Automatic token authentication

In this article

GitHub provides a token that you can use to authenticate on behalf of GitHub Actions.

Note: GitHub-hosted runners are not currently supported on GitHub Enterprise Server. You can see more information about planned future support on the GitHub public roadmap.

About the `GITHUB_TOKEN` secret

At the start of each workflow run, GitHub automatically creates a unique `GITHUB_TOKEN` secret to use in your workflow. You can use the `GITHUB_TOKEN` to authenticate in a workflow run.

When you enable GitHub Actions, GitHub installs a GitHub App on your repository. The `GITHUB_TOKEN` secret is a GitHub App installation access token. You can use the installation access token to authenticate on behalf of the GitHub App installed on your repository. The token's permissions are limited to the repository that contains your workflow. For more information, see "Permissions for the `GITHUB_TOKEN`."

Before each job begins, GitHub fetches an installation access token for the job. The `GITHUB_TOKEN` expires when a job finishes or after a maximum of 24 hours.

The token is also available in the `github.token` context. For more information, see "Contexts."

Using the `GITHUB_TOKEN` in a workflow

You can use the `GITHUB_TOKEN` by using the standard syntax for referencing secrets: `${{ secrets.GITHUB_TOKEN }}`. Examples of using the `GITHUB_TOKEN` include passing the token as an input to an action, or using it to make an authenticated GitHub Enterprise Server API request.

Important: An action can access the `GITHUB_TOKEN` through the `github.token` context even if the workflow does not explicitly pass the `GITHUB_TOKEN` to the action. As a good security practice, you should always make sure that actions only have the minimum access they require by limiting the permissions granted to the `GITHUB_TOKEN`. For more information, see "Permissions for the `GITHUB_TOKEN`."

When you use the repository's `GITHUB_TOKEN` to perform tasks, events triggered by the `GITHUB_TOKEN` will not create a new workflow run. This prevents you from accidentally creating recursive workflow runs. For example, if a workflow run pushes code using the repository's `GITHUB_TOKEN`, a new workflow will not run even when the repository contains a workflow configured to run when `push` events occur.

Commits pushed by a GitHub Actions workflow that uses the `GITHUB_TOKEN` do not trigger a GitHub Pages build.

Example 1: passing the `GITHUB_TOKEN` as an input

This example workflow uses the labeler action, which requires the `GITHUB_TOKEN` as the value for the `repo-token` input parameter:

YAML

```yaml
name: Pull request labeler
on: [ pull_request_target ]

jobs:
  triage:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      pull-requests: write
    steps:
      - uses: actions/labeler@v3
        with:
          repo-token: ${{ secrets.GITHUB_TOKEN }}
```

Example 2: calling the REST API

You can use the `GITHUB_TOKEN` to make authenticated API calls. This example workflow creates an issue using the GitHub REST API:

```
name: Create issue on commit

on: [ push ]

jobs:
  create_issue:
    runs-on: ubuntu-latest
    permissions:
      issues: write
    steps:
      - name: Create issue using REST API
        run: |
          curl --request POST \
          --url http(s)://HOSTNAME/api/v3/repos/${{ github.repository }}/issues \
          --header 'authorization: Bearer ${{ secrets.GITHUB_TOKEN }}' \
          --header 'content-type: application/json' \
          --data '{
            "title": "Automated issue for commit: ${{ github.sha }}",
            "body": "This issue was automatically created by the GitHub Action work
            }' \
          --fail
```

Permissions for the `GITHUB_TOKEN`

For information about the API endpoints GitHub Apps can access with each permission, see "Permissions required for GitHub Apps."

The following table shows the permissions granted to the `GITHUB_TOKEN` by default. People with admin permissions to an organization or repository can set the default permissions to be either permissive or restricted. For information on how to set the default permissions for the `GITHUB_TOKEN` for your enterprise, organization, or repository, see "Enforcing policies for GitHub Actions in your enterprise," "Disabling or limiting GitHub Actions for your organization," or "Managing GitHub Actions settings for a repository."

Scope	Default access (permissive)	Default access (restricted)	Maximum access for pull requests from public forked repositories
actions	read/write	none	read
checks	read/write	none	read
contents	read/write	read	read
deployments	read/write	none	read
issues	read/write	none	read
metadata	read	read	read
packages	read/write	none	read
pages	read/write	none	read
pull-requests	read/write	none	read
repository-projects	read/write	none	read
security-events	read/write	none	read
statuses	read/write	none	read

Note: Private repositories can control if pull requests from forks can run workflows, and configure the permissions assigned to `GITHUB_TOKEN`. For more information, see "Managing GitHub Actions settings for a repository."

Modifying the permissions for the `GITHUB_TOKEN`

You can modify the permissions for the `GITHUB_TOKEN` in individual workflow files. If the default permissions for the `GITHUB_TOKEN` are restrictive, you may have to elevate the permissions to allow some actions and commands to run successfully. If the default permissions are permissive, you can edit the workflow file to remove some permissions from the `GITHUB_TOKEN`. As a good security practice, you should grant the `GITHUB_TOKEN` the least required access.

You can see the permissions that `GITHUB_TOKEN` had for a specific job in the "Set up job" section of the workflow run log. For more information, see "Using workflow run logs."

You can use the `permissions` key in your workflow file to modify permissions for the `GITHUB_TOKEN` for an entire workflow or for individual jobs. This allows you to configure the minimum required permissions for a workflow or job. When the `permissions` key is used, all unspecified permissions are set to no access, with the exception of the `metadata` scope, which always gets read access.

You can use the `permissions` key to add and remove read permissions for forked repositories, but typically you can't grant write access. The exception to this behavior is where an admin user has selected the **Send write tokens to workflows from pull requests** option in the GitHub Actions settings. For more information, see "Managing GitHub Actions settings for a repository."

The two workflow examples earlier in this article show the `permissions` key being used at the workflow level, and at the job level. In Example 1 the two permissions are specified for the entire workflow. In Example 2 write access is granted for one scope for a single job.

For full details of the `permissions` key, see "Workflow syntax for GitHub Actions."

How the permissions are calculated for a workflow job

The permissions for the `GITHUB_TOKEN` are initially set to the default setting for the enterprise, organization, or repository. If the default is set to the restricted permissions at any of these levels then this will apply to the relevant repositories. For example, if you choose the restricted default at the organization level then all repositories in that organization will use the restricted permissions as the default. The permissions are then adjusted based on any configuration within the workflow file, first at the workflow level and then at the job level. Finally, if the workflow was triggered by a pull request from a forked repository, and the **Send write tokens to workflows from pull requests** setting is not selected, the permissions are adjusted to change any write permissions to read only.

Granting additional permissions

If you need a token that requires permissions that aren't available in the `GITHUB_TOKEN`, you can create a GitHub App and generate an installation access token within your workflow. For more information, see "Making authenticated API requests with a GitHub App in a GitHub Actions workflow." Alternatively, you can create a personal access token, store it as a secret in your repository, and use the token in your workflow with the `${{ secrets.SECRET_NAME }}` syntax. For more information, see "Managing your personal access tokens" and "Encrypted secrets."

Further reading

- "Resources in the REST API"