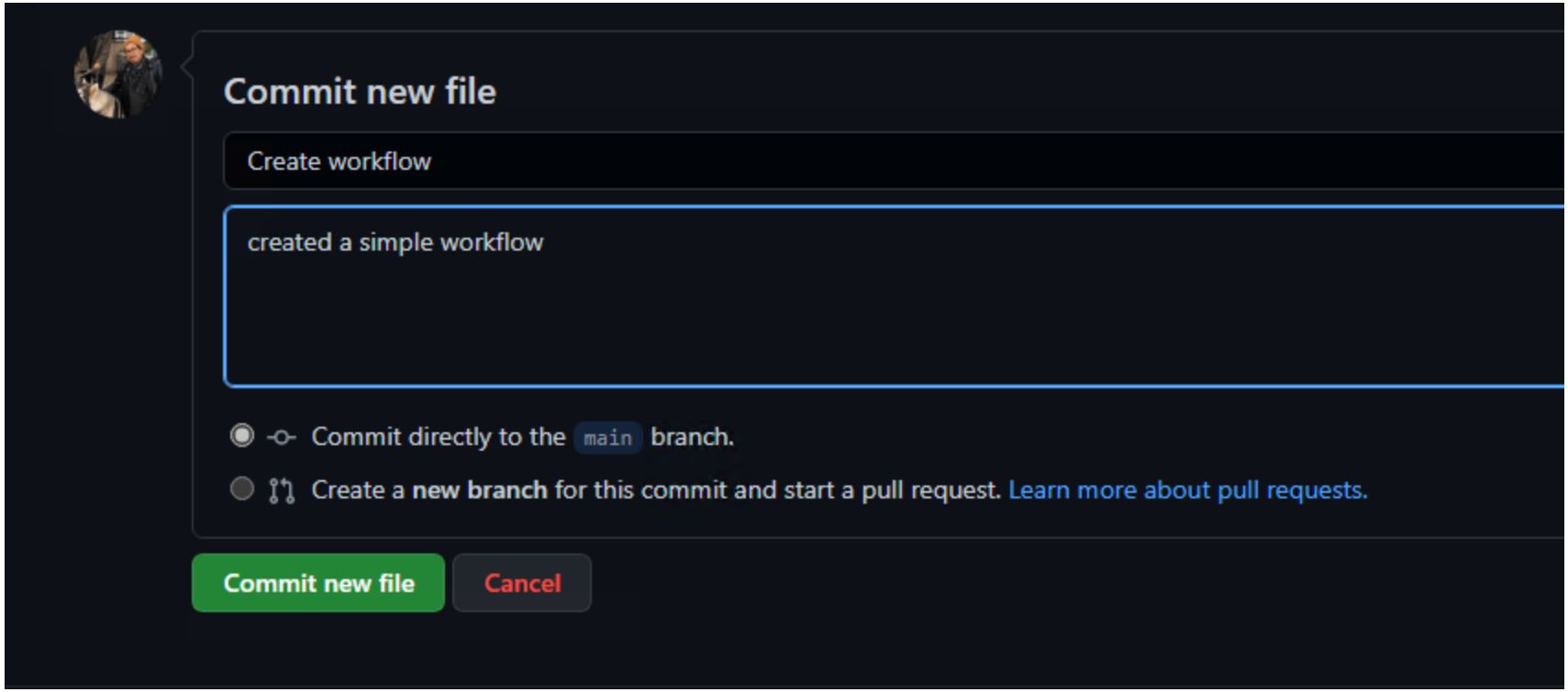
GitHub-Overview / .github / workflows / firstWorkFlow.yml in main

Edit new file | Preview

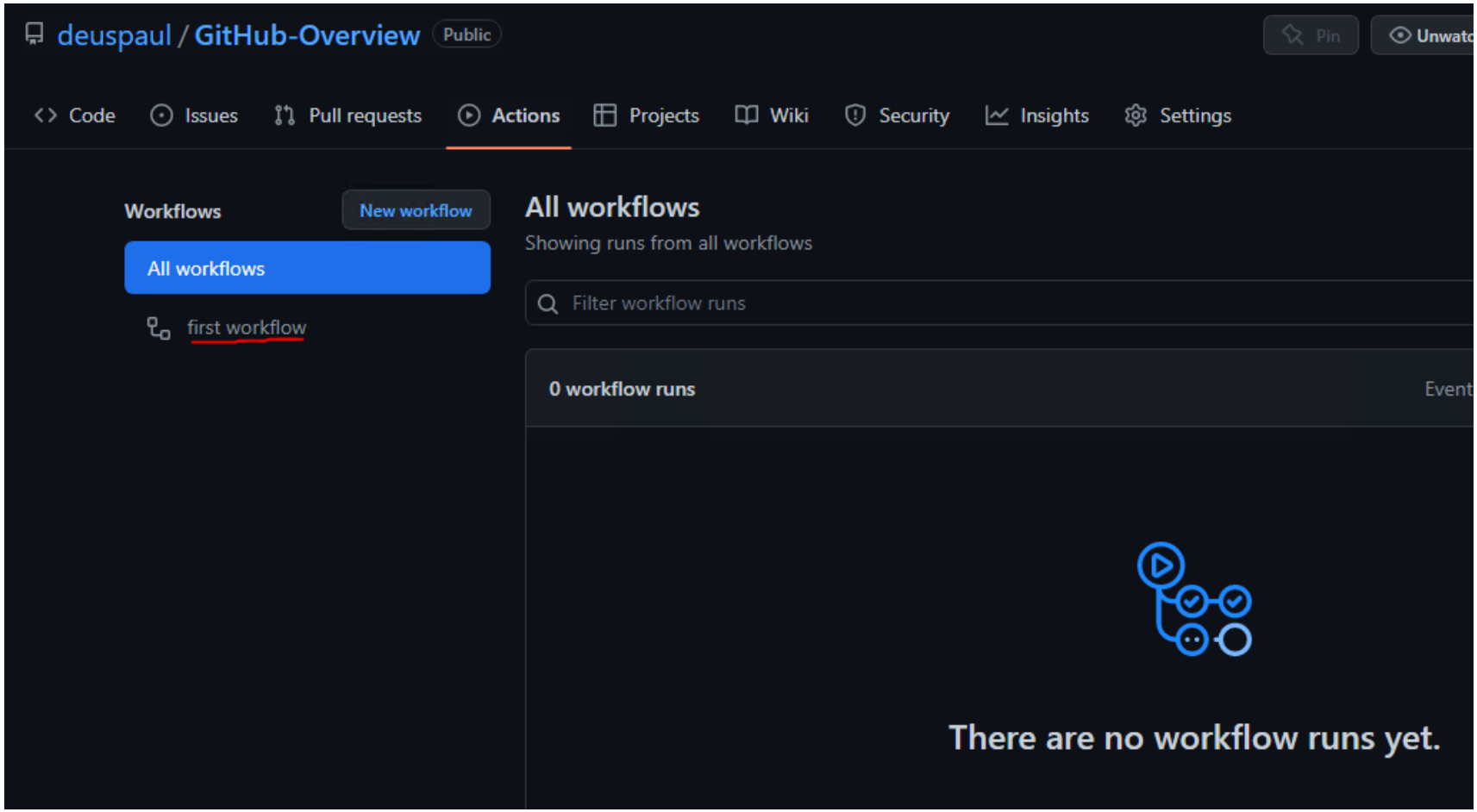
```yaml
name: first workflow

on:
  workflow_dispatch:

jobs:
  firstjob:
    runs-on: self-hosted
    steps:
      - name: firstJob script
        shell: cmd
        run: echo Hello World! This is the first job

  secondjob:
    runs-on: ubuntu-latest
    steps:
      - name: secondJob script
        shell: bash
        run: |
          echo this is the second job
          echo and this is a multi-line script
```

```yaml
name: first workflow

on:
  workflow_dispatch:

jobs:
  firstjob:
    runs-on: self-hosted
    steps:
      - name: firstJob script
        shell: cmd
        run: echo Hello World! This is the first job

  secondjob:
    runs-on: ubuntu-latest
    steps:
      - name: secondJob script
        shell: bash
        run: |
          echo this is the second job
          echo and this is a multi-line script
```

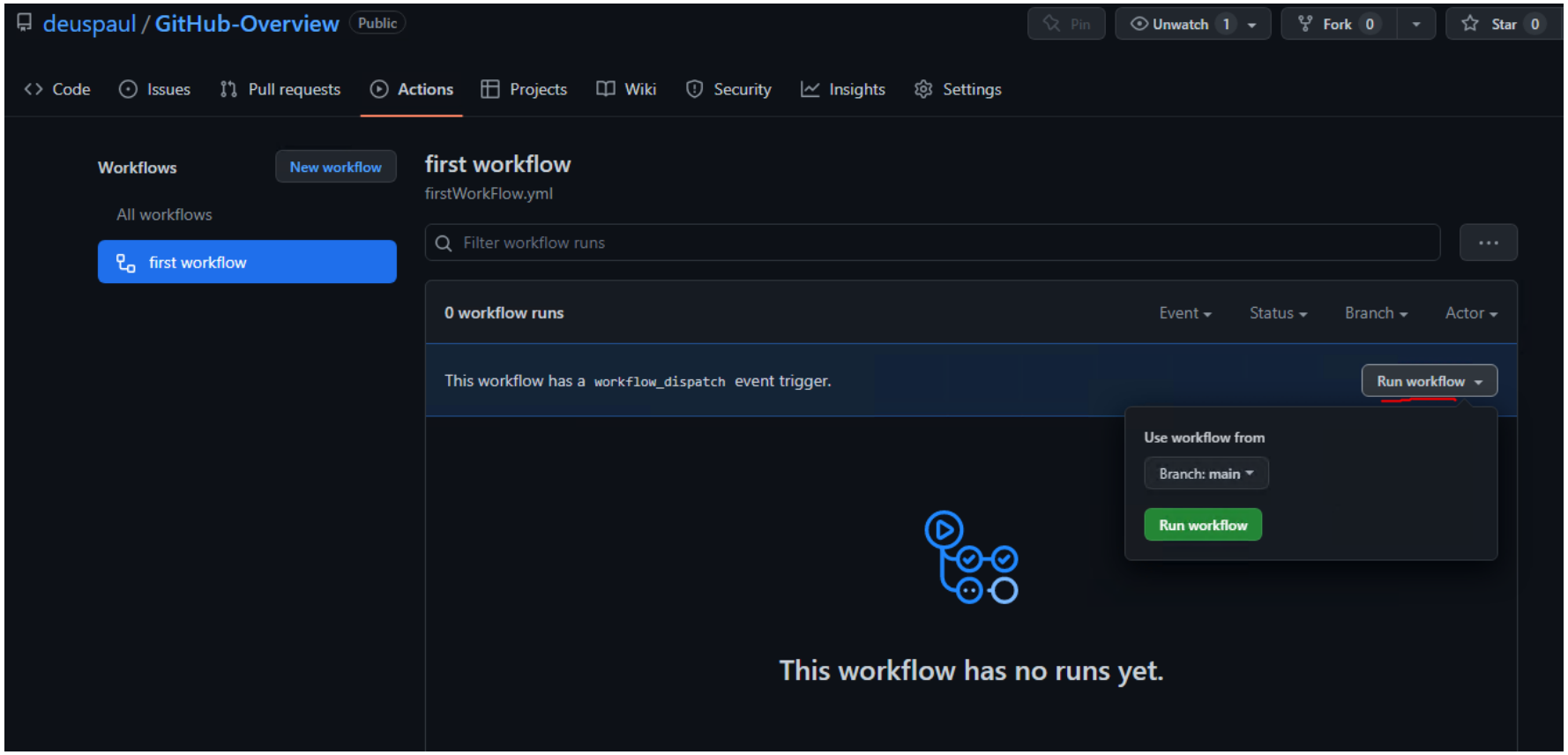
13) Scroll to the bottom and add a descriptive message and click on “commit new file”



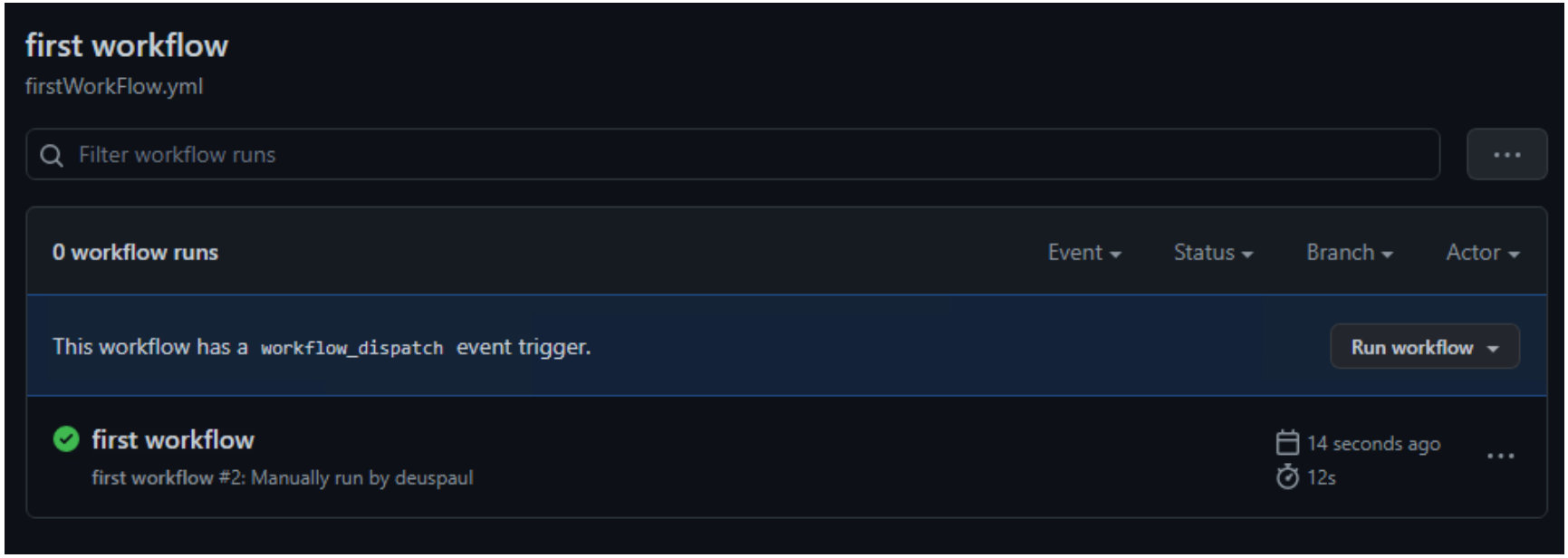
14) Click on “Actions”, you should see your workflow listed there:



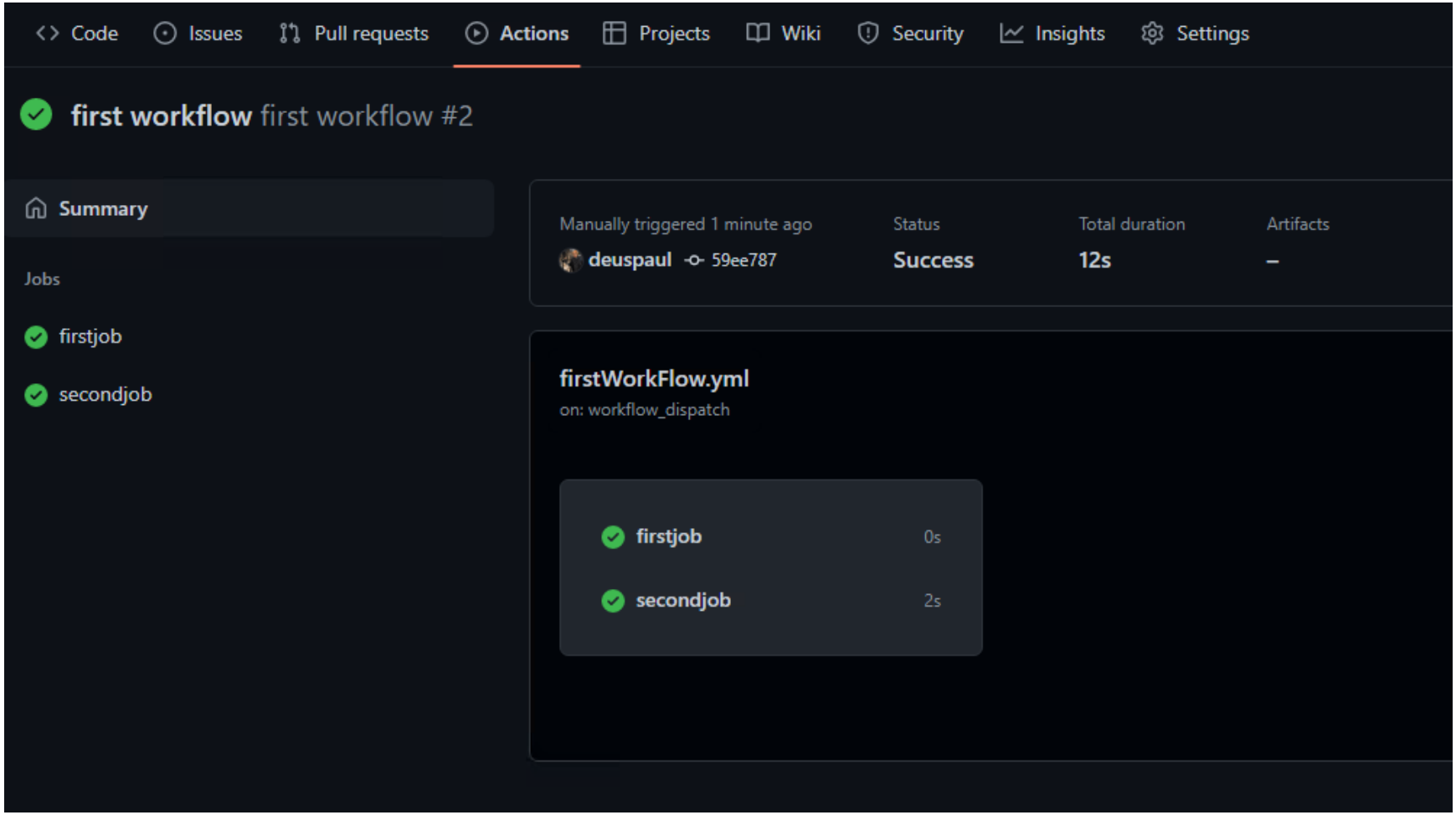
15) Next, select your workflow and click on “run workflow”, select “main” branch, and click on “run workflow”



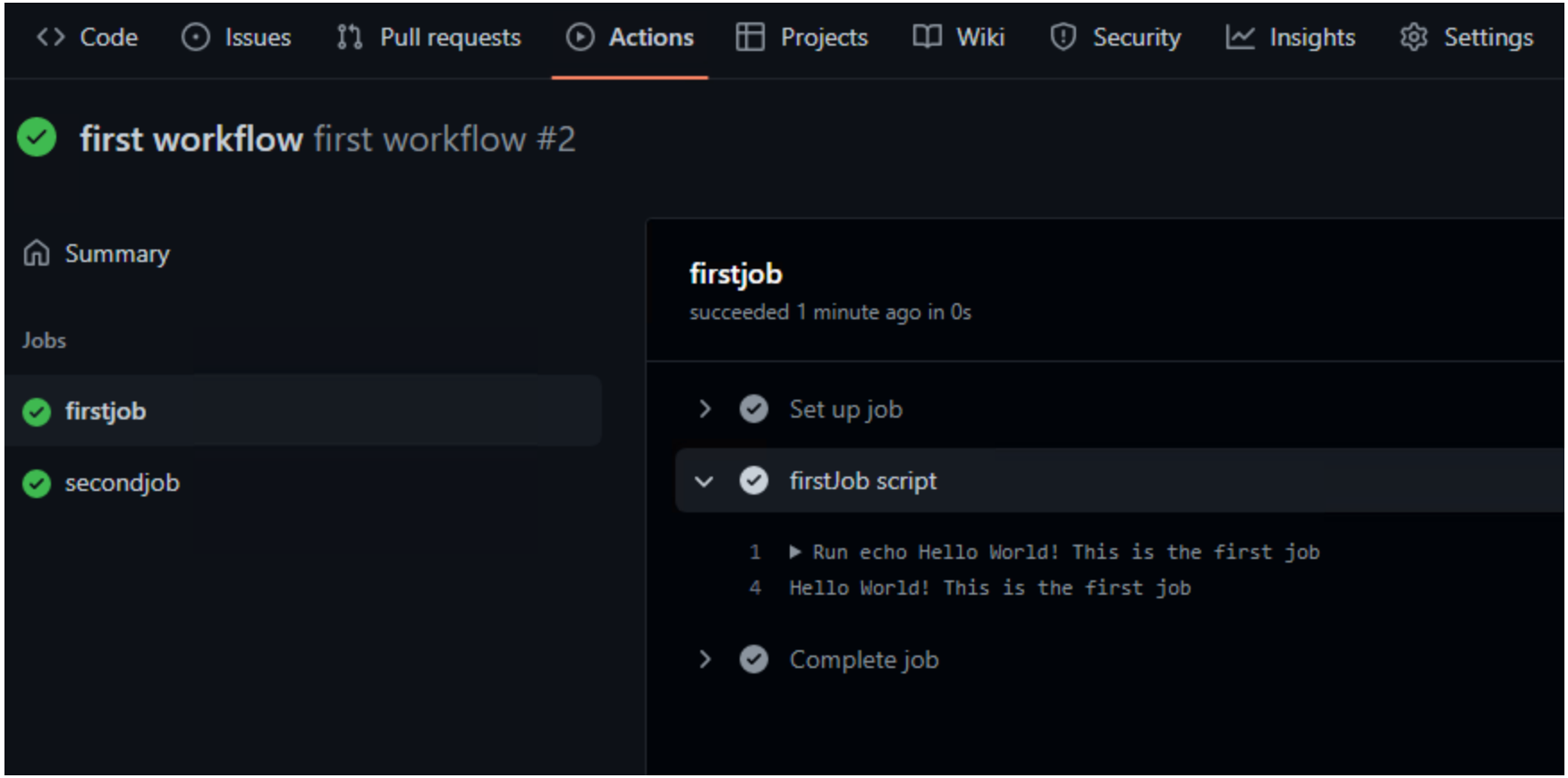
16) After a couple of seconds, the job will be launched, and you can see its details by clicking on it:

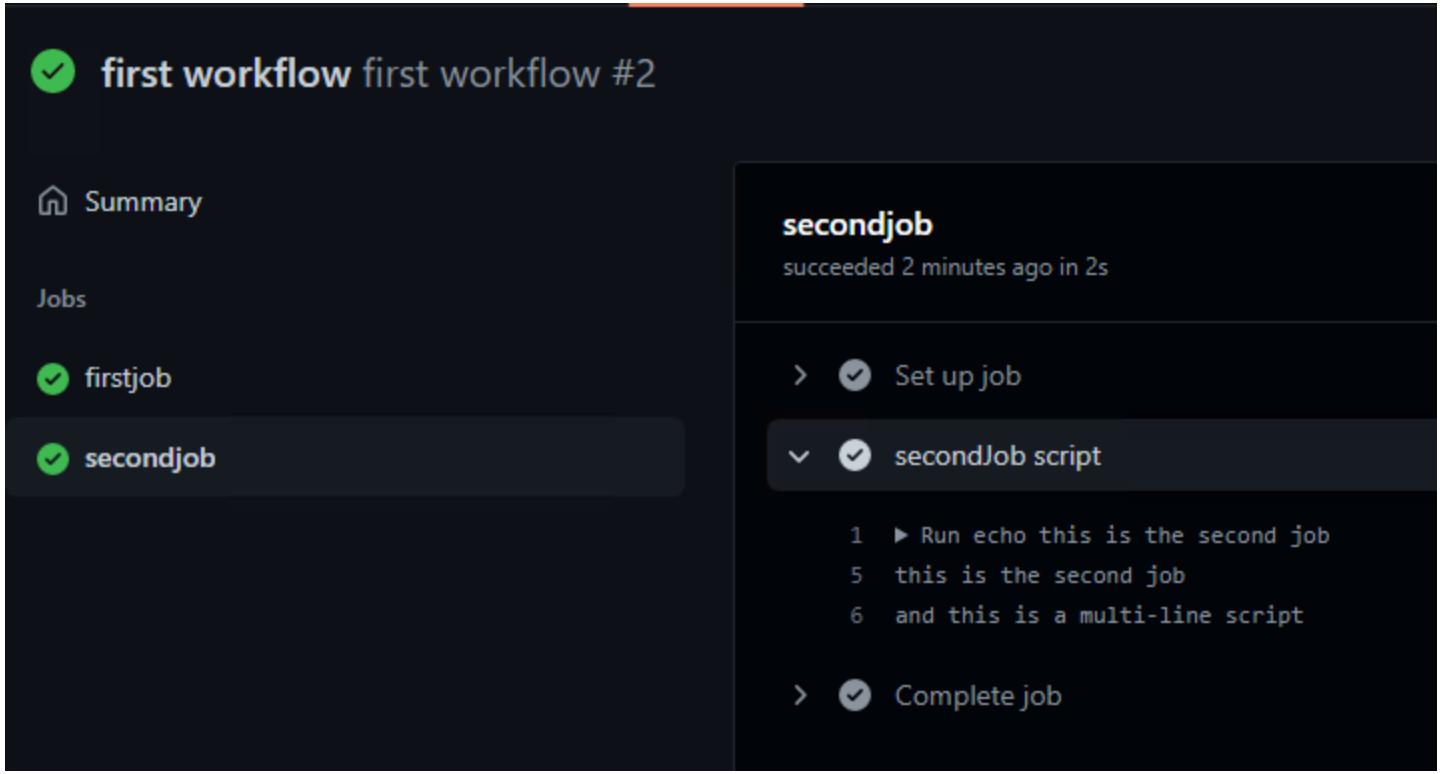


17) Here you can see it is a workflow made up of 2 jobs

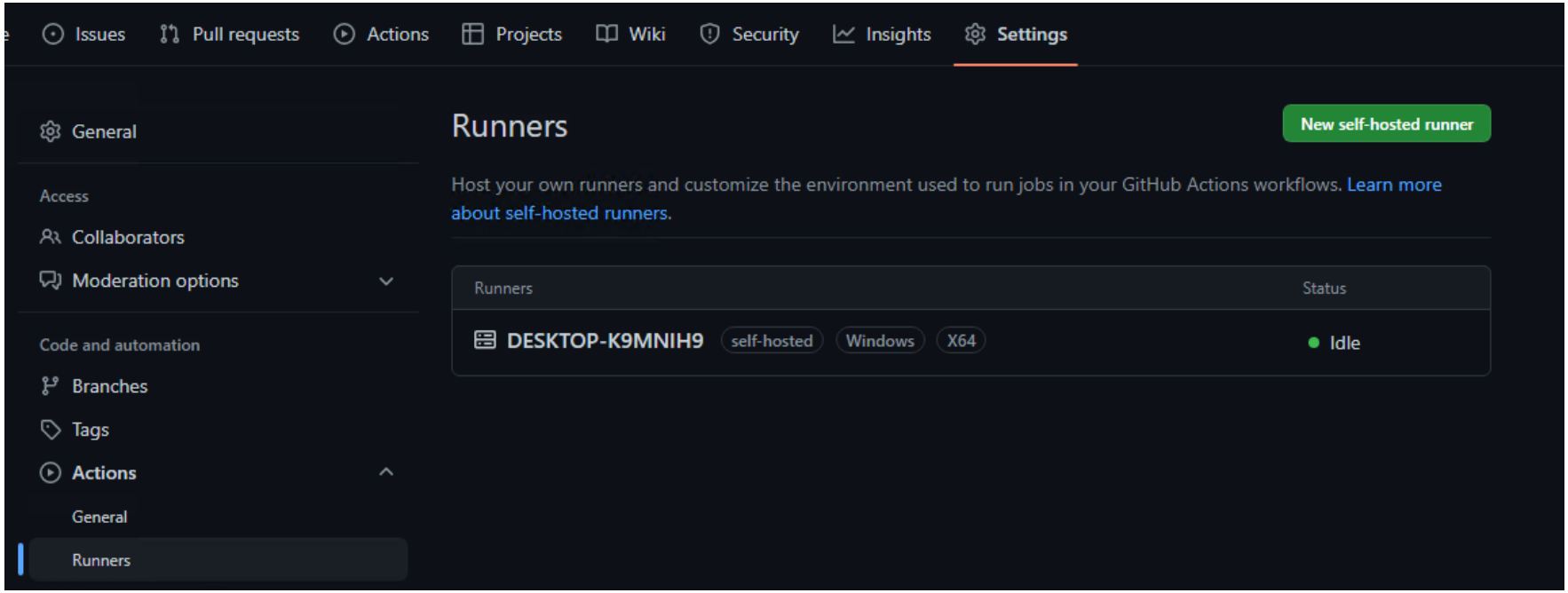


18) Upon clicking on each job, you can see the steps within:


Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
first workflow first workflow #2
Summary
Jobs
firstjob
secondjob
firstjob
succeeded 1 minute ago in 0s
Set up job
firstJob script
1 ▶ Run echo Hello World! This is the first job
4 Hello World! This is the first job
Complete job

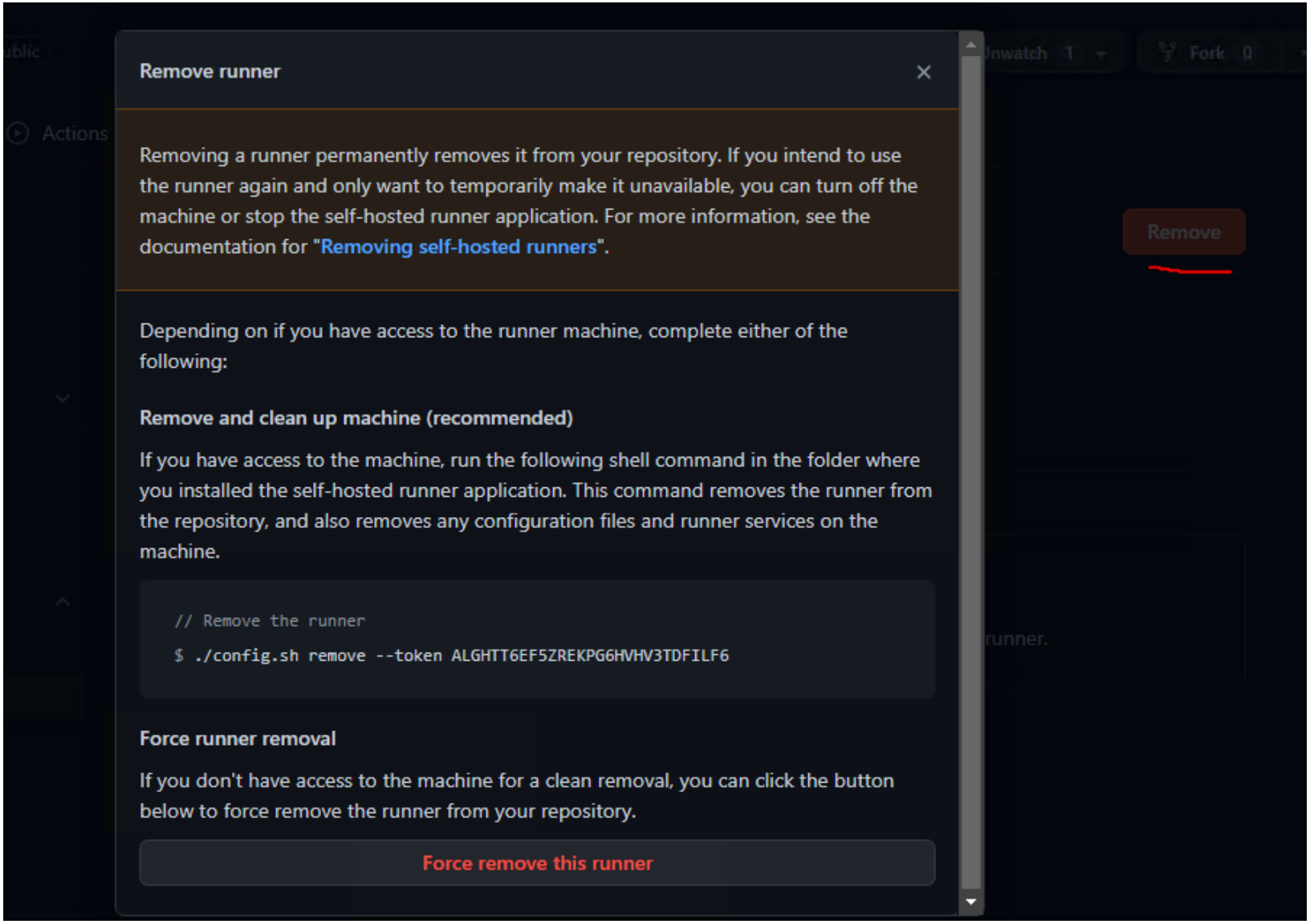


19) Now lets go ahead and remove the self-hosted runner. Click on “settings”, “actions” > “runners” and click on your runner



20) Click on the remove button, that should show the following pop-up. Ideally you should remove the runner with the command shown in the pop-up (change “config.sh” to “config.cmd” in case of windows), or you can

Also click on the “force remove this runner button”



Exercise #3: Advanced workflow

1) Go back to our repository from the previous exercise and click on the code tab. Access the ".github" and "workflows" directories and click on our workflow "firstWorkflow. Next, click on the pencil icon to the right to edit the workflow file:

deuspaul / GitHub-Overview Public

Pin | Unwatch 1 | Fork 0 | Star

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main | GitHub-Overview / .github / workflows / firstWorkFlow.yml | View runs | Go to file | ...

deuspaul Create workflow ... | Latest commit 59ee787 3 hours ago | History

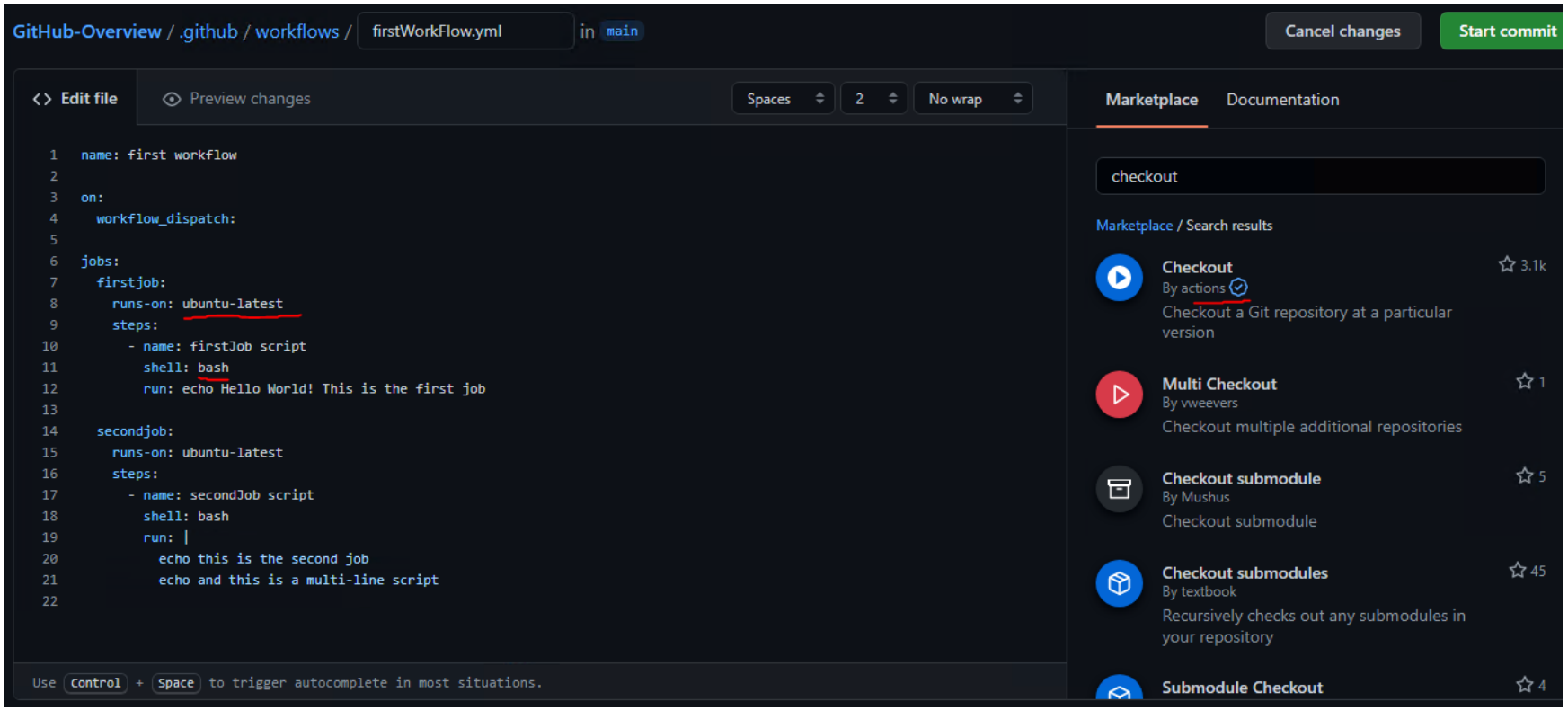
1 contributor

21 lines (18 sloc) | 415 Bytes | Raw | Blame

```yaml
name: first workflow

on:
  workflow_dispatch:

jobs:
  firstjob:
    runs-on: self-hosted
    steps:
      - name: firstJob script
        shell: cmd
        run: echo Hello World! This is the first job

  secondjob:
    runs-on: ubuntu-latest
    steps:
      - name: secondJob script
        shell: bash
        run: |
          echo this is the second job
          echo and this is a multi-line script
```

2) The first thing we will want to do is to change the runner for the first job, from “self-hosted” to “ubuntu-latest” along with the shell of the job from “cmd” to “bash”.

3) Next, we will add some GitHub actions to it. On the right side you will notice that there is a list of several GitHub actions from the marketplace. Search “checkout” and locate an action by “actions” with the blue checkmark indicating a verified creator and click on it:



4) It will display its usage instructions, though it’s a little bit crowded for that window, so go ahead and click on the “view full Marketplace listing” link instead.

5) In the full marketplace listing page, it shows you the description for the selected version, the new features for this version as well as the usage and some example scenarios. If you would like to change to a previous version, you can do so by clicking on the green button at the top right labeled “Use latest version”.

This is one of the most used functions as it clones your GitHub repository to the runner so you can work with your code

GitHub Action

Checkout

v2.4.2 Latest version

Use latest version

test-local passing

Checkout V3

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

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

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

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

What's new

- Updated to the node16 runtime by default
  - This requires a minimum Actions Runner version of v2.285.0 to run, which is by default

Verified creator

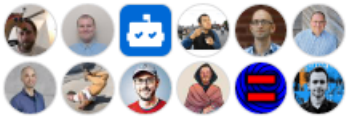
GitHub has verified that this action was created by **actions**.

Learn more about verified Actions.

Stars

Star 3.1k

Contributors



Categories

Utilities

Links

actions/checkout

6) Go back to the tab where we are editing the code of your workflow and add this action to a step

```
- name: checkout
  uses: actions/checkout@v3
```

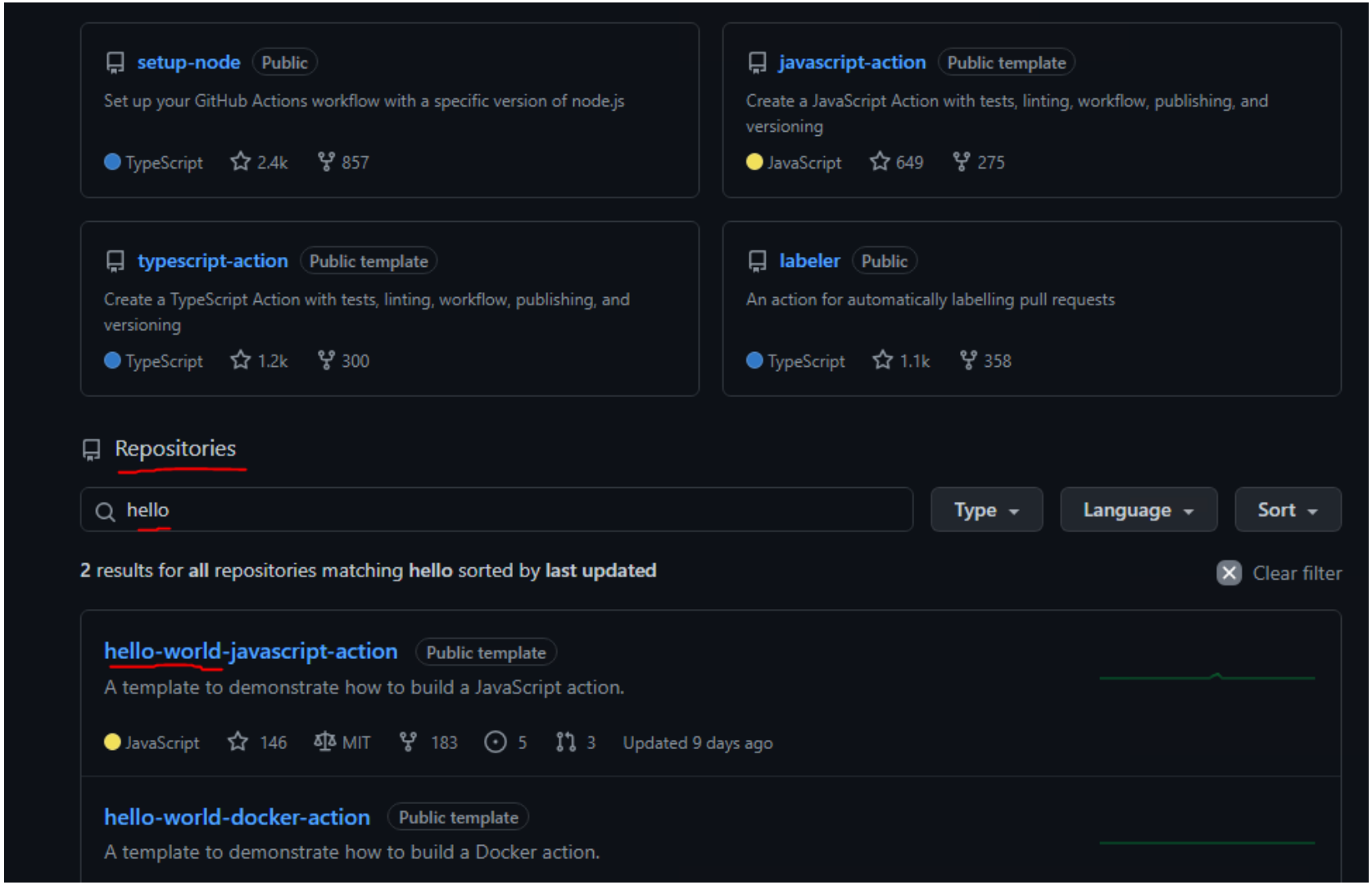
GitHub-Overview / .github / workflows / firstWorkFlow.yml in main

Edit file | Preview changes

```yaml
name: first workflow

on:
  workflow_dispatch:

jobs:
  firstjob:
    runs-on: ubuntu-latest
    steps:
      - name: firstJob script
        shell: bash
        run: echo Hello World! This is the first job
      - name: checkout
        uses: actions/checkout@v3

  secondjob:
    runs-on: ubuntu-latest
    steps:
      - name: secondJob script
        shell: bash
        run: |
          echo this is the second job
          echo and this is a multi-line script
```

7) Next, lets add another action that we can interact with, this time we will use an action from a repository instead of an action from the marketplace. In another tab open https://github.com/actions, and in the repositories search bar, type “hello”, and click on the “hello-

world-javascript-action”



8) At the bottom of the repository, you will find the instructions to use this action under the “Example usage” section. Make note of the output section as well, this means that this action has an output, which in this case is the time in which the action ran:

Hello world JavaScript action

This action prints "Hello World" or "Hello" + the name of a person to greet to the log. To learn how this action was built, see "Creating a JavaScript action" in the GitHub Help documentation.

Inputs

`who-to-greet`

Required The name of the person to greet. Default `"World"`.

Outputs

`time`

The time we greeted you.

Example usage

```
uses: actions/hello-world-javascript-action@main
with:
  who-to-greet: 'Mona the Octocat'
```

9) Copy the code from the “Example usage” code block, add a new step with name “greetings” and paste the code that we copied into this step. Add a key for “id” to this step with the value “greetings” so we can reference its output in other steps.

```
- name: greetings
  id: greetings
  uses: actions/hello-world-javascript-action@main
  with:
    who-to-greet: 'Mona the Octocat'
```

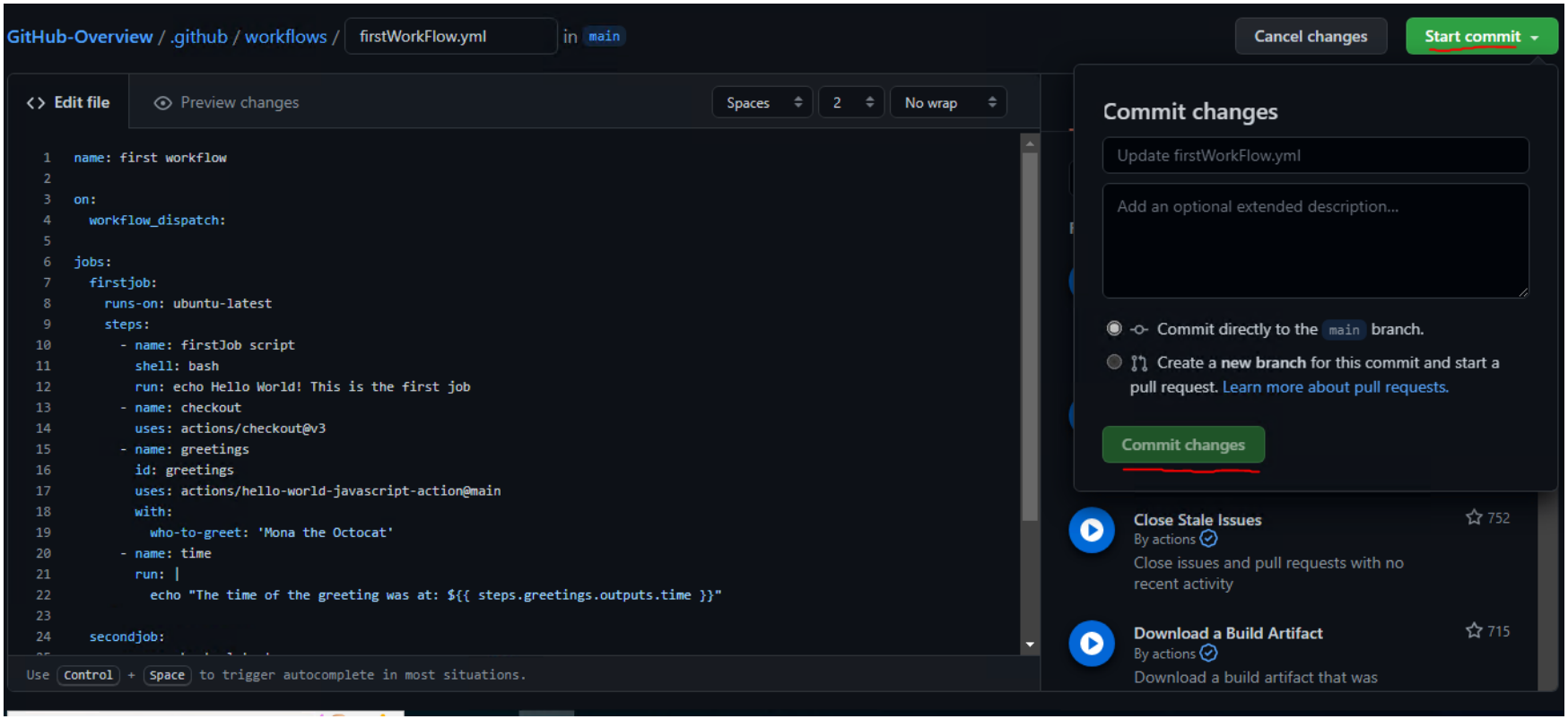
```
- name: greetings
  id: greetings
  uses: actions/hello-world-javascript-action@main
  with:
    who-to-greet: 'Mona the Octocat'
```

10) Create another step with the name “time” and the run command with the following code:

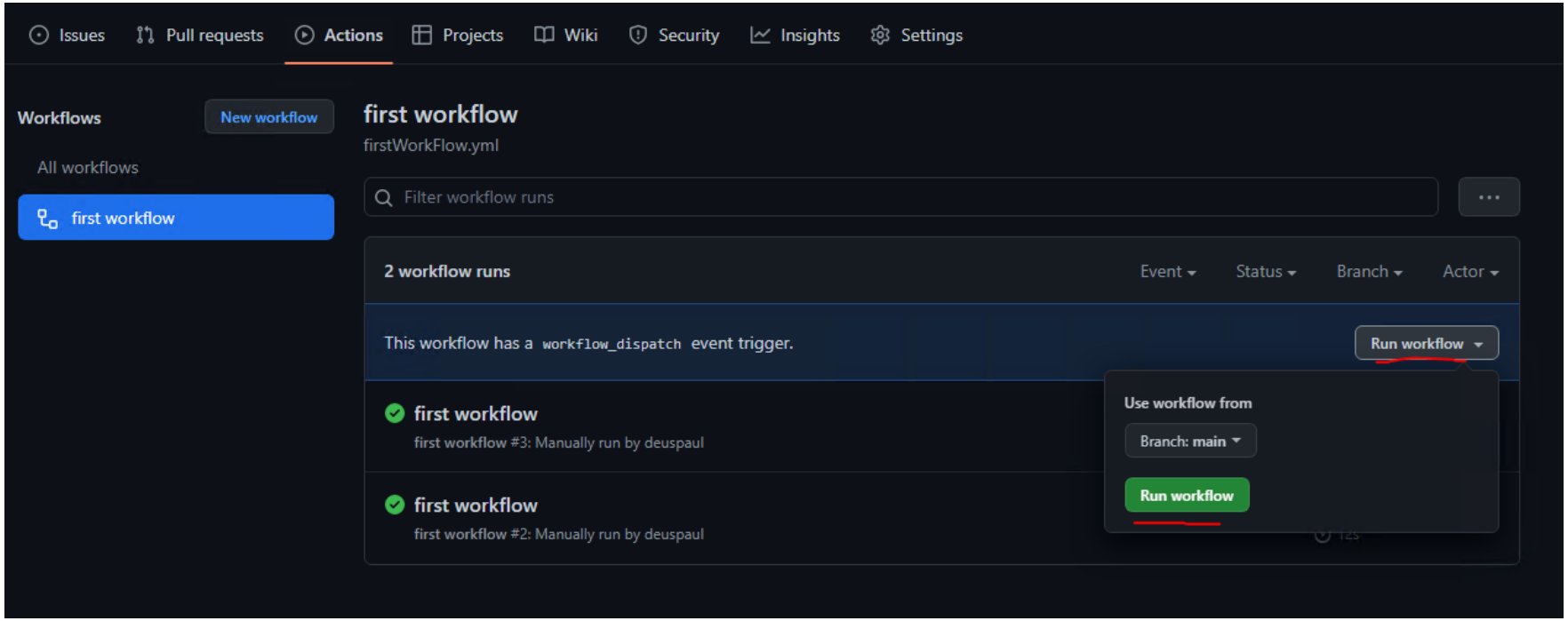
```
- name: time
  run: |
    echo "The time of the greeting was at: ${{ steps.greetings.outputs.time }}"
```

```
- name: time
  run: |
    echo "The time of the greeting was at: ${{ steps.greetings.outputs.time }}"
```

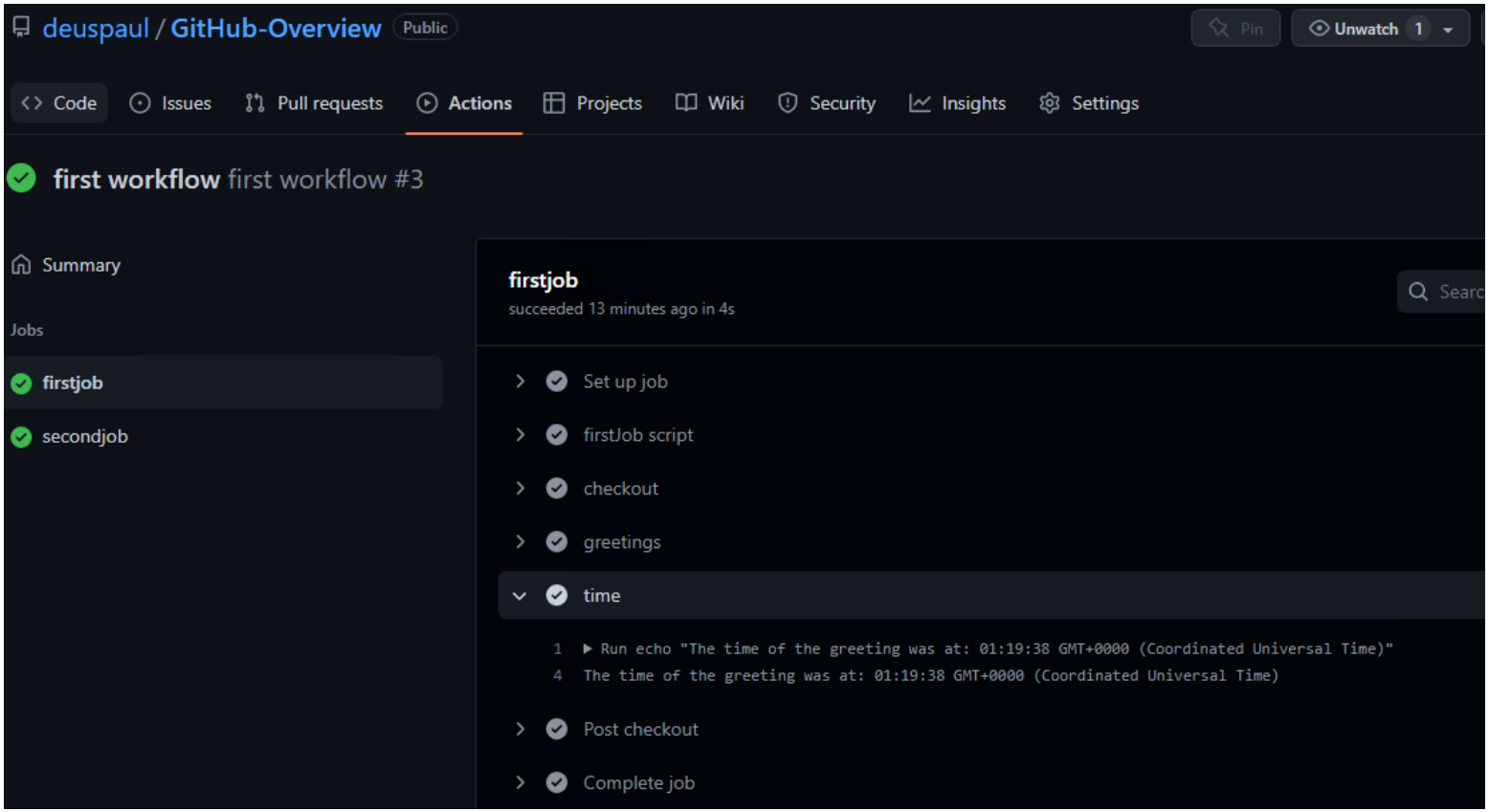
11) Go ahead and click on the green “Start commit” button, add a message and description and click on commit changes (leave “commit directly to main branch selected)



12) Click on the actions tab, select your workflow and click on the “run workflow” button (remember we have set it up to work this way with the “workflow_dispatch” trigger



13) Click on the first job, there you should see our 3 new stages, “checkout” which basically initializes a git repository in the runner, links our GitHub repository with the git remote add command and then fetches the contents, though we do not have anything in our repository right now.
Then in the “greetings” step, it basically outputs Hello followed by the value of the “who-to-greet” parameter
And in the “time” step just outputs our message followed by the time



14) Now let's go back to our editor to keep working on our workflow. The first thing we are going to do is make job2 wait for job1 to finish in order to serialize them instead of having them run in parallel. This is done with the "needs" keyword, so add the following code right after the "runs-on: ubuntu-latest" line:

    needs: firstjob

15) Next we will learn how to use variables. The highest level for variables is at the workflow level, this variable will be available to all the jobs and steps, so add the following code right after the line of code where we name our workflow:

env:
 WORKFLOW_VAR: "This variable is declared at the workflow level"

```
1  name: first workflow
2  env:
3    WORKFLOW_VAR: "This variable is declared at the workflow level"
4
5  on:
6    workflow_dispatch:
```

16) Next, lets add a variable at the job level, so in the “secondjob” add the following block of code after the line of code with the “needs” keyword

env:
 JOB_VAR: " This is a job variable"

```
secondjob:
  runs-on: ubuntu-latest
  needs: firstjob
  env:
    JOB_VAR: "This is a job variable"
  steps:
    - name: secondJob script
```

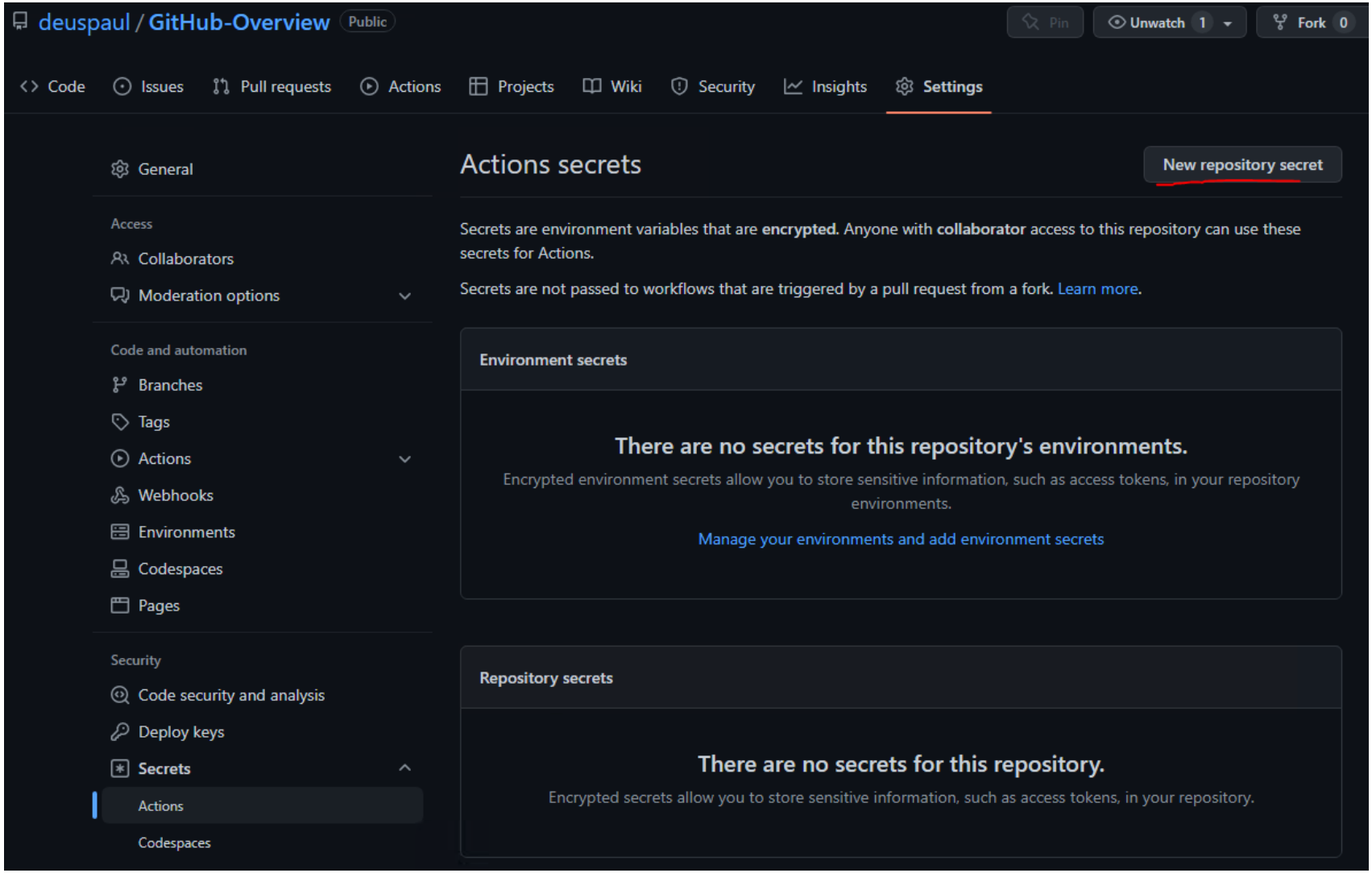
17) And last, the variable at the step level, so lets go ahead and create a new step with name “environment variables” next add “env:” and the step variable “STEP_VAR” with the following code:

- name: environment variables
 env:

STEP_VAR: "This variable is declared at the step level"

```
- name: environment variables
  env:
    STEP_VAR: "This variable is declared at the step level"
```

18) Next lets add a secret to our repository. Right click on “Settings” and open it in a new tab. Locate the “Security” section in the left menu, click on “Secrets” and then on “Actions”. From there click on “New repository secret”



19) Name it “SOME_SECRET” and in the value of the secret field, input “password”, and click on “add secret”

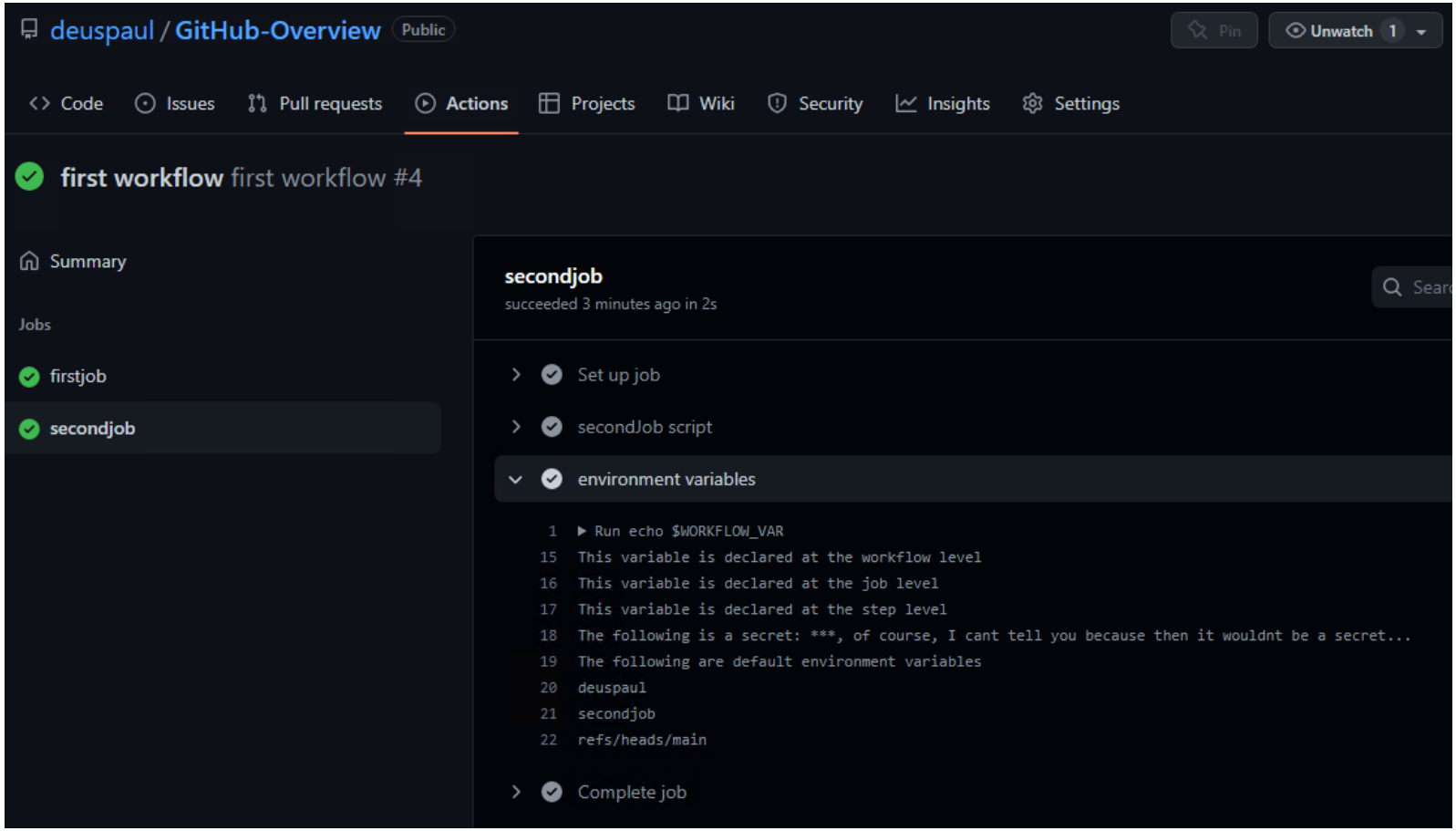
20) Go back to the tab where we are editing our workflow, and lets echo all our variables in the step we created in the previous step. Add the following code:

```
run: |
  echo $WORKFLOW_VAR
  echo $JOB_VAR
  echo $STEP_VAR
  echo "The following is a secret: ${{ secrets.SOME_SECRET }}, of course, I cant tell you because then it wouldnt be a secret..."
  echo "The following are default environment variables:"
  echo $GITHUB_ACTOR
  echo $GITHUB_JOB
  echo $GITHUB_REF
```

The code for the second job should look as follows:

```yaml
  secondjob:
    runs-on: ubuntu-latest
    needs: firstjob
    env:
      JOB_VAR: "This variable is declared at the job level"
    steps:
      - name: secondJob script
        shell: bash
        run: |
          echo "this is the second job"
          echo and this is a multi-line script
      - name: environment variables
        env:
          STEP_VAR: "This variable is declared at the step level"
        run: |
          echo $WORKFLOW_VAR
          echo $JOB_VAR
          echo $STEP_VAR
          echo "The following is a secret: ${{ secrets.SOME_SECRET }}, of course, I cant tell you because then it wouldnt be a secret..."
          echo "The following are default environment variables"
          echo $GITHUB_ACTOR
          echo $GITHUB_JOB
          echo $GITHUB_REF
```

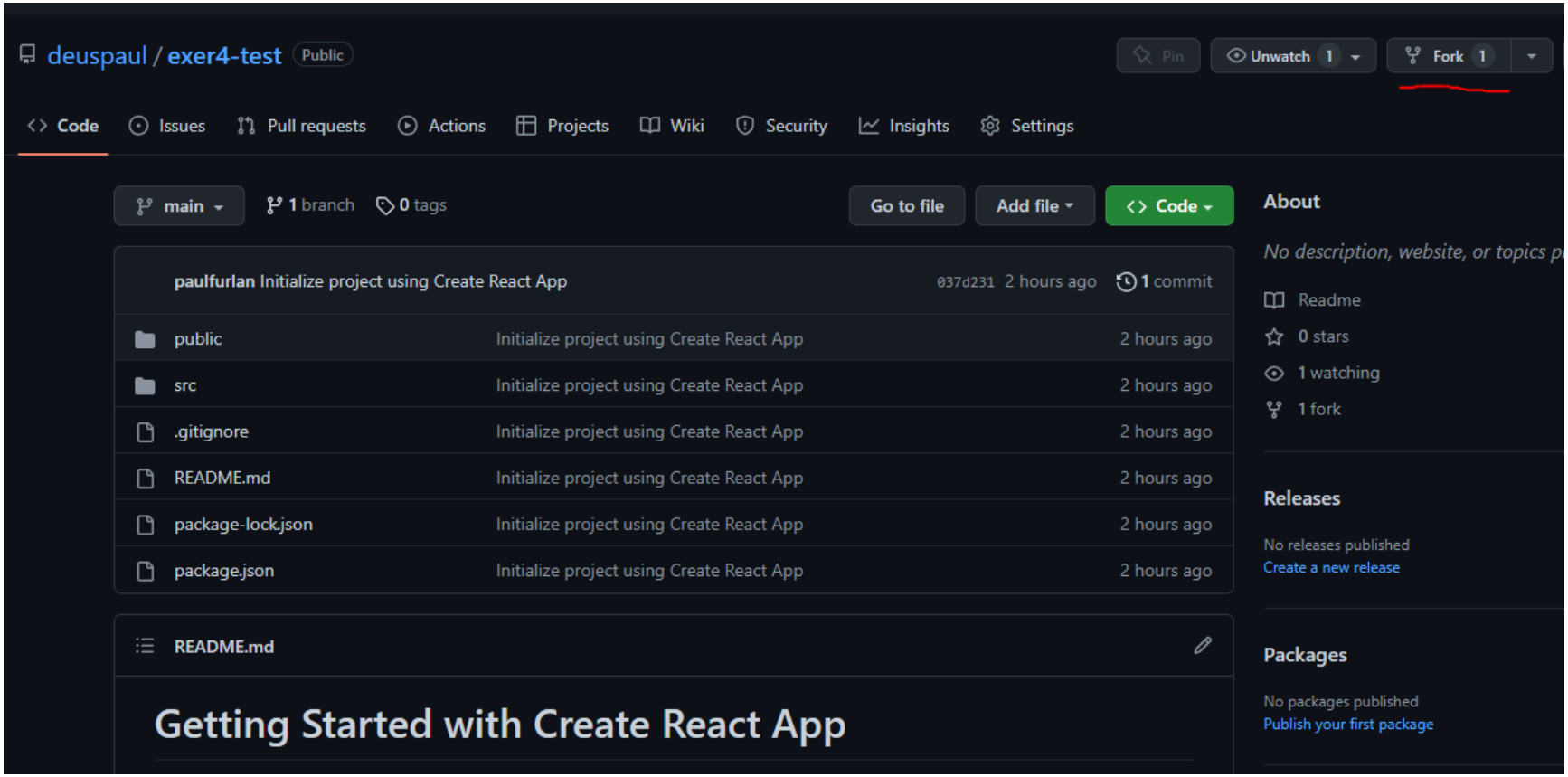
21) In the actions tab, you will notice that the second job comes after the first job now, and the output of the variables will be the following:



Exercise #4: Building & testing

1) Access the following link GitHub repository and click on the fork button: https://github.com/deuspaul/exer4-test

   This is a simple NPM based react project created with the “npx create-react-app” command



2) Click on “Actions” and select “set up a workflow yourself”. Erase all the content from the default file and add the following code:

```
name: build
on:
  workflow_dispatch:

jobs:
  build:
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v3
```

3) In this workflow we will be building and testing our project. The first thing we will do is setup our cache to speed up this job in our workflow. To do so, add the following step to the code from the previous step.

```
- name: Cache
  uses: actions/cache@v3.0.8
  with:
    path: ~/.npm
    key: buildCache
```

Edit file Preview changes

```yaml
name: build
on:
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Cache
        uses: actions/cache@v3.0.8
        with:
          path: ~/.npm
          key: buildCache
```

4) Next we instruct it to install the packages it depends on based on the package-lock file in the repository. The difference here is that npm ci is used in automated environments such as CI pipelines.

   - run: npm ci

5) Next we will use the included function to run unit tests. It already has a default single unit test defined in /src/App.test.js, which just tests that the text: “learn react” is in the page. We also run code coverage to get a report on the % of code covered by tests by adding --
   --coverage. We also have to tell it that we are running this command in a workflow/pipeline, to do so, we set an environment variable named “CI” to “true”

   - run: npm test -- --coverage
     env:
       CI: true

exer4-test-fork / .github / workflows / main.yml in main

Edit file | Preview changes

```yaml
name: build
on:
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Cache
        uses: actions/cache@v3.0.8
        with:
          path: ~/.npm
          key: buildCache
      - run: npm ci
      - run: npm test -- --coverage
        env:
          CI: true
```

6) Next, we will upload the artifacts generated by the tests so they can be reviewed outside of the pipeline. To do so, add the following block of code:

```
- name: Upload code coverage
  uses: actions/upload-artifact@v3.1.0
  with:
```

```
    name: codecoverage
    path: coverage
```

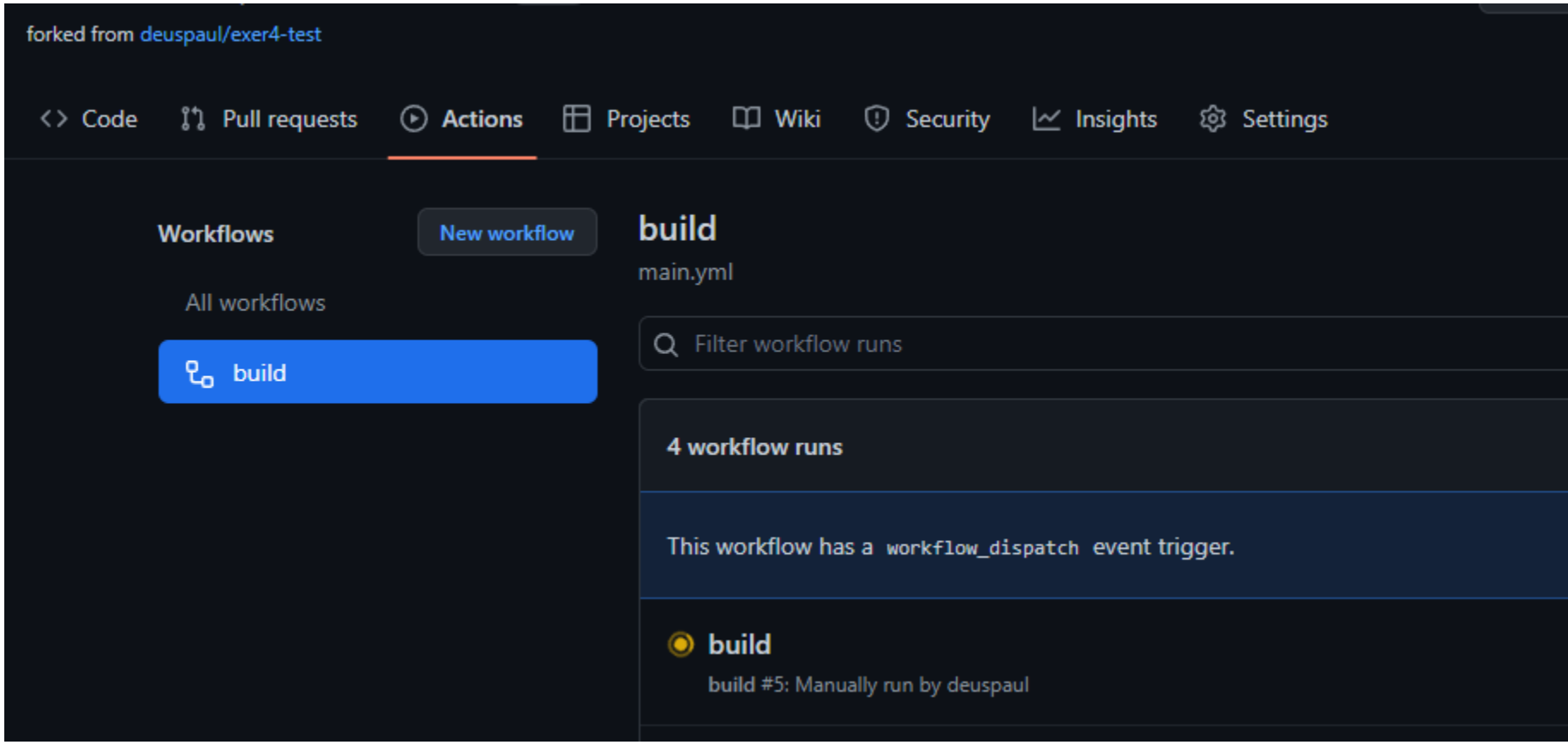
7) And finally, we will build our application with the “npm run build” command followed by an action to upload the build artifacts.

```
- name: build
  run: npm run build
- name: Upload build files
  uses: actions/upload-artifact@v3.1.0
  with:
    name: build
    path: build
```

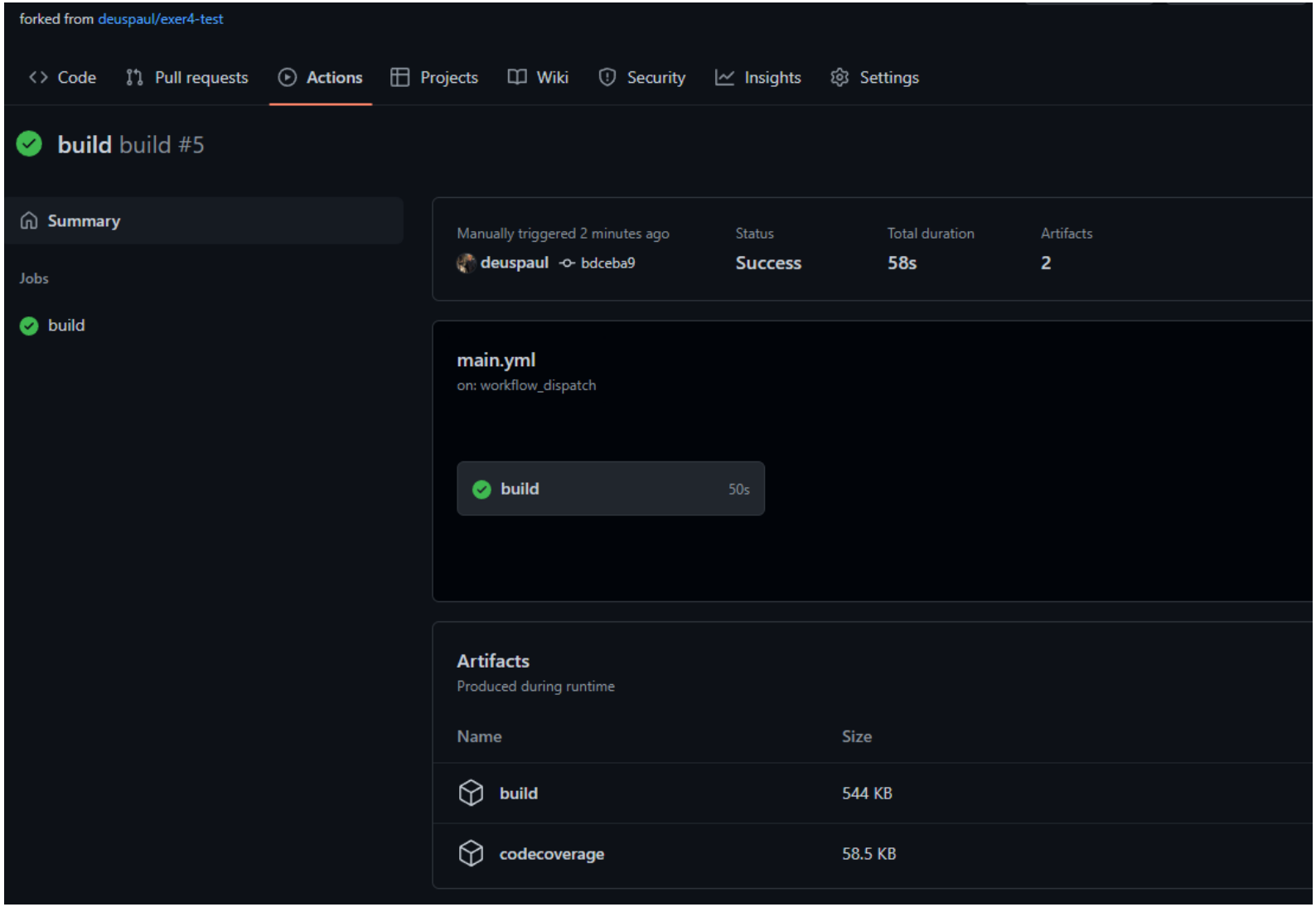
30 lines (29 sloc) | 664 Bytes

```yaml
name: build
on:
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Cache
        uses: actions/cache@v3.0.8
        with:
          path: ~/.npm
          key: buildCache
      - run: npm ci
      - run: npm test -- --coverage
        env:
          CI: true
      - name: Upload code coverage
        uses: actions/upload-artifact@v3.1.0
        with:
          name: codecoverage
          path: coverage
      - name: build
        run: npm run build
      - name: Upload build files
        uses: actions/upload-artifact@v3.1.0
        with:
          name: build
          path: build
```

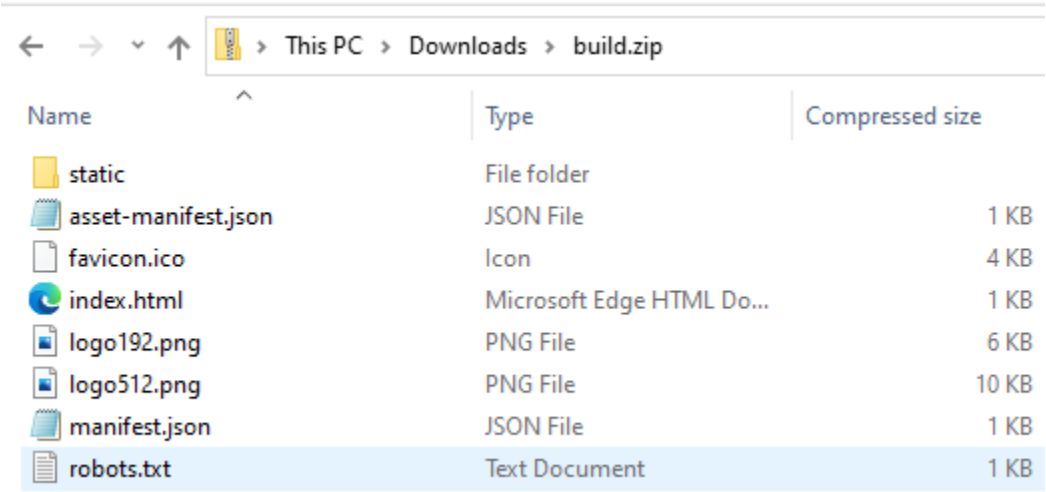
8) Head on over to actions and click on “run workflow” on main branch



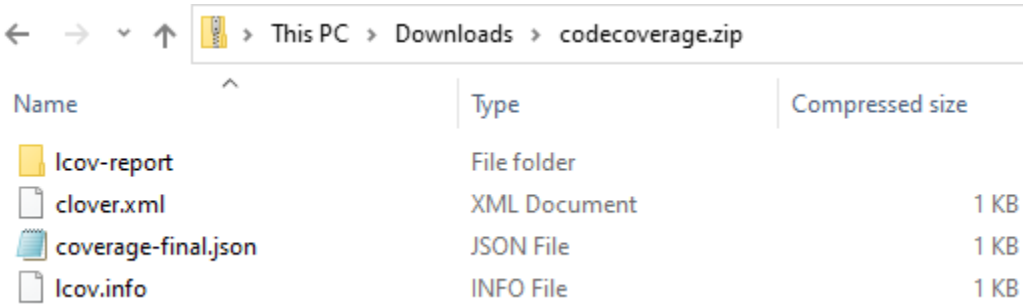
9) The workflow should look like the following and you should be able to see the artifacts at the bottom:



10) The contents of the build file should look like the following:

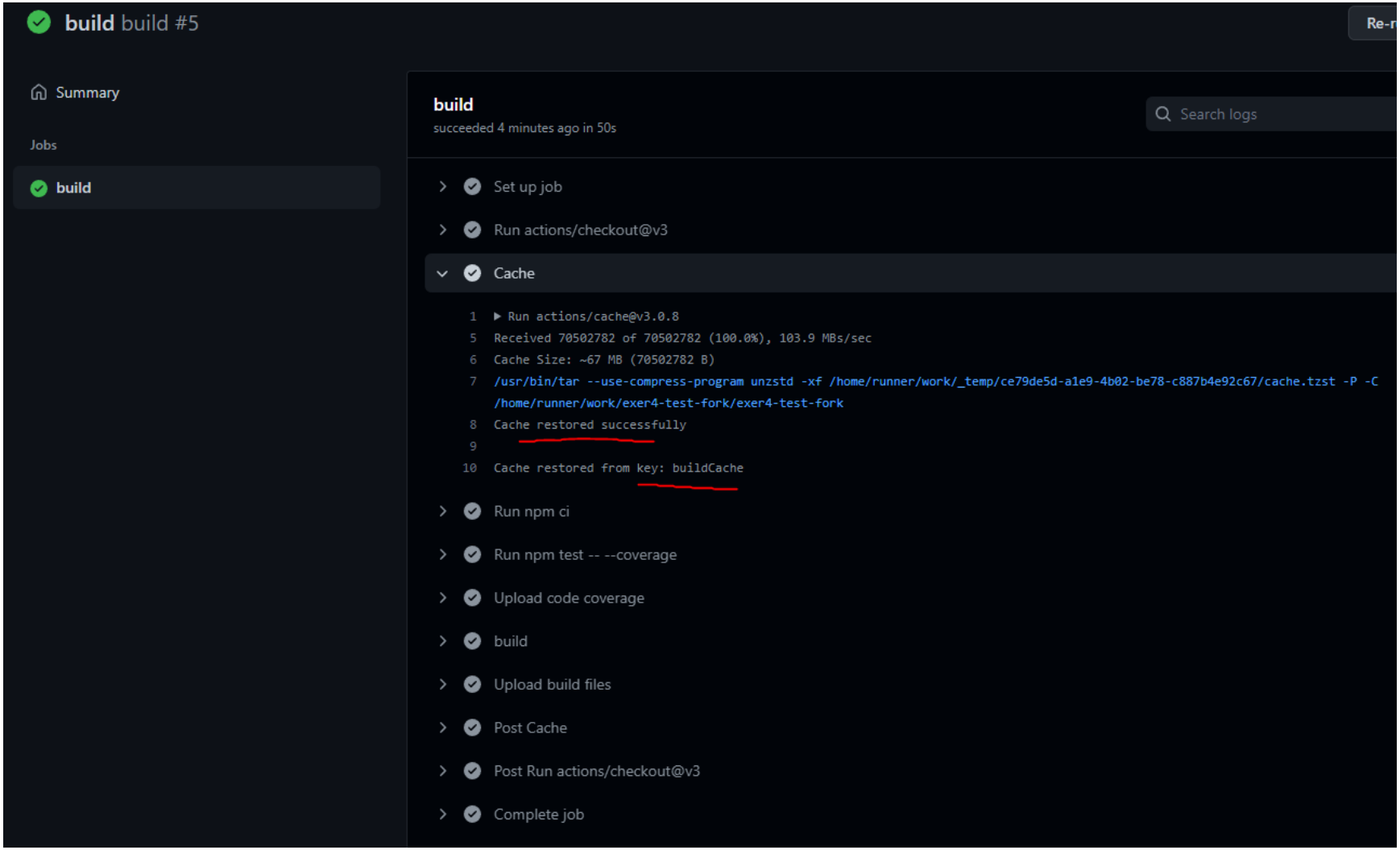


11) And the ones from code coverage should look like the following:



12) Upon clicking on the build job, you should notice the following:

If you have run this workflow more than once, the cache should say cache restored. Otherwise, if it is the first time it is run, the cache will be created:



13) The report for the unit test and code coverage can be viewed in the pipeline job as well as in the artifacts where it can be viewed as a webpage:

build build #5

Summary

Jobs

build

build

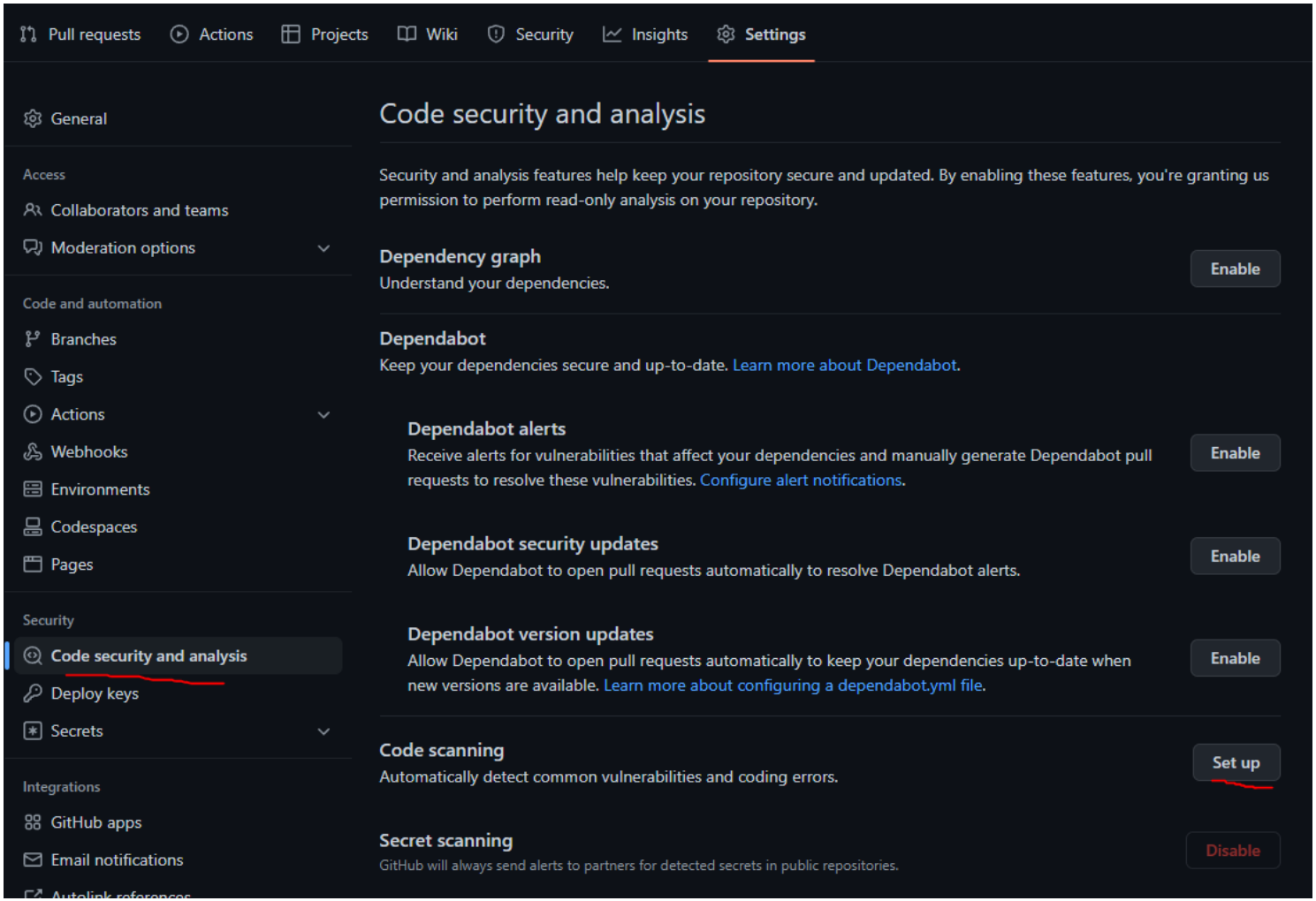
succeeded 6 minutes ago in 50s

- Set up job
- Run actions/checkout@v3
- Cache
- Run npm ci
- Run npm test -- --coverage

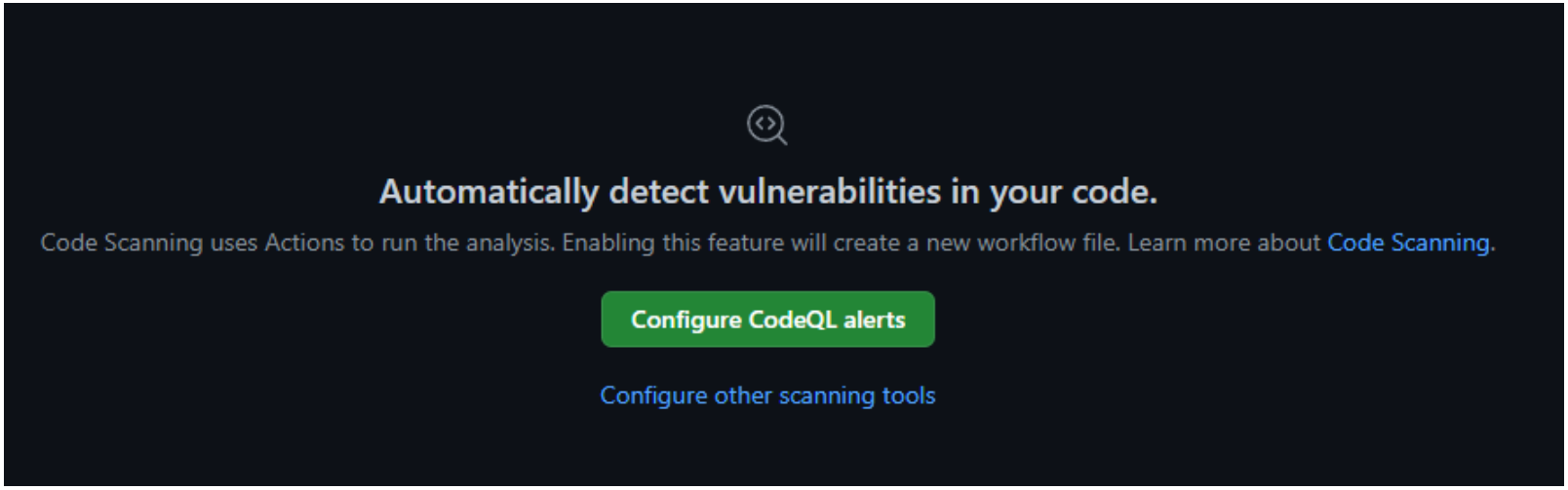
```
1  ▶ Run npm test -- --coverage
6
7  > react-build@0.1.0 test
8  > react-scripts test
9
10 PASS src/App.test.js
11   ✓ renders learn react link (36 ms)
12
13 --------------------|---------|----------|---------|---------|-------------------
14 File                | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
15 --------------------|---------|----------|---------|---------|-------------------
16 All files           |    8.33 |        0 |   33.33 |    8.33 |
17  App.js             |     100 |      100 |     100 |     100 |
18  index.js           |       0 |      100 |     100 |       0 | 7-17
19  reportWebVitals.js |       0 |        0 |       0 |       0 | 1-8
20 --------------------|---------|----------|---------|---------|-------------------
21 Test Suites: 1 passed, 1 total
22 Tests:       1 passed, 1 total
23 Snapshots:   0 total
24 Time:        1.738 s
25 Ran all test suites.
```

14) Next lets set up code scanning. Click on settings, then “code security and analysis” within the “security” section at the left.

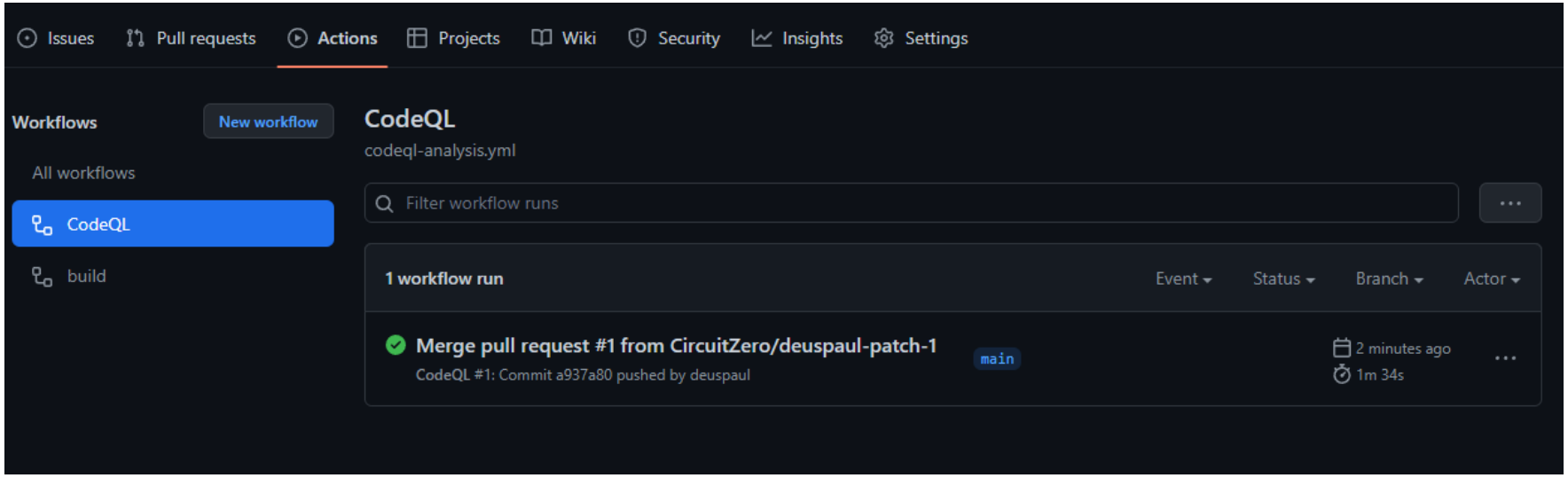
15) Click on the “set up” button for Code Scanning.



16) Then click on “Configure CodeQL alerts”



17) Leave the default settings, and click on “Start commit”. This will add a new workflow to your repository that will run code scanning



18) You can view details about the scan as it is running by expanding the “Perform CodeQL Analysis” step:

Merge pull request #1 from CircuitZero/deuspaul-patch-1 CodeQL #1

Re-run all jo

Summary

Jobs

Analyze (javascript)

Analyze (javascript)

succeeded 3 minutes ago in 1m 23s

Search logs

Set up job

Checkout repository

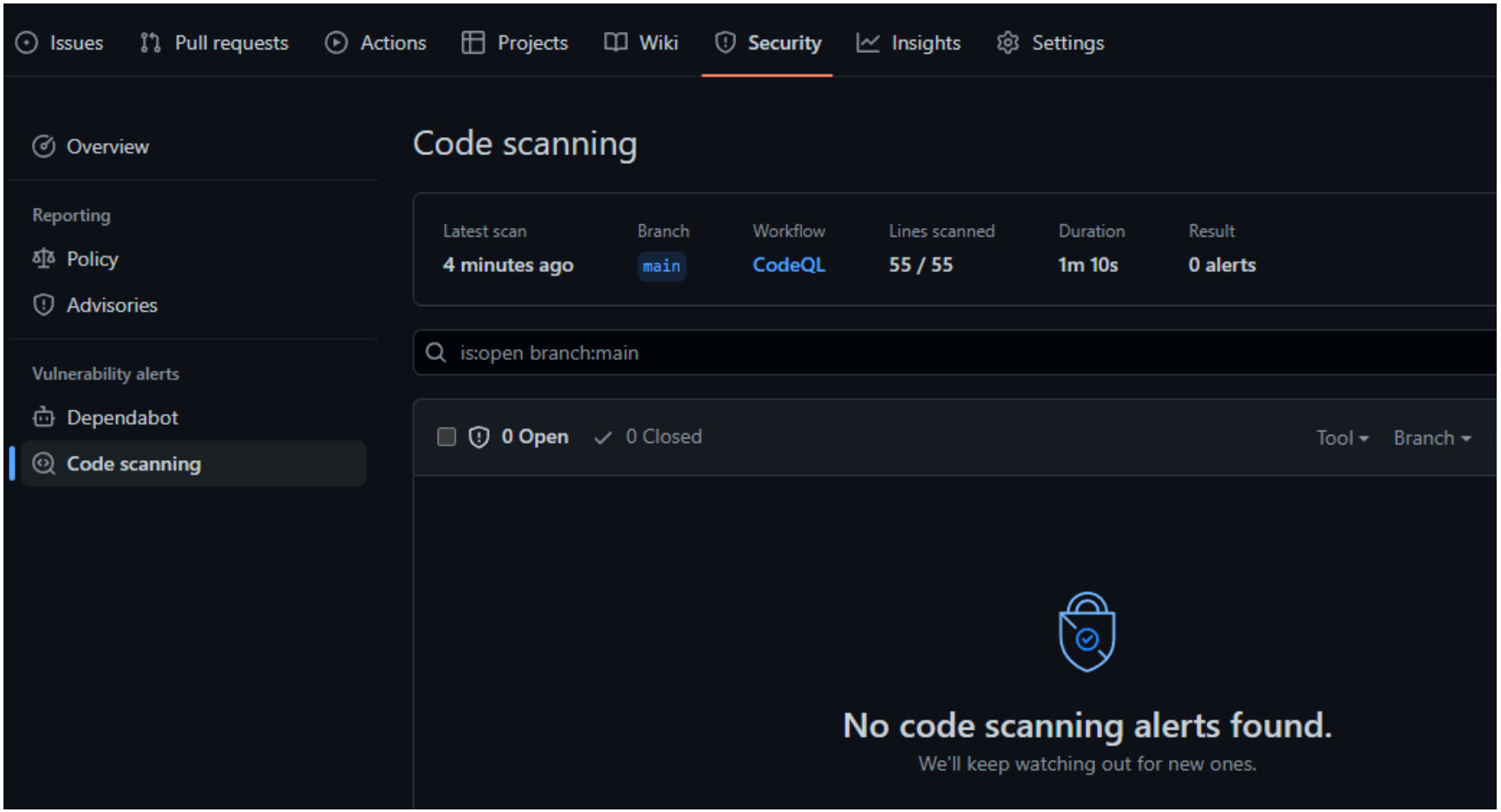
Initialize CodeQL

Autobuild

Perform CodeQL Analysis

```
1    ▶ Run github/codeql-action/analyze@v2
28   /opt/hostedtoolcache/CodeQL/0.0.0-20220825/x64/codeql/codeql version --format=terse
29   2.10.4
30   ▶ Extracting javascript
273  ▶ Finalizing javascript
277  ▶ Running queries for javascript
807  ▶ Interpreting results for javascript
1088 Analysis produced the following diagnostic data:
1089
1090 |          Diagnostic          |  Summary  |
1091 +------------------------------+-----------+
1092 | Successfully extracted files | 9 results |
1093
1094 Analysis produced the following metric data:
1095
1096 |                                  Metric                                  | Value |
1097 +--------------------------------------------------------------------------+-------+
1098 | Total lines of JavaScript and TypeScript code in the database            |    55 |
1099 | Total lines of user written JavaScript and TypeScript code in the database |    55 |
1100
1101
1102 /opt/hostedtoolcache/CodeQL/0.0.0-20220825/x64/codeql/codeql database print-baseline /home/runner/work/_temp/codeql_databases/javascri
1103 Counted a baseline of 55 lines of code for javascript.
1104 Counted a baseline of 55 lines of code for javascript.
1105
1106 ▶ Cleaning up databases
1112
1113 ▶ Uploading results
1117
1118 /opt/hostedtoolcache/CodeQL/0.0.0-20220825/x64/codeql/codeql database bundle /home/runner/work/_temp/codeql_databases/javascript --
     output=/home/runner/work/_temp/codeql_databases/javascript.zip --name=javascript
1119 Creating bundle metadata for /home/runner/work/_temp/codeql_databases/javascript...
```

19) And you can view the status in the “Security” tab and then click on “Code scanning”



Exercise #5: Deploy

1) Make sure you have NPM installed in your system, if you don't have it installed follow the documentation:
   https://nodejs.org/en/download/

2) Next open a terminal/powershell/cmd and install surge: npm install –global surge

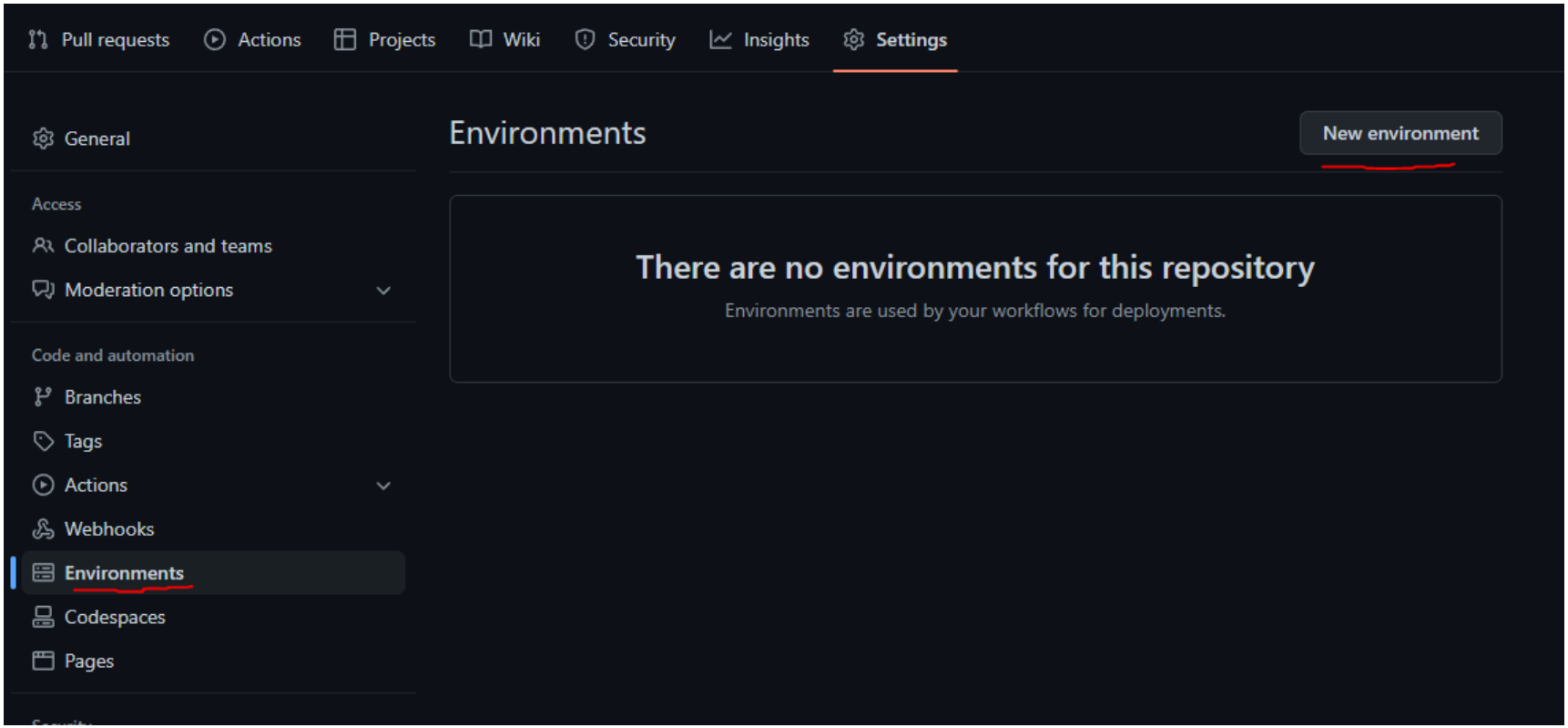
3) Next run: "surge" in terminal/powershell/cmd. If you get an error message about scripts being disabled in the system, run the following to bypass scripts for the process:
   Set-ExecutionPolicy bypass -Scope Process

4) It should ask you for your email, and to create a password.
   Then it will ask you for your project location and a domain to host your project. You can hit ctrl+c to escape from the command as we are just creating the account now.
   You should receive an email to activate your account.

```
PS C:\dev\projects\react-build\react-build> Set-ExecutionPolicy bypass -Scope Process
PS C:\dev\projects\react-build\react-build> surge

   Welcome to surge! (surge.sh)
   Login (or create surge account) by entering email & password.

          email: paulfurlan@live.com
       password:

   Running as paulfurlan@live.com (Student)

        project:
         domain: melted-chickens.surge.sh

   Aborted - Not initiated.

PS C:\dev\projects\react-build\react-build> ^C
PS C:\dev\projects\react-build\react-build>
```

5) Generate a token with the command: surge token
   copy that token

6) Open the project from the previous exercise (the one we forked) and click on “settings”. Then click on “environments” and click on “New environment”. Name it “production” and do not add any protection rules.



7) Next, click on “Add Secret” within “Environment Secrets” and create the following:
SURGE_LOGIN (Input the email address for your surge account)
SURGE_TOKEN (paste the token that we copied on step #5)

General

Access

Collaborators and teams

Moderation options

Code and automation

Branches

Tags

Actions

Webhooks

Environments

Codespaces

Pages

Security

Code security and analysis

Deploy keys

Secrets

Integrations

GitHub apps

Email notifications

Autolink references

Environments / Configure production

Environment protection rules

Can be used to configure manual approvals and timeouts.

Required reviewers

Specify people or teams that may approve workflow runs when they access this environment.

Wait timer

Set an amount of time to wait before allowing deployments to proceed.

Save protection rules

Deployment branches

Can be used to limit what branches can deploy to this environment using branch name patterns.

All branches

Environment secrets

Secrets are encrypted environment variables. They are accessible only by GitHub Actions in the context of this environment.

SURGE_LOGIN	Updated 29 seconds ago	Update	Remove
SURGE_TOKEN	Updated now	Update	Remove

Add Secret

8) Next let's edit our workflow to deploy our application. Create a new job and add the following:

```
deploy:
  runs-on: ubuntu-latest
  needs: build
  environment:
    name: production
    url: http://< WhateverURLThatIsNotTaken >.surge.sh
```

```
31    deploy:
32      runs-on: ubuntu-latest
33      needs: build
34      environment:
35        name: production
36        url: http://melted-chickens.surge.sh
```

9) Now we are ready to add the steps. Let'sdownload the build artifact from the "build" job.

```
steps:
  - name: Download the Build Artifact
    uses: actions/download-artifact@v3.0.0
    with:
      name: build
```

10) Next, lets deploy our application. Add another step with the following code:

```
- name: deploy to surge
  run: npx surge –project '.' –domain melted-chickens.surge.sh
  env:
    SURGE_LOGIN: ${{ secrets.SURGE_LOGIN }}
    SURGE_TOKEN: ${{ secrets.SURGE_TOKEN }}
```

```yaml
  deploy:
    runs-on: ubuntu-latest
    needs: build
    environment:
      name: production
      url: http://melted-chickens.surge.sh
    steps:
      - name: Download the Build Artifact
        uses: actions/download-artifact@v3.0.0
        with:
          name: build
      - name: deploy to surge
        run: npx surge --project '.' --domain melted-chickens.surge.sh
        env:
          SURGE_LOGIN: ${{ secrets.SURGE_LOGIN }}
          SURGE_TOKEN: ${{ secrets.SURGE_TOKEN }}

```

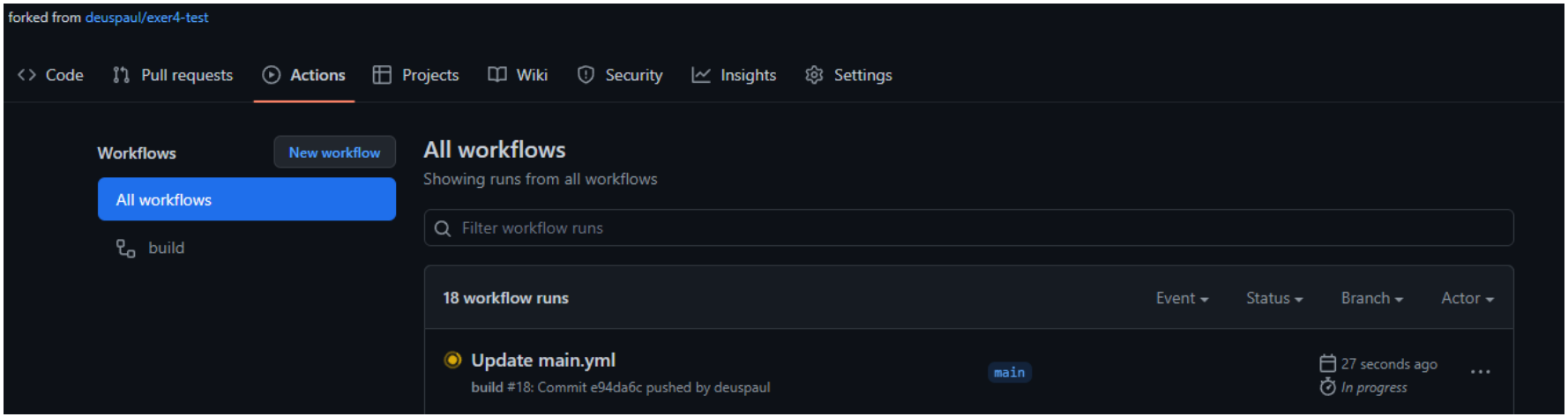
11) Change the trigger from “workflow_dispatch” to “push: branches: -main”

on:
 push:
  branches:
   - main

```yaml
name: build
on:
  push:
    branches:
      - main
```

12) Click on “start commit” and commit directly to main branch

13) This will automatically trigger our workflow from now on everytime we make changes on the main branch



14) And once the job finishes successfully, you should see your environment url in the deploy job:

Triggered via push 1 minute ago

deuspaul pushed e94da6c main

Status

Success

Total duration

1m 19s

Artifacts

2

main.yml

on: push

build 49s

deploy 12s

http://melted-chickens.surge.sh

Artifacts

Produced during runtime

Name	Size
build	544 KB
codecoverage	58.5 KB

As well as in the main page of your repository:

CircuitZero / exer4-test-fork Public

forked from deuspaul/exer4-test

Edit Pins | Watch 0 | Fork 1 | Star

Code | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main | 3 branches | 0 tags | Go to file | Add file | Code

This branch is 12 commits ahead, 2 commits behind deuspaul:main. Contribute | Sync fork

deuspaul Update main.yml e94da6c 2 minutes ago 17 commits

.github/workflows	Update main.yml	2 minutes ago
public	Initialize project using Create React App	5 hours ago
src	Initialize project using Create React App	5 hours ago
.gitignore	Initialize project using Create React App	5 hours ago
README.md	Initialize project using Create React App	5 hours ago
package-lock.json	Initialize project using Create React App	5 hours ago
package.json	Initialize project using Create React App	5 hours ago

README.md

Getting Started with Create React App

About

No description, website, or topics provided.

Readme

0 stars

0 watching

1 fork

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Environments 1

production Active