This version of GitHub Enterprise was discontinued on 2023-03-15. No patch releases will be made, even for critical security issues. For better performance, improved security, and new features, upgrade to the latest version of GitHub Enterprise. For help with the upgrade, contact GitHub Enterprise support.

Metadata syntax for GitHub Actions

In this article

About YAML syntax for GitHub Actions
name
author
description
inputs
outputs for Docker container and JavaScript actions
outputs for composite actions
runs
runs for JavaScript actions
runs for composite actions
runs for Docker container actions
branding

You can create actions to perform tasks in your repository. Actions require a metadata file that uses YAML syntax.

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 YAML syntax for GitHub Actions

All actions require a metadata file. The metadata filename must be either `action.yml` or `action.yaml`. The data in the metadata file defines the inputs, outputs, and runs configuration for your action.

Action metadata files use YAML syntax. If you're new to YAML, you can read "Learn YAML in five minutes."

`name`

Required The name of your action. GitHub displays the `name` in the **Actions** tab to help visually identify actions in each job.

`author`

Optional The name of the action's author.

`description`

Required A short description of the action.

`inputs`

Optional Input parameters allow you to specify data that the action expects to use during runtime. GitHub stores input parameters as environment variables. Input ids with uppercase letters are converted to lowercase during runtime. We recommend using lowercase input ids.

Example: Specifying inputs

This example configures two inputs: `num-octocats` and `octocat-eye-color`. The `num-octocats` input is not required and will default to a value of '1'; `octocat-eye-color` is required and has no default value. Workflow files that use this action must use the `with` keyword to set an input value for `octocat-eye-color`. For more information about the `with` syntax, see "Workflow syntax for GitHub Actions."

```yaml
inputs:
  num-octocats:
    description: 'Number of Octocats'
    required: false
    default: '1'
  octocat-eye-color:
    description: 'Eye color of the Octocats'
    required: true
```

When you specify an input in a workflow file or use a default input value, GitHub creates an environment variable for the input with the name `INPUT_<VARIABLE_NAME>`. The environment variable created converts input names to uppercase letters and replaces spaces with `_` characters.

If the action is written using a composite, then it will not automatically get `INPUT_<VARIABLE_NAME>`. If the conversion doesn't occur, you can change these inputs manually.

To access the environment variable in a Docker container action, you must pass the input using the `args` keyword in the action metadata file. For more information about the action metadata file for Docker container actions, see "Creating a Docker container action."

For example, if a workflow defined the `num-octocats` and `octocat-eye-color` inputs, the action code could read the values of the inputs using the `INPUT_NUM-OCTOCATS` and `INPUT_OCTOCAT-EYE-COLOR` environment variables.

`inputs.<input_id>`

Required A `string` identifier to associate with the input. The value of `<input_id>` is a map of the input's metadata. The `<input_id>` must be a unique identifier within the `inputs` object. The `<input_id>` must start with a letter or `_` and contain only alphanumeric characters, `-`, or `_`.

`inputs.<input_id>.description`

Required A `string` description of the input parameter.

`inputs.<input_id>.required`

Optional A `boolean` to indicate whether the action requires the input parameter. Set to `true` when the parameter is required.

`inputs.<input_id>.default`

Optional A `string` representing the default value. The default value is used when an input parameter isn't specified in a workflow file.

`inputs.<input_id>.deprecationMessage`

Optional If the input parameter is used, this `string` is logged as a warning message. You can use this warning to notify users that the input is deprecated and mention any alternatives.

`outputs` for Docker container and JavaScript actions

Optional Output parameters allow you to declare data that an action sets. Actions that run later in a workflow can use the output data set in previously run actions. For example, if you had an action that performed the addition of two inputs (x + y = z), the action could output the sum (z) for other actions to use as an input.

Outputs are Unicode strings, and can be a maximum of 1 MB. The total of all outputs in a workflow run can be a maximum of 50 MB.

If you don't declare an output in your action metadata file, you can still set outputs and use them in a workflow. For more information on setting outputs in an action, see "Workflow commands for GitHub Actions."

Example: Declaring outputs for Docker container and JavaScript actions

```yaml
outputs:
  sum: # id of the output
    description: 'The sum of the inputs'
```

`outputs.<output_id>`

Required A `string` identifier to associate with the output. The value of `<output_id>` is a map of the output's metadata. The `<output_id>` must be a unique identifier within the `outputs` object. The `<output_id>` must start with a letter or `_` and contain only alphanumeric characters, `-`, or `_`.

`outputs.<output_id>.description`

Required A `string` description of the output parameter.

`outputs` for composite actions

Optional `outputs` use the same parameters as `outputs.<output_id>` and `outputs.<output_id>.description` (see "`outputs` for Docker container and JavaScript actions"), but also includes the `value` token.

Outputs are Unicode strings, and can be a maximum of 1 MB. The total of all outputs in a workflow run can be a maximum of 50 MB.

Example: Declaring outputs for composite actions

```yaml
outputs:
  random-number:
    description: "Random number"
    value: ${{ steps.random-number-generator.outputs.random-id }}
runs:
  using: "composite"
  steps:
    - id: random-number-generator
      run: echo "::set-output name=random-id::$(echo $RANDOM)"
      shell: bash
```

`outputs.<output_id>.value`

Required The value that the output parameter will be mapped to. You can set this to a `string` or an expression with context. For example, you can use the `steps` context to set the `value` of an output to the output value of a step.

For more information on how to use context syntax, see "Contexts."

`runs`

Required Specifies whether this is a JavaScript action, a composite action, or a Docker container action and how the action is executed.

`runs` for JavaScript actions

Required Configures the path to the action's code and the runtime used to execute the code.

Example: Using Node.js v16

```yaml
runs:
  using: 'node16'
  main: 'main.js'
```

`runs.using` for JavaScript actions

Required The runtime used to execute the code specified in `main`.

- Use `node16` for Node.js v16.

`runs.main`

Required The file that contains your action code. The runtime specified in `using` executes this file.

`runs.pre`

Optional Allows you to run a script at the start of a job, before the `main:` action begins. For example, you can use `pre:` to run a prerequisite setup script. The runtime specified with the `using` syntax will execute this file. The `pre:` action always runs by default but you can override this using `runs.pre-if`.

In this example, the `pre:` action runs a script called `setup.js`:

```
runs:
  using: 'node16'
  pre: 'setup.js'
  main: 'index.js'
  post: 'cleanup.js'
```

`runs.pre-if`

Optional Allows you to define conditions for the `pre:` action execution. The `pre:` action will only run if the conditions in `pre-if` are met. If not set, then `pre-if` defaults to `always()`. In `pre-if`, status check functions evaluate against the job's status, not the action's own status.

Note that the `step` context is unavailable, as no steps have run yet.

In this example, `cleanup.js` only runs on Linux-based runners:

```
  pre: 'cleanup.js'
  pre-if: runner.os == 'linux'
```

`runs.post`

Optional Allows you to run a script at the end of a job, once the `main:` action has completed. For example, you can use `post:` to terminate certain processes or remove unneeded files. The runtime specified with the `using` syntax will execute this file.

In this example, the `post:` action runs a script called `cleanup.js`:

```
runs:
  using: 'node16'
  main: 'index.js'
  post: 'cleanup.js'
```

The `post:` action always runs by default but you can override this using `post-if`.

`runs.post-if`

Optional Allows you to define conditions for the `post:` action execution. The `post:` action will only run if the conditions in `post-if` are met. If not set, then `post-if` defaults to `always()`. In `post-if`, status check functions evaluate against the job's status, not the action's own status.

For example, this `cleanup.js` will only run on Linux-based runners:

```
  post: 'cleanup.js'
  post-if: runner.os == 'linux'
```

`runs` for composite actions

Required Configures the path to the composite action.

`runs.using` for composite actions

Required You must set this value to `'composite'`.

`runs.steps`

Required The steps that you plan to run in this action. These can be either `run` steps or `uses` steps.

`runs.steps[*].run`

Optional The command you want to run. This can be inline or a script in your action repository:

```yaml
runs:
  using: "composite"
  steps:
    - run: ${{ github.action_path }}/test/script.sh
      shell: bash
```

Alternatively, you can use `$GITHUB_ACTION_PATH`:

```yaml
runs:
  using: "composite"
  steps:
    - run: $GITHUB_ACTION_PATH/script.sh
      shell: bash
```

For more information, see "Contexts".

`runs.steps[*].shell`

Optional The shell where you want to run the command. You can use any of the shells listed here. Required if `run` is set.

`runs.steps[*].if`

Optional You can use the `if` conditional to prevent a step from running unless a condition is met. You can use any supported context and expression to create a conditional.

When you use expressions in an `if` conditional, you may omit the expression syntax (`${{ }}`) because GitHub automatically evaluates the `if` conditional as an expression. For more information, see "Expressions."

Example: Using contexts

This step only runs when the event type is a `pull_request` and the event action is `unassigned`.

```yaml
steps:
  - run: echo This event is a pull request that had an assignee removed.
    if: ${{ github.event_name == 'pull_request' && github.event.action == 'unassigne
```

Example: Using status check functions

The `my backup step` only runs when the previous step of a composite action fails. For more information, see "Expressions."

```yaml
steps:
  - name: My first step
    uses: octo-org/action-name@main
  - name: My backup step
    if: ${{ failure() }}
    uses: actions/heroku@1.0.0
```

`runs.steps[*].name`

Optional The name of the composite step.

`runs.steps[*].id`

Optional A unique identifier for the step. You can use the `id` to reference the step in contexts. For more information, see "Contexts."

`runs.steps[*].env`

Optional Sets a `map` of environment variables for only that step. If you want to modify the environment variable stored in the workflow, use `echo "{name}={value}" >> $GITHUB_ENV` in a composite step.

`runs.steps[*].working-directory`

Optional Specifies the working directory where the command is run.

`runs.steps[*].uses`

Optional Selects an action to run as part of a step in your job. An action is a reusable unit of code. You can use an action defined in the same repository as the workflow, a public repository, or in a published Docker container image.

We strongly recommend that you include the version of the action you are using by specifying a Git ref, SHA, or Docker tag number. If you don't specify a version, it could break your workflows or cause unexpected behavior when the action owner publishes an update.

- Using the commit SHA of a released action version is the safest for stability and security.
- Using the specific major action version allows you to receive critical fixes and security patches while still maintaining compatibility. It also assures that your workflow should still work.
- Using the default branch of an action may be convenient, but if someone releases a new major version with a breaking change, your workflow could break.

Some actions require inputs that you must set using the `with` keyword. Review the action's README file to determine the inputs required.

```yaml
runs:
  using: "composite"
  steps:
    # Reference a specific commit
    - uses: actions/checkout@8f4b7f84864484a7bf31766abe9204da3cbe65b3
    # Reference the major version of a release
    - uses: actions/checkout@v2
    # Reference a specific version
    - uses: actions/checkout@v2.2.0
    # Reference a branch
    - uses: actions/checkout@main
    # References a subdirectory in a public GitHub repository at a specific branch,
    - uses: actions/aws/ec2@main
    # References a local action
    - uses: ./.github/actions/my-action
    # References a docker public registry action
    - uses: docker://gcr.io/cloud-builders/gradle
    # Reference a docker image published on docker hub
    - uses: docker://alpine:3.8
```

`runs.steps[*].with`

Optional A `map` of the input parameters defined by the action. Each input parameter is a key/value pair. For more information, see Example: Specifying inputs.

```
runs:
  using: "composite"
  steps:
    - name: My first step
      uses: actions/hello_world@main
      with:
        first_name: Mona
        middle_name: The
        last_name: Octocat
```

`runs` for Docker container actions

Required Configures the image used for the Docker container action.

Example: Using a Dockerfile in your repository

```
runs:
  using: 'docker'
  image: 'Dockerfile'
```

Example: Using public Docker registry container

```
runs:
  using: 'docker'
  image: 'docker://debian:stretch-slim'
```

`runs.using` for Docker container actions

Required You must set this value to `'docker'`.

`runs.pre-entrypoint`

Optional Allows you to run a script before the `entrypoint` action begins. For example, you can use `pre-entrypoint:` to run a prerequisite setup script. GitHub Actions uses `docker run` to launch this action, and runs the script inside a new container that uses the same base image. This means that the runtime state is different from the main `entrypoint` container, and any states you require must be accessed in either the workspace, `HOME`, or as a `STATE_` variable. The `pre-entrypoint:` action always runs by default but you can override this using `runs.pre-if`.

The runtime specified with the `using` syntax will execute this file.

In this example, the `pre-entrypoint:` action runs a script called `setup.sh`:

```
runs:
  using: 'docker'
  image: 'Dockerfile'
  args:
    - 'bzz'
  pre-entrypoint: 'setup.sh'
  entrypoint: 'main.sh'
```

`runs.image`

Required The Docker image to use as the container to run the action. The value can be the Docker base image name, a local `Dockerfile` in your repository, or a public image in Docker Hub or another registry. To reference a `Dockerfile` local to your repository, the file must be named `Dockerfile` and you must use a path relative to your action metadata file. The `docker` application will execute this file.

`runs.env`

Optional Specifies a key/value map of environment variables to set in the container environment.

`runs.entrypoint`

Optional Overrides the Docker `ENTRYPOINT` in the `Dockerfile`, or sets it if one wasn't already specified. Use `entrypoint` when the `Dockerfile` does not specify an `ENTRYPOINT` or you want to override the `ENTRYPOINT` instruction. If you omit `entrypoint`, the commands you specify in the Docker `ENTRYPOINT` instruction will execute. The Docker `ENTRYPOINT` instruction has a *shell* form and *exec* form. The Docker `ENTRYPOINT` documentation recommends using the *exec* form of the `ENTRYPOINT` instruction.

For more information about how the `entrypoint` executes, see "Dockerfile support for GitHub Actions."

`runs.post-entrypoint`

Optional Allows you to run a cleanup script once the `runs.entrypoint` action has completed. GitHub Actions uses `docker run` to launch this action. Because GitHub Actions runs the script inside a new container using the same base image, the runtime state is different from the main `entrypoint` container. You can access any state you need in either the workspace, `HOME`, or as a `STATE_` variable. The `post-entrypoint:` action always runs by default but you can override this using `runs.post-if`.

```
runs:
  using: 'docker'
  image: 'Dockerfile'
  args:
    - 'bzz'
  entrypoint: 'main.sh'
  post-entrypoint: 'cleanup.sh'
```

`runs.args`

Optional An array of strings that define the inputs for a Docker container. Inputs can include hardcoded strings. GitHub passes the `args` to the container's `ENTRYPOINT` when the container starts up.

The `args` are used in place of the `CMD` instruction in a `Dockerfile`. If you use `CMD` in your `Dockerfile`, use the guidelines ordered by preference:

1. Document required arguments in the action's README and omit them from the `CMD` instruction.
2. Use defaults that allow using the action without specifying any `args`.
3. If the action exposes a `--help` flag, or something similar, use that to make your action self-documenting.

If you need to pass environment variables into an action, make sure your action runs a command shell to perform variable substitution. For example, if your `entrypoint` attribute is set to `"sh -c"`, `args` will be run in a command shell. Alternatively, if your `Dockerfile` uses an `ENTRYPOINT` to run the same command (`"sh -c"`), `args` will execute in a command shell.

For more information about using the `CMD` instruction with GitHub Actions, see "Dockerfile support for GitHub Actions."

Example: Defining arguments for the Docker container

```yaml
runs:
  using: 'docker'
  image: 'Dockerfile'
  args:
    - ${{ inputs.greeting }}
    - 'foo'
    - 'bar'
```

`branding`

Optional You can use a color and Feather icon to create a badge to personalize and distinguish your action. Badges are shown next to your action name in GitHub Marketplace.

Example: Configuring branding for an action

```yaml
branding:
  icon: 'award'
  color: 'green'
```

`branding.color`

The background color of the badge. Can be one of: `white`, `yellow`, `blue`, `green`, `orange`, `red`, `purple`, or `gray-dark`.

`branding.icon`

The name of the v4.28.0 Feather icon to use.

Omitted icons

Brand icons, and all the following icons, are omitted.

- coffee

- columns
- divide-circle
- divide-square
- divide
- frown
- hexagon
- key
- meh
- mouse-pointer
- smile
- tool
- x-octagon

Exhaustive list of all currently supported icons

- activity
- airplay
- alert-circle
- alert-octagon
- alert-triangle
- align-center
- align-justify
- align-left
- align-right
- anchor
- aperture
- archive
- arrow-down-circle
- arrow-down-left
- arrow-down-right
- arrow-down
- arrow-left-circle
- arrow-left
- arrow-right-circle
- arrow-right
- arrow-up-circle
- arrow-up-left
- arrow-up-right
- arrow-up
- at-sign
- award
- bar-chart-2
- bar-chart
- battery-charging
- battery
- bell-off
- bell
- bluetooth
- bold
- book-open
- book

- bookmark
- box
- briefcase
- calendar
- camera-off
- camera
- cast
- check-circle
- check-square
- check
- chevron-down
- chevron-left
- chevron-right
- chevron-up
- chevrons-down
- chevrons-left
- chevrons-right
- chevrons-up
- circle
- clipboard
- clock
- cloud-drizzle
- cloud-lightning
- cloud-off
- cloud-rain
- cloud-snow
- cloud
- code
- command
- compass
- copy
- corner-down-left
- corner-down-right
- corner-left-down
- corner-left-up
- corner-right-down
- corner-right-up
- corner-up-left
- corner-up-right
- cpu
- credit-card
- crop
- crosshair
- database
- delete
- disc
- dollar-sign
- download-cloud
- download
- droplet
- edit-2

- edit-3
- edit
- external-link
- eye-off
- eye
- fast-forward
- feather
- file-minus
- file-plus
- file-text
- file
- film
- filter
- flag
- folder-minus
- folder-plus
- folder
- gift
- git-branch
- git-commit
- git-merge
- git-pull-request
- globe
- grid
- hard-drive
- hash
- headphones
- heart
- help-circle
- home
- image
- inbox
- info
- italic
- layers
- layout
- life-buoy
- link-2
- link
- list
- loader
- lock
- log-in
- log-out
- mail
- map-pin
- map
- maximize-2
- maximize
- menu
- message-circle

- message-square
- mic-off
- mic
- minimize-2
- minimize
- minus-circle
- minus-square
- minus
- monitor
- moon
- more-horizontal
- more-vertical
- move
- music
- navigation-2
- navigation
- octagon
- package
- paperclip
- pause-circle
- pause
- percent
- phone-call
- phone-forwarded
- phone-incoming
- phone-missed
- phone-off
- phone-outgoing
- phone
- pie-chart
- play-circle
- play
- plus-circle
- plus-square
- plus
- pocket
- power
- printer
- radio
- refresh-ccw
- refresh-cw
- repeat
- rewind
- rotate-ccw
- rotate-cw
- rss
- save
- scissors
- search
- send
- server

- settings
- share-2
- share
- shield-off
- shield
- shopping-bag
- shopping-cart
- shuffle
- sidebar
- skip-back
- skip-forward
- slash
- sliders
- smartphone
- speaker
- square
- star
- stop-circle
- sun
- sunrise
- sunset
- tablet
- tag
- target
- terminal
- thermometer
- thumbs-down
- thumbs-up
- toggle-left
- toggle-right
- trash-2
- trash
- trending-down
- trending-up
- triangle
- truck
- tv
- type
- umbrella
- underline
- unlock
- upload-cloud
- upload
- user-check
- user-minus
- user-plus
- user-x
- user
- users
- video-off
- video

- voicemail
- volume-1
- volume-2
- volume-x
- volume
- watch
- wifi-off
- wifi
- wind
- x-circle
- x-square
- x
- zap-off
- zap
- zoom-in
- zoom-out