Triggering a workflow

In this article

How to automatically trigger GitHub Actions workflows

About workflow triggers

Workflow triggers are events that cause a workflow to run. These events can be:

- Events that occur in your workflow's repository
- Events that occur outside of GitHub Enterprise Cloud and trigger a `repository_dispatch` event on GitHub Enterprise Cloud
- Scheduled times
- Manual

For example, you can configure your workflow to run when a push is made to the default branch of your repository, when a release is created, or when an issue is opened.

Workflow triggers are defined with the `on` key. For more information, see "Workflow syntax for GitHub Actions."

The following steps occur to trigger a workflow run:

1. An event occurs on your repository. The event has an associated commit SHA and Git ref.
2. GitHub Enterprise Cloud searches the `.github/workflows` directory in your repository for workflow files that are present in the associated commit SHA or Git ref of the event.
3. A workflow run is triggered for any workflows that have `on:` values that match the triggering event. Some events also require the workflow file to be present on the default branch of the repository in order to run.

   Each workflow run will use the version of the workflow that is present in the associated commit SHA or Git ref of the event. When a workflow runs, GitHub Enterprise Cloud sets the `GITHUB_SHA` (commit SHA) and `GITHUB_REF` (Git ref) environment variables in the runner environment. For more information, see "Variables."

Triggering a workflow from a workflow

When you use the repository's `GITHUB_TOKEN` to perform tasks, events triggered by the `GITHUB_TOKEN`, with the exception of `workflow_dispatch` and `repository_dispatch`, 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. For more information, see "Automatic token authentication."

If you do want to trigger a workflow from within a workflow run, you can use a GitHub App installation access token or a personal access token instead of `GITHUB_TOKEN` to trigger events that require a token.

If you use a GitHub App, you'll need to create a GitHub App and store the app ID and private key as secrets. For more information, see "Making authenticated API requests with a GitHub App in a GitHub Actions workflow." If you use a personal access token, you'll need to create a personal access token and store it as a secret. For more information about creating a personal access token, see "Managing your personal access tokens." For more information about storing secrets, see "Using secrets in GitHub Actions."

To minimize your GitHub Actions usage costs, ensure that you don't create recursive or unintended workflow runs.

For example, the following workflow uses a personal access token (stored as a secret called `MY_TOKEN`) to add a label to an issue via GitHub CLI. Any workflows that run when a label is added will run once this step is performed.

```yaml
on:
  issues:
    types:
      - opened

jobs:
  label_issue:
    runs-on: ubuntu-latest
    steps:
      - env:
          GITHUB_TOKEN: ${{ secrets.MY_TOKEN }}
          ISSUE_URL: ${{ github.event.issue.html_url }}
        run: |
          gh issue edit $ISSUE_URL --add-label "triage"
```

Conversely, the following workflow uses `GITHUB_TOKEN` to add a label to an issue. It will not trigger any workflows that run when a label is added.

```yaml
on:
  issues:
    types:
      - opened

jobs:
  label_issue:
    runs-on: ubuntu-latest
    steps:
      - env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          ISSUE_URL: ${{ github.event.issue.html_url }}
        run: |
          gh issue edit $ISSUE_URL --add-label "triage"
```

Using events to trigger workflows

Use the `on` key to specify what events trigger your workflow. For more information about events you can use, see "Events that trigger workflows."

Using a single event

For example, a workflow with the following `on` value will run when a push is made to any branch in the workflow's repository:

```yaml
on: push
```

Using multiple events

You can specify a single event or multiple events. For example, a workflow with the following `on` value will run when a push is made to any branch in the repository or when someone forks the repository:

```yaml
on: [push, fork]
```

If you specify multiple events, only one of those events needs to occur to trigger your workflow. If multiple triggering events for your workflow occur at the same time, multiple workflow runs will be triggered.

Using activity types and filters with multiple events

You can use activity types and filters to further control when your workflow will run. For more information, see Using event activity types and Using filters. If you specify activity types or filters for an event and your workflow triggers on multiple events, you must configure each event separately. You must append a colon (`:`) to all events, including events without configuration.

For example, a workflow with the following `on` value will run when:

- A label is created
- A push is made to the `main` branch in the repository
- A push is made to a GitHub Pages-enabled branch

```yaml
on:
  label:
    types:
      - created
  push:
    branches:
      - main
  page_build:
```

Using event activity types

Some events have activity types that give you more control over when your workflow should run. Use `on.<event_name>.types` to define the type of event activity that will trigger a workflow run.

For example, the `issue_comment` event has the `created`, `edited`, and `deleted` activity types. If your workflow triggers on the `label` event, it will run whenever a label is

created, edited, or deleted. If you specify the `created` activity type for the `label` event, your workflow will run when a label is created but not when a label is edited or deleted.

```yaml
on:
  label:
    types:
      - created
```

If you specify multiple activity types, only one of those event activity types needs to occur to trigger your workflow. If multiple triggering event activity types for your workflow occur at the same time, multiple workflow runs will be triggered. For example, the following workflow triggers when an issue is opened or labeled. If an issue with two labels is opened, three workflow runs will start: one for the issue opened event and two for the two issue labeled events.

```yaml
on:
  issues:
    types:
      - opened
      - labeled
```

For more information about each event and their activity types, see "Events that trigger workflows."

Using filters

Some events have filters that give you more control over when your workflow should run.

For example, the `push` event has a `branches` filter that causes your workflow to run only when a push to a branch that matches the `branches` filter occurs, instead of when any push occurs.

```yaml
on:
  push:
    branches:
      - main
      - 'releases/**'
```

Using filters to target specific branches for pull request events

When using the `pull_request` and `pull_request_target` events, you can configure a workflow to run only for pull requests that target specific branches.

Use the `branches` filter when you want to include branch name patterns or when you want to both include and exclude branch names patterns. Use the `branches-ignore` filter when you only want to exclude branch name patterns. You cannot use both the `branches` and `branches-ignore` filters for the same event in a workflow.

If you define both `branches`/`branches-ignore` and `paths`/`paths-ignore`, the workflow will only run when both filters are satisfied.

The `branches` and `branches-ignore` keywords accept glob patterns that use characters like `*`, `**`, `+`, `?`, `!` and others to match more than one branch name. If a name contains any of these characters and you want a literal match, you need to escape each of these special characters with `\`. For more information about glob patterns, see the "Workflow syntax for GitHub Actions."

Example: Including branches

The patterns defined in `branches` are evaluated against the Git ref's name. For example, the following workflow would run whenever there is a `pull_request` event for a pull request targeting:

- A branch named `main` (`refs/heads/main`)
- A branch named `mona/octocat` (`refs/heads/mona/octocat`)
- A branch whose name starts with `releases/`, like `releases/10` (`refs/heads/releases/10`)

```yaml
on:
  pull_request:
    # Sequence of patterns matched against refs/heads
    branches:
      - main
      - 'mona/octocat'
      - 'releases/**'
```

Note: You should not use branch filtering to skip workflow runs if the workflow has been configured to be required. For more information, see "Required workflows."

If a workflow is skipped due to branch filtering, path filtering, or a commit message, then checks associated with that workflow will remain in a "Pending" state. A pull request that requires those checks to be successful will be blocked from merging.

Example: Excluding branches

When a pattern matches the `branches-ignore` pattern, the workflow will not run. The patterns defined in `branches-ignore` are evaluated against the Git ref's name. For example, the following workflow would run whenever there is a `pull_request` event unless the pull request is targeting:

- A branch named `mona/octocat` (`refs/heads/mona/octocat`)
- A branch whose name matches `releases/**-alpha`, like `releases/beta/3-alpha` (`refs/heads/releases/beta/3-alpha`)

```yaml
on:
  pull_request:
    # Sequence of patterns matched against refs/heads
    branches-ignore:
      - 'mona/octocat'
      - 'releases/**-alpha'
```

Example: Including and excluding branches

You cannot use `branches` and `branches-ignore` to filter the same event in a single workflow. If you want to both include and exclude branch patterns for a single event, use the `branches` filter along with the `!` character to indicate which branches should be excluded.

If you define a branch with the `!` character, you must also define at least one branch without the `!` character. If you only want to exclude branches, use `branches-ignore` instead.

The order that you define patterns matters.

- A matching negative pattern (prefixed with `!`) after a positive match will exclude the Git ref.

- A matching positive pattern after a negative match will include the Git ref again.

The following workflow will run on `pull_request` events for pull requests that target `releases/10` or `releases/beta/mona`, but not for pull requests that target `releases/10-alpha` or `releases/beta/3-alpha` because the negative pattern `!releases/**-alpha` follows the positive pattern.

```yaml
on:
  pull_request:
    branches:
      - 'releases/**'
      - '!releases/**-alpha'
```

Using filters to target specific branches or tags for push events

When using the `push` event, you can configure a workflow to run on specific branches or tags.

Use the `branches` filter when you want to include branch name patterns or when you want to both include and exclude branch names patterns. Use the `branches-ignore` filter when you only want to exclude branch name patterns. You cannot use both the `branches` and `branches-ignore` filters for the same event in a workflow.

Use the `tags` filter when you want to include tag name patterns or when you want to both include and exclude tag names patterns. Use the `tags-ignore` filter when you only want to exclude tag name patterns. You cannot use both the `tags` and `tags-ignore` filters for the same event in a workflow.

If you define only `tags` / `tags-ignore` or only `branches` / `branches-ignore`, the workflow won't run for events affecting the undefined Git ref. If you define neither `tags` / `tags-ignore` or `branches` / `branches-ignore`, the workflow will run for events affecting either branches or tags. If you define both `branches` / `branches-ignore` and `paths` / `paths-ignore`, the workflow will only run when both filters are satisfied.

The `branches`, `branches-ignore`, `tags`, and `tags-ignore` keywords accept glob patterns that use characters like `*`, `**`, `+`, `?`, `!` and others to match more than one branch or tag name. If a name contains any of these characters and you want a literal match, you need to *escape* each of these special characters with `\`. For more information about glob patterns, see the "Workflow syntax for GitHub Actions."

Example: Including branches and tags

The patterns defined in `branches` and `tags` are evaluated against the Git ref's name. For example, the following workflow would run whenever there is a `push` event to:

- A branch named `main` (`refs/heads/main`)
- A branch named `mona/octocat` (`refs/heads/mona/octocat`)
- A branch whose name starts with `releases/`, like `releases/10` (`refs/heads/releases/10`)
- A tag named `v2` (`refs/tags/v2`)
- A tag whose name starts with `v1.`, like `v1.9.1` (`refs/tags/v1.9.1`)

```yaml
on:
  push:
    # Sequence of patterns matched against refs/heads
    branches:
      - main
      - 'mona/octocat'
      - 'releases/**'
```

```
    # Sequence of patterns matched against refs/tags
    tags:
      - v2
      - v1.*
```

Example: Excluding branches and tags

When a pattern matches the `branches-ignore` or `tags-ignore` pattern, the workflow will not run. The patterns defined in `branches` and `tags` are evaluated against the Git ref's name. For example, the following workflow would run whenever there is a `push` event, unless the `push` event is to:

- A branch named `mona/octocat` (`refs/heads/mona/octocat`)
- A branch whose name matches `releases/**-alpha`, like `releases/beta/3-alpha` (`refs/heads/releases/beta/3-alpha`)
- A tag named `v2` (`refs/tags/v2`)
- A tag whose name starts with `v1.`, like `v1.9` (`refs/tags/v1.9`)

```
on:
  push:
    # Sequence of patterns matched against refs/heads
    branches-ignore:
      - 'mona/octocat'
      - 'releases/**-alpha'
    # Sequence of patterns matched against refs/tags
    tags-ignore:
      - v2
      - v1.*
```

Example: Including and excluding branches and tags

You can't use `branches` and `branches-ignore` to filter the same event in a single workflow. Similarly, you can't use `tags` and `tags-ignore` to filter the same event in a single workflow. If you want to both include and exclude branch or tag patterns for a single event, use the `branches` or `tags` filter along with the `!` character to indicate which branches or tags should be excluded.

If you define a branch with the `!` character, you must also define at least one branch without the `!` character. If you only want to exclude branches, use `branches-ignore` instead. Similarly, if you define a tag with the `!` character, you must also define at least one tag without the `!` character. If you only want to exclude tags, use `tags-ignore` instead.

The order that you define patterns matters.

- A matching negative pattern (prefixed with `!`) after a positive match will exclude the Git ref.
- A matching positive pattern after a negative match will include the Git ref again.

The following workflow will run on pushes to `releases/10` or `releases/beta/mona`, but not on `releases/10-alpha` or `releases/beta/3-alpha` because the negative pattern `!releases/**-alpha` follows the positive pattern.

```
on:
  push:
    branches:
      - 'releases/**'
      - '!releases/**-alpha'
```

Using filters to target specific paths for pull request or push events

When using the `push` and `pull_request` events, you can configure a workflow to run based on what file paths are changed. Path filters are not evaluated for pushes of tags.

Use the `paths` filter when you want to include file path patterns or when you want to both include and exclude file path patterns. Use the `paths-ignore` filter when you only want to exclude file path patterns. You cannot use both the `paths` and `paths-ignore` filters for the same event in a workflow. If you want to both include and exclude path patterns for a single event, use the `paths` filter prefixed with the `!` character to indicate which paths should be excluded.

> **Note:** The order that you define `paths` patterns matters:
>
> - A matching negative pattern (prefixed with `!`) after a positive match will exclude the path.
> - A matching positive pattern after a negative match will include the path again.

If you define both `branches` / `branches-ignore` and `paths` / `paths-ignore`, the workflow will only run when both filters are satisfied.

The `paths` and `paths-ignore` keywords accept glob patterns that use the `*` and `**` wildcard characters to match more than one path name. For more information, see the "Workflow syntax for GitHub Actions."

Example: Including paths

If at least one path matches a pattern in the `paths` filter, the workflow runs. For example, the following workflow would run anytime you push a JavaScript file (`.js`).

```yaml
on:
  push:
    paths:
      - '**.js'
```

> **Note:** You should not use path filtering to skip workflow runs if the workflow has been configured to be required. For more information, see "Required workflows."
>
> If a workflow is skipped due to path filtering, branch filtering, or a commit message, then checks associated with that workflow will remain in a "Pending" state. A pull request that requires those checks to be successful will be blocked from merging.

Example: Excluding paths

When all the path names match patterns in `paths-ignore`, the workflow will not run. If any path names do not match patterns in `paths-ignore`, even if some path names match the patterns, the workflow will run.

A workflow with the following path filter will only run on `push` events that include at least one file outside the `docs` directory at the root of the repository.

```yaml
on:
  push:
    paths-ignore:
      - 'docs/**'
```

Example: Including and excluding paths

You can not use `paths` and `paths-ignore` to filter the same event in a single workflow. If you want to both include and exclude path patterns for a single event, use the `paths` filter prefixed with the `!` character to indicate which paths should be excluded.

If you define a path with the `!` character, you must also define at least one path without the `!` character. If you only want to exclude paths, use `paths-ignore` instead.

The order that you define `paths` patterns matters:

- A matching negative pattern (prefixed with `!`) after a positive match will exclude the path.
- A matching positive pattern after a negative match will include the path again.

This example runs anytime the `push` event includes a file in the `sub-project` directory or its subdirectories, unless the file is in the `sub-project/docs` directory. For example, a push that changed `sub-project/index.js` or `sub-project/src/index.js` will trigger a workflow run, but a push changing only `sub-project/docs/readme.md` will not.

```yaml
on:
  push:
    paths:
      - 'sub-project/**'
      - '!sub-project/docs/**'
```

Git diff comparisons

> **Note:** If you push more than 1,000 commits, or if GitHub does not generate the diff due to a timeout, the workflow will always run.

The filter determines if a workflow should run by evaluating the changed files and running them against the `paths-ignore` or `paths` list. If there are no files changed, the workflow will not run.

GitHub generates the list of changed files using two-dot diffs for pushes and three-dot diffs for pull requests:

- **Pull requests:** Three-dot diffs are a comparison between the most recent version of the topic branch and the commit where the topic branch was last synced with the base branch.
- **Pushes to existing branches:** A two-dot diff compares the head and base SHAs directly with each other.
- **Pushes to new branches:** A two-dot diff against the parent of the ancestor of the deepest commit pushed.

Diffs are limited to 300 files. If there are files changed that aren't matched in the first 300 files returned by the filter, the workflow will not run. You may need to create more specific filters so that the workflow will run automatically.

For more information, see "About comparing branches in pull requests."

Using filters to target specific branches for workflow run events

When using the `workflow_run` event, you can specify what branches the triggering workflow must run on in order to trigger your workflow.

The `branches` and `branches-ignore` filters accept glob patterns that use characters like `*`, `**`, `+`, `?`, `!` and others to match more than one branch name. If a name contains any of these characters and you want a literal match, you need to *escape* each of these special characters with `\`. For more information about glob patterns, see the "Workflow

syntax for GitHub Actions."

For example, a workflow with the following trigger will only run when the workflow named `Build` runs on a branch whose name starts with `releases/`:

```yaml
on:
  workflow_run:
    workflows: ["Build"]
    types: [requested]
    branches:
      - 'releases/**'
```

A workflow with the following trigger will only run when the workflow named `Build` runs on a branch that is not named `canary`:

```yaml
on:
  workflow_run:
    workflows: ["Build"]
    types: [requested]
    branches-ignore:
      - "canary"
```

You cannot use both the `branches` and `branches-ignore` filters for the same event in a workflow. If you want to both include and exclude branch patterns for a single event, use the `branches` filter along with the `!` character to indicate which branches should be excluded.

The order that you define patterns matters.

- A matching negative pattern (prefixed with `!`) after a positive match will exclude the branch.
- A matching positive pattern after a negative match will include the branch again.

For example, a workflow with the following trigger will run when the workflow named `Build` runs on a branch that is named `releases/10` or `releases/beta/mona` but will not `releases/10-alpha`, `releases/beta/3-alpha`, or `main`.

```yaml
on:
  workflow_run:
    workflows: ["Build"]
    types: [requested]
    branches:
      - 'releases/**'
      - '!releases/**-alpha'
```

Defining inputs for manually triggered workflows

When using the `workflow_dispatch` event, you can optionally specify inputs that are passed to the workflow. The triggered workflow receives the inputs in the `inputs` context. For more information, see "Contexts."

Notes:

- The workflow will also receive the inputs in the `github.event.inputs` context. The information in the `inputs` context and `github.event.inputs` context is identical except that the `inputs` context preserves Boolean values as Booleans instead of converting them to strings. The `choice` type resolves to a string and is a single selectable option.
- The maximum number of top-level properties for `inputs` is 10.
- The maximum payload for `inputs` is 65,535 characters.

```yaml
on:
  workflow_dispatch:
    inputs:
      logLevel:
        description: 'Log level'
        required: true
        default: 'warning'
        type: choice
        options:
          - info
          - warning
          - debug
      print_tags:
        description: 'True to print to STDOUT'
        required: true
        type: boolean
      tags:
        description: 'Test scenario tags'
        required: true
        type: string
      environment:
        description: 'Environment to run tests against'
        type: environment
        required: true

jobs:
  print-tag:
    runs-on: ubuntu-latest
    if:  ${{ inputs.print_tags }}
    steps:
      - name: Print the input tag to STDOUT
        run: echo  The tags are ${{ inputs.tags }}
```

Defining inputs, outputs, and secrets for reusable workflows

You can define inputs and secrets that a reusable workflow should receive from a calling workflow. You can also specify outputs that a reusable workflow will make available to a calling workflow. For more information, see "Reusing workflows."

Using event information

Information about the event that triggered a workflow run is available in the `github.event` context. The properties in the `github.event` context depend on the type of event that triggered the workflow. For example, a workflow triggered when an issue is labeled would have information about the issue and label.

Viewing all properties of an event

Reference the webhook event documentation for common properties and example payloads. For more information, see "Webhook events and payloads."

You can also print the entire `github.event` context to see what properties are available for the event that triggered your workflow:

```yaml
jobs:
  print_context:
    runs-on: ubuntu-latest
    steps:
      - env:
          EVENT_CONTEXT: ${{ toJSON(github.event) }}
        run: |
```

```
          echo $EVENT_CONTEXT
```

Accessing and using event properties

You can use the `github.event` context in your workflow. For example, the following workflow runs when a pull request that changes `package*.json`, `.github/CODEOWNERS`, or `.github/workflows/**` is opened. If the pull request author (`github.event.pull_request.user.login`) is not `octobot` or `dependabot[bot]`, then the workflow uses the GitHub CLI to label and comment on the pull request (`github.event.pull_request.number`).

```yaml
on:
  pull_request:
    types:
      - opened
    paths:
      - '.github/workflows/**'
      - '.github/CODEOWNERS'
      - 'package*.json'

jobs:
  triage:
    if: >-
      github.event.pull_request.user.login != 'octobot' &&
      github.event.pull_request.user.login != 'dependabot[bot]'
    runs-on: ubuntu-latest
    steps:
      - name: "Comment about changes we can't accept"
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          PR: ${{ github.event.pull_request.html_url }}
        run: |
          gh pr edit $PR --add-label 'invalid'
          gh pr comment $PR --body 'It looks like you edited `package*.json`, `.github/CODEOWNERS`, or `.github/workflows/**`. We do not allow contributions to these files. Please review our [contributing guidelines](https://github.com/octo-org/octo-repo/blob/main/CONTRIBUTING.md) for what contributions are accepted.'
```

For more information about contexts, see "Contexts." For more information about event payloads, see "Webhook events and payloads."

Further controlling how your workflow will run

If you want more granular control than events, event activity types, or event filters provide, you can use conditionals and environments to control whether individual jobs or steps in your workflow will run.

Using conditionals

You can use conditionals to further control whether jobs or steps in your workflow will run.

Example using a value in the event payload

For example, if you want the workflow to run when a specific label is added to an issue, you can trigger on the `issues labeled` event activity type and use a conditional to check what label triggered the workflow. The following workflow will run when any label is added to an issue in the workflow's repository, but the `run_if_label_matches` job will only execute if the label is named `bug`.

```yaml
on:
  issues:
    types:
      - labeled

jobs:
  run_if_label_matches:
    if: github.event.label.name == 'bug'
    runs-on: ubuntu-latest
    steps:
      - run: echo 'The label was bug'
```

Example using event type

For example, if you want to run different jobs or steps depending on what event triggered the workflow, you can use a conditional to check whether a specific event type exists in the event context. The following workflow will run whenever an issue or pull request is closed. If the workflow ran because an issue was closed, the `github.event` context will contain a value for `issue` but not for `pull_request`. Therefore, the `if_issue` step will run but the `if_pr` step will not run. Conversely, if the workflow ran because a pull request was closed, the `if_pr` step will run but the `if_issue` step will not run.

```yaml
on:
  issues:
    types:
      - closed
  pull_request:
    types:
      - closed

jobs:
  state_event_type:
    runs-on: ubuntu-latest
    steps:
    - name: if_issue
      if: github.event.issue
      run: |
        echo An issue was closed
    - name: if_pr
      if: github.event.pull_request
      run: |
        echo A pull request was closed
```

For more information about what information is available in the event context, see "Using event information." For more information about how to use conditionals, see "Expressions."

Using environments to manually trigger workflow jobs

If you want to manually trigger a specific job in a workflow, you can use an environment that requires approval from a specific team or user. First, configure an environment with required reviewers. For more information, see "Using environments for deployment." Then, reference the environment name in a job in your workflow using the `environment:` key. Any job referencing the environment will not run until at least one reviewer approves the job.

For example, the following workflow will run whenever there is a push to main. The `build` job will always run. The `publish` job will only run after the `build` job successfully completes (due to `needs: [build]`) and after all of the rules (including required reviewers) for the environment called `production` pass (due to `environment: production`).

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

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: build
        echo 'building'

  publish:
    needs: [build]
    runs-on: ubuntu-latest
    environment: production
    steps:
      - name: publish
        echo 'publishing'
```

Environments, environment secrets, and deployment protection rules are available in public repositories for all current GitHub plans. They are not available on legacy plans, such as Bronze, Silver, or Gold. For access to environments, environment secrets, and deployment branches in private or internal repositories, you must use GitHub Pro, GitHub Team, or GitHub Enterprise.

Available events

For a full list of available events, see "Events that trigger workflows."