Essential features of GitHub Actions

In this article

GitHub Actions are designed to help you build robust and dynamic automations. This guide will show you how to craft GitHub Actions workflows that include environment variables, customized scripts, and more.

Overview

GitHub Actions allow you to customize your workflows to meet the unique needs of your application and team. In this guide, we'll discuss some of the essential customization techniques such as using variables, running scripts, and sharing data and artifacts between jobs.

Using variables in your workflows

GitHub Actions include default environment variables for each workflow run. If you need to use custom environment variables, you can set these in your YAML workflow file. This example demonstrates how to create custom variables named `POSTGRES_HOST` and `POSTGRES_PORT`. These variables are then available to the `node client.js` script.

```yaml
jobs:
  example-job:
      steps:
        - name: Connect to PostgreSQL
          run: node client.js
          env:
            POSTGRES_HOST: postgres
            POSTGRES_PORT: 5432
```

For more information, see "Variables."

Adding scripts to your workflow

You can use a GitHub Actions workflow to run scripts and shell commands, which are then executed on the assigned runner. This example demonstrates how to use the `run` keyword to execute the command `npm install -g bats` on the runner.

```yaml
jobs:
  example-job:
```

```
    runs-on: ubuntu-latest
    steps:
      - run: npm install -g bats
```

To use a workflow to run a script stored in your repository you must first check out the repository to the runner. Having done this, you can use the `run` keyword to run the script on the runner. The following example runs two scripts, each in a separate job step. The location of the scripts on the runner is specified by setting a default working directory for run commands. For more information, see "Setting default values for jobs."

```
jobs:
  example-job:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: ./scripts
    steps:
      - name: Check out the repository to the runner
        uses: actions/checkout@v4
      - name: Run a script
        run: ./my-script.sh
      - name: Run another script
        run: ./my-other-script.sh
```

Any scripts that you want a workflow job to run must be executable. You can do this either within the workflow by passing the script as an argument to the interpreter that will run the script - for example, `run: bash script.sh` - or by making the file itself executable. You can give the file the execute permission by using the command `git update-index --chmod=+x PATH/TO/YOUR/script.sh` locally, then committing and pushing the file to the repository. Alternatively, for workflows that are run on Linux and Mac runners, you can add a command to give the file the execute permission in the workflow job, prior to running the script:

```
jobs:
  example-job:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: ./scripts
    steps:
      - name: Check out the repository to the runner
        uses: actions/checkout@v4
      - name: Make the script files executable
        run: chmod +x my-script.sh my-other-script.sh
      - name: Run the scripts
        run: |
          ./my-script.sh
          ./my-other-script.sh
```

For more information about the `run` keyword, see "Workflow syntax for GitHub Actions."

Sharing data between jobs

If your job generates files that you want to share with another job in the same workflow, or if you want to save the files for later reference, you can store them in GitHub as *artifacts*. Artifacts are the files created when you build and test your code. For example, artifacts might include binary or package files, test results, screenshots, or log files. Artifacts are associated with the workflow run where they were created and can be used by another job. All actions and workflows called within a run have write access to that run's artifacts.

For example, you can create a file and then upload it as an artifact.

```yaml
jobs:
  example-job:
    name: Save output
    steps:
      - shell: bash
        run: |
          expr 1 + 1 > output.log
      - name: Upload output file
        uses: actions/upload-artifact@v3
        with:
          name: output-log-file
          path: output.log
```

To download an artifact from a separate workflow run, you can use the `actions/download-artifact` action. For example, you can download the artifact named `output-log-file`.

```yaml
jobs:
  example-job:
    steps:
      - name: Download a single artifact
        uses: actions/download-artifact@v3
        with:
          name: output-log-file
```

To download an artifact from the same workflow run, your download job should specify `needs: upload-job-name` so it doesn't start until the upload job finishes.

For more information about artifacts, see "Storing workflow data as artifacts."

Next steps

To continue learning about GitHub Actions, see "About workflows."