Understanding GitHub Actions

Learn the basics of GitHub Actions, including core concepts and essential terminology.

Get started with GitHub Actions

1 of 6 in learning path

Next: Finding and customizing actions

In this article

Overview 🔗

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.

GitHub Actions goes beyond just DevOps and lets you run workflows when other events happen in your repository. For example, you can run a workflow to automatically add the appropriate labels whenever someone creates a new issue in your repository.

GitHub provides Linux, Windows, and macOS virtual machines to run your workflows, or you can host your own self-hosted runners in your own data center or cloud infrastructure.

For more information about introducing GitHub Actions to your enterprise, see "Introducing GitHub Actions to your enterprise."

The components of GitHub Actions 🔗

You can configure a GitHub Actions *workflow* to be triggered when an *event* occurs in your repository, such as a pull request being opened or an issue being created. Your workflow contains one or more *jobs* which can run in sequential order or in parallel. Each job will run inside its own virtual machine *runner*, or inside a container, and has one or more *steps* that either run a script that you define or run an *action*, which is a reusable extension that can simplify your workflow.
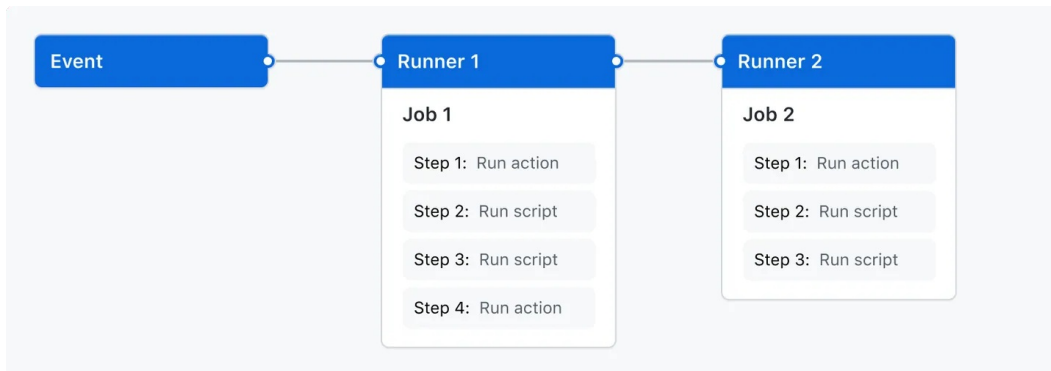
Workflows

A workflow is a configurable automated process that will run one or more jobs. Workflows are defined by a YAML file checked in to your repository and will run when triggered by an event in your repository, or they can be triggered manually, or at a defined schedule.

Workflows are defined in the `.github/workflows` directory in a repository, and a repository can have multiple workflows, each of which can perform a different set of tasks. For example, you can have one workflow to build and test pull requests, another workflow to deploy your application every time a release is created, and still another workflow that adds a label every time someone opens a new issue.

You can reference a workflow within another workflow. For more information, see "Reusing workflows."

For more information about workflows, see "Using workflows."

Events

An event is a specific activity in a repository that triggers a workflow run. For example, activity can originate from GitHub when someone creates a pull request, opens an issue, or pushes a commit to a repository. You can also trigger a workflow to run on a schedule, by posting to a REST API, or manually.

For a complete list of events that can be used to trigger workflows, see Events that trigger workflows.

Jobs

A job is a set of *steps* in a workflow that is executed on the same runner. Each step is either a shell script that will be executed, or an *action* that will be run. Steps are executed in order and are dependent on each other. Since each step is executed on the same runner, you can share data from one step to another. For example, you can have a step that builds your application followed by a step that tests the application that was built.

You can configure a job's dependencies with other jobs; by default, jobs have no dependencies and run in parallel with each other. When a job takes a dependency on another job, it will wait for the dependent job to complete before it can run. For example, you may have multiple build jobs for different architectures that have no dependencies, and a packaging job that is dependent on those jobs. The build jobs will run in parallel, and when they have all completed successfully, the packaging job will run.

For more information about jobs, see "Using jobs."

Actions

An *action* is a custom application for the GitHub Actions platform that performs a complex but frequently repeated task. Use an action to help reduce the amount of repetitive code that you write in your workflow files. An action can pull your git repository from GitHub, set up the correct toolchain for your build environment, or set up the authentication to your cloud provider.

You can write your own actions, or you can find actions to use in your workflows in the GitHub Marketplace.

To share actions across your enterprise without publishing the actions publicly, you can store the actions in an internal repository, then configure the repository to allow access to GitHub Actions workflows in other repositories owned by the same organization or by any organization in the enterprise. For more information, see "Sharing actions and workflows with your enterprise."

For more information, see "Creating actions."

Runners ∞

A runner is a server that runs your workflows when they're triggered. Each runner can run a single job at a time. GitHub provides Ubuntu Linux, Microsoft Windows, and macOS runners to run your workflows; each workflow run executes in a fresh, newly-provisioned virtual machine. GitHub also offers larger runners, which are available in larger configurations. For more information, see "About larger runners." If you need a different operating system or require a specific hardware configuration, you can host your own runners. For more information about self-hosted runners, see "Hosting your own runners."

Create an example workflow ∞

GitHub Actions uses YAML syntax to define the workflow. Each workflow is stored as a separate YAML file in your code repository, in a directory named `.github/workflows`.

You can create an example workflow in your repository that automatically triggers a series of commands whenever code is pushed. In this workflow, GitHub Actions checks out the pushed code, installs the bats testing framework, and runs a basic command to output the bats version: `bats -v`.

1. In your repository, create the `.github/workflows/` directory to store your workflow files.

2. In the `.github/workflows/` directory, create a new file called `learn-github-actions.yml` and add the following code.

```YAML
name: learn-github-actions
run-name: ${{ github.actor }} is learning GitHub Actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v3
        with:
          node-version: '14'
      - run: npm install -g bats
      - run: bats -v
```

3. Commit these changes and push them to your GitHub repository.

Your new GitHub Actions workflow file is now installed in your repository and will run automatically each time someone pushes a change to the repository. To see the details about a workflow's execution history, see "Viewing the activity for a workflow run."

Understanding the workflow file 🔗

To help you understand how YAML syntax is used to create a workflow file, this section explains each line of the introduction's example:

```
YAML                                          Beside   Inline   ⧉

name: learn-github-actions
```

Optional - The name of the workflow as it will appear in the "Actions" tab of the GitHub repository. If this field is omitted, the name of the workflow file will be used instead.

```
run-name: ${{ github.actor }} is learning GitHub Actions
```

Optional - The name for workflow runs generated from the workflow, which will appear in the list of workflow runs on your repository's "Actions" tab. This example uses an expression with the `github` context to display the username of the actor that triggered the workflow run. For more information, see "Workflow syntax for GitHub Actions."

```
on: [push]
```

Specifies the trigger for this workflow. This example uses the `push` event, so a workflow run is triggered every time someone pushes a change to the repository or merges a pull request. This is triggered by a push to every branch; for examples of syntax that runs only on pushes to specific branches, paths, or tags, see "Workflow syntax for GitHub Actions."

```
jobs:
```

Groups together all the jobs that run in the `learn-github-actions` workflow.

```
check-bats-version:
```

Defines a job named `check-bats-version`. The child keys will define properties of the job.

```
runs-on: ubuntu-latest
```

Configures the job to run on the latest version of an Ubuntu Linux runner. This means that the job will execute on a fresh virtual machine hosted by GitHub. For syntax examples using other runners, see "Workflow syntax for GitHub Actions"

```
    steps:
```

Groups together all the steps that run in the `check-bats-version` job. Each item nested under this section is a separate action or shell script.

```
    - uses: actions/checkout@v4
```

The `uses` keyword specifies that this step will run `v4` of the `actions/checkout` action. This is an action that checks out your repository onto the runner, allowing you to run scripts or other actions against your code (such as build and test tools). You should use the checkout action any time your workflow will use the repository's code.

```
    - uses: actions/setup-node@v3
      with:
        node-version: '14'
```

This step uses the `actions/setup-node@v3` action to install the specified version of the Node.js. (This example uses version 14.) This puts both the `node` and `npm` commands in your `PATH`.

```
    - run: npm install -g bats
```

The `run` keyword tells the job to execute a command on the runner. In this case, you are using `npm` to install the `bats` software testing package.

```
    - run: bats -v
```

Finally, you'll run the `bats` command with a parameter that outputs the software version.

Visualizing the workflow file 🔗

In this diagram, you can see the workflow file you just created and how the GitHub Actions components are organized in a hierarchy. Each step executes a single action or shell script. Steps 1 and 2 run actions, while steps 3 and 4 run shell scripts. To find more prebuilt actions for your workflows, see "[Finding and customizing actions](#)."
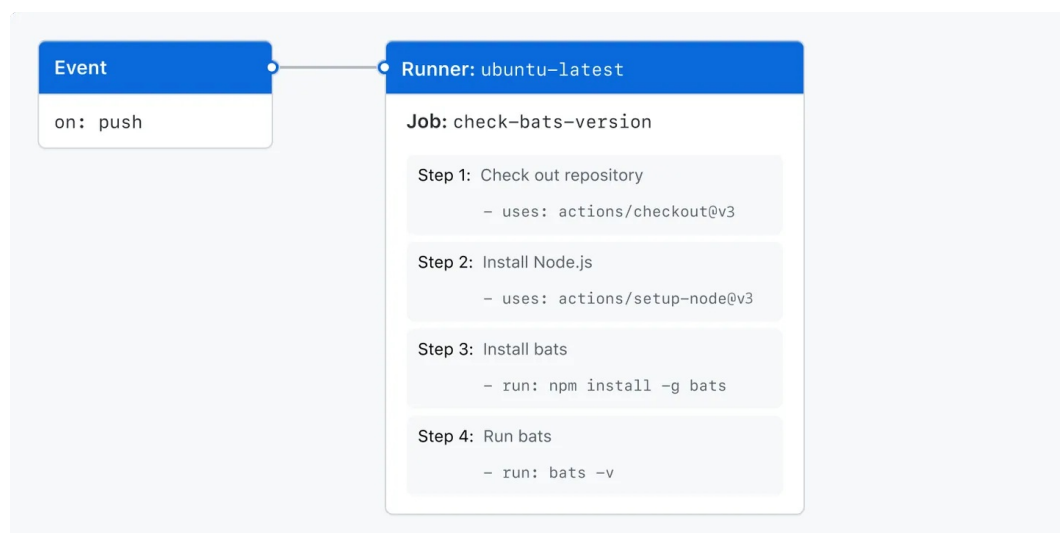
Viewing the activity for a workflow run 🔗

When your workflow is triggered, a *workflow run* is created that executes the workflow. After a workflow run has started, you can see a visualization graph of the run's progress and view each step's activity on GitHub.

1. On GitHub.com, navigate to the main page of the repository.

2. Under your repository name, click ⓘ **Actions**.
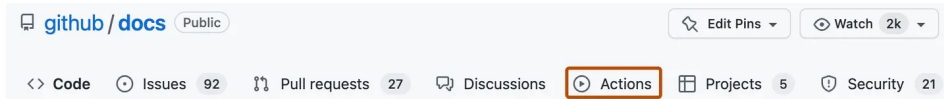
3. In the left sidebar, click the workflow you want to see.
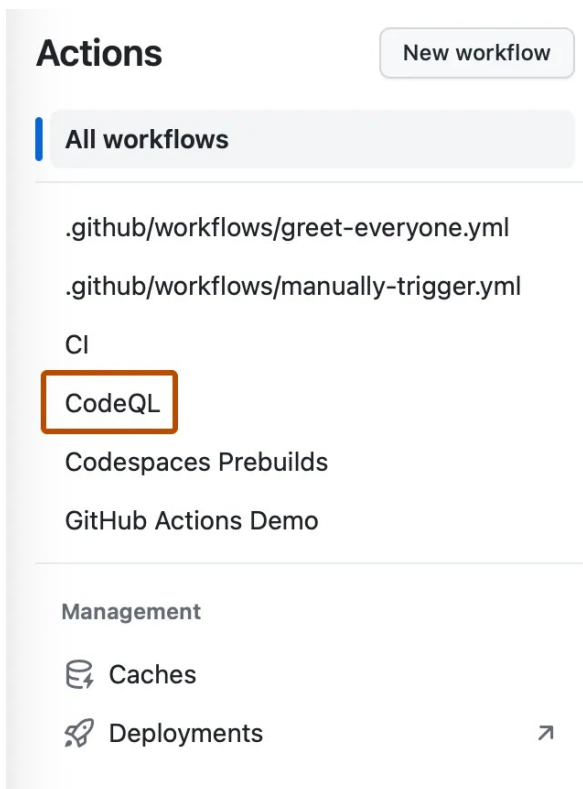
4. From the list of workflow runs, click the name of the run to see the workflow run summary.

5. In the left sidebar or in the visualization graph, click the job you want to see.

6. To view the results of a step, click the step.

Next steps 🔗

GitHub Actions can help you automate nearly every aspect of your application development processes. Ready to get started? Here are some helpful resources for taking your next steps with GitHub Actions:

- For a quick way to create a GitHub Actions workflow, see "[Using starter workflows](#)."
- For continuous integration (CI) workflows to build and test your code, see "[Automating builds and tests](#)."

- For building and publishing packages, see "[Publishing packages](#)."
- For deploying projects, see "[Deployment](#)."
- For automating tasks and processes on GitHub, see "[Managing issues and pull requests](#)."
- For examples that demonstrate more complex features of GitHub Actions, including many of the above use cases, see "[Examples](#)." You can see detailed examples that explain how to test your code on a runner, access the GitHub CLI, and use advanced features such as concurrency and test matrices.

Further reading 🔗

- "[About GitHub Actions for enterprises](#)"

- For building and publishing packages, see "[Publishing packages](#)."
- For deploying projects, see "[Deployment](#)."
- For automating tasks and processes on GitHub, see "[Managing issues and pull requests](#)."
- For examples that demonstrate more complex features of GitHub Actions, including many of the above use cases, see "[Examples](#)." You can see detailed examples that explain how to test your code on a runner, access the GitHub CLI, and use advanced features such as concurrency and test matrices.