GitHub Actions Quick Start Tutorial

Using GitHub Actions requires a GitHub repository. The jobs in a workflow can automatically execute actions related to the repository and interact with the repository's code too.

Create Workflow

To create workflows in GitHub Actions:

1. Create a new directory in your repository and name it `.github/workflows` if such a directory doesn't already exist.
2. In the new directory, click the Add File button on the top-right side of the window and select Create new file. Name the newly created file `demo-github-actions-workflow.yml`
3. Paste the following code in the `demo-github-actions-workflow.yml`

```
name: Demo GitHub Actions Workflow

on: [push]

jobs:

 Discover-GitHub-Actions-Workflows:

   runs-on: ubuntu-latest

   steps:

    —run: echo "${{ github.event_name }} event automatically triggered this
job."

    —run: echo " A ${{ runner.os }} server hosted by GitHub has this job
running"

    —run: echo "Details of your repository: repo-name: ${{ github.repository }}
and branch name is${{ github.ref }} and your repository is ${{
github.repository }}."

    —name: Checking out the repository's code

       uses: actions/checkout@v3

    —run: echo "The runner has cloned your ${{ github.repository }}
repository."

    —run: echo "The workflow can now test your code with the runner."

    —name: List files in the repository
```

```
        run: |

          ls ${{ github.workspace }}

     -run: echo " This job has a ${{ job.status }} status."
```

4. Click the Propose new file to create a pull-request for committing this file to the repository.
5. Go to the bottom of the page and select the Create a new branch for this commit and start a pull request option.

When the yml file gets committed, the push event triggers the workflow described in it. The results of the workflow get shown in a different option.

View Results of Workflow

To see the results after running a GitHub Actions workflow:

1. Go to the repository's main page and click Actions on the bar under the repository name.
2. A sidebar on the left shows all the workflows on the page that opens. Click the one that displays Demo-GitHub-Actions-Workflow, the name of the workflow created for this tutorial.
3. The list in the center shows the workflows that have run. Select the one that displays Create demo-github-actions-workflow.yml to see the results of the one created in the previous section, Create Workflow.
4. Under the sidebar on the left that displays Jobs, select the one displaying Discover-GitHub-Actions-Workflows.
5. The log that opens shows each step of the workflow and what output it caused.
6. The steps for the echo statements in the workflow show the variables replaced with their respective values. Click on a step that displays List files in the repository, and it shows the files present in the repository.

Note: GitHub provides pre-built workflows that users can use as a template and customize per their needs. GitHub suggests the right starter workflow based on what libraries, languages, and frameworks are in the repository code.

GitHub Actions Best Practices

Here are some tips for using GitHub Actions.

Use Minimal Actions

An action's virtual machine has high bandwidth and is relatively fast, but complex actions take longer to set up and execute, resulting in more time spent waiting. GitHub Actions has limited virtual machine plans—the free plan has a cap of 2000 free minutes in a given month.

When setting up a workflow or action, you might not feel that a few seconds make a big difference—however, these seconds add up depending on the events triggering the actions and workflows. An important best practice when creating new actions is to try to keep them as minimal as possible.

For instance, you should use light Docker images for actions that run in containers—e.g., alpine, alpine-node. Keep the installations to a minimum to reduce the action's running time from boot to completion.

Keeping actions light is important regardless of whether you create a standalone action or a full CI/CD workflow. GitHub Actions sets up and runs each action in a clean environment for every run, so it has to download and install all the dependencies every time.

Avoid Unnecessary Dependency Downloads

Another way to reduce the running time is to avoid installing or downloading dependencies wherever possible. You can use several methods to minimize dependency downloads, although these are all based on two main strategies.

The first approach is to publish an entire node module folder. This strategy is suitable for publishing standalone actions in Node-based projects. The second approach is to leverage the caching mechanism in GitHub wherever possible. This strategy is suitable for standalone actions as well as actions running within a CI workflow.

Avoid Hardcoding Secrets

An important feature of GitHub Actions is secret management. You store encrypted secrets within the repository settings, providing them as environment variables or inputs to your actions whenever you choose. GitHub Actions automatically redacts secrets logged accidentally or intentionally. However, the GitHub documentation also recommends that you avoid logging secrets because this automatic redaction capability is not fully effective, especially for secrets containing structured data.

The key takeaway of this approach is to avoid manually hardcoding any secrets into public or private workflows. You should set secrets manually in your repository's settings and use step inputs or environment variables to access them.

Store Authorship Details in the Action Metadata

You can encourage authors of actions to take ownership of their code by specifying the action author in the metadata—this data resides in a YML file defining the action. This metadata can include many details about each action, including its required inputs, branding, entry point, outputs, and author.

There are two main use cases for including the action's author in the metadata file. First, when you have public actions, it is important to attribute each action to the correct author. In addition to giving authors credit for their work, it allows others who are using the actions to identify and refer to the author with any questions.

The second use case applies to private actions with an organization. The issue of credit is less relevant because no one outside the company will see it. However, the internal team still needs to know who is responsible for each action—the author maintains and answers questions about the action.