Variables

In this article

GitHub sets default variables for each GitHub Actions workflow run. You can also set custom variables for use in a single workflow or multiple workflows.

About variables

Variables provide a way to store and reuse non-sensitive configuration information. You can store any configuration data such as compiler flags, usernames, or server names as variables. Variables are interpolated on the runner machine that runs your workflow. Commands that run in actions or workflow steps can create, read, and modify variables.

You can set your own custom variables or use the default environment variables that GitHub sets automatically. For more information, see "Default environment variables".

You can set a custom variable in two ways.

- To define an environment variable for use in a single workflow, you can use the `env` key in the workflow file. For more information, see "Defining environment variables for a single workflow".
- To define a configuration variable across multiple workflows, you can define it at the organization, repository, or environment level. For more information, see "Defining configuration variables for multiple workflows".

Warning: By default, variables render unmasked in your build outputs. If you need greater security for sensitive information, such as passwords, use secrets instead. For more information, see "Using secrets in GitHub Actions".

Defining environment variables for a single workflow

To set a custom environment variable for a single workflow, you can define it using the `env` key in the workflow file. The scope of a custom variable set by this method is limited to the element in which it is defined. You can define variables that are scoped for:

- The entire workflow, by using `env` at the top level of the workflow file.
- The contents of a job within a workflow, by using `jobs.<job_id>.env`.

- A specific step within a job, by using `jobs.<job_id>.steps[*].env`.

YAML

```yaml
name: Greeting on variable day

on:
  workflow_dispatch

env:
  DAY_OF_WEEK: Monday

jobs:
  greeting_job:
    runs-on: ubuntu-latest
    env:
      Greeting: Hello
    steps:
      - name: "Say Hello Mona it's Monday"
        run: echo "$Greeting $First_Name. Today is $DAY_OF_WEEK!"
        env:
          First_Name: Mona
```

You can access `env` variable values using runner environment variables or using contexts. The example above shows three custom variables being used as runner environment variables in an `echo` command: `$DAY_OF_WEEK`, `$Greeting`, and `$First_Name`. The values for these variables are set, and scoped, at the workflow, job, and step level respectively. The interpolation of these variables happens on the runner.

The commands in the `run` steps of a workflow, or a referenced action, are processed by the shell you are using on the runner. The instructions in the other parts of a workflow are processed by GitHub Actions and are not sent to the runner. You can use either runner environment variables or contexts in `run` steps, but in the parts of a workflow that are not sent to the runner you must use contexts to access variable values. For more information, see "Using contexts to access variable values."

Because runner environment variable interpolation is done after a workflow job is sent to a runner machine, you must use the appropriate syntax for the shell that's used on the runner. In this example, the workflow specifies `ubuntu-latest`. By default, Linux runners use the bash shell, so you must use the syntax `$NAME`. By default, Windows runners use PowerShell, so you would use the syntax `$env:NAME`. For more information about shells, see "Workflow syntax for GitHub Actions."

Naming conventions for environment variables

When you set an environment variable, you cannot use any of the default environment variable names. For a complete list of default environment variables, see "Default environment variables" below. If you attempt to override the value of one of these default variables, the assignment is ignored.

Any new variables you set that point to a location on the filesystem should have a `_PATH` suffix. The `GITHUB_ENV` and `GITHUB_WORKSPACE` default variables are exceptions to this convention.

> **Note**: You can list the entire set of environment variables that are available to a workflow step by using `run: env` in a step and then examining the output for the step.

Defining configuration variables for multiple workflows

Note: Configuration variables for GitHub Actions are in beta and subject to change.

You can create configuration variables for use across multiple workflows, and can define them at either the organization, repository, or environment level.

For example, you can use configuration variables to set default values for parameters passed to build tools at an organization level, but then allow repository owners to override these parameters on a case-by-case basis.

When you define configuration variables, they are automatically available in the `vars` context. For more information, see "Using the `vars` context to access configuration variable values".

Configuration variable precedence

If a variable with the same name exists at multiple levels, the variable at the lowest level takes precedence. For example, if an organization-level variable has the same name as a repository-level variable, then the repository-level variable takes precedence. Similarly, if an organization, repository, and environment all have a variable with the same name, the environment-level variable takes precedence.

For reusable workflows, the variables from the caller workflow's repository are used. Variables from the repository that contains the called workflow are not made available to the caller workflow.

Naming conventions for configuration variables

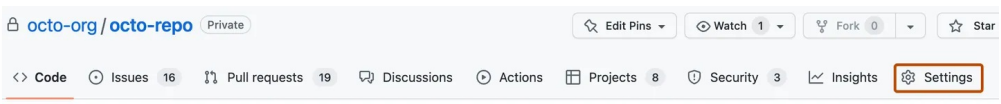
The following rules apply to configuration variable names:

- Names can only contain alphanumeric characters (`[a-z]`, `[A-Z]`, `[0-9]`) or underscores (`_`). Spaces are not allowed.
- Names must not start with the `GITHUB_` prefix.
- Names must not start with a number.
- Names are case insensitive.
- Names must be unique at the level they are created at.

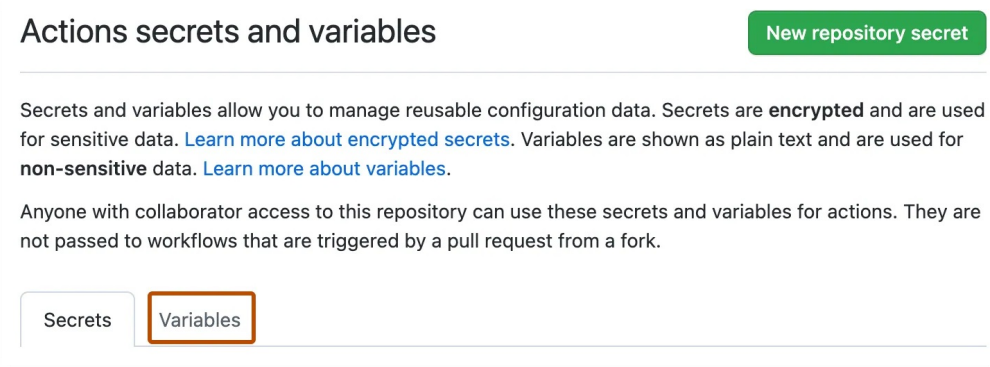
Creating configuration variables for a repository

To create secrets or variables on GitHub for a personal account repository, you must be the repository owner. To create secrets or variables on GitHub for an organization repository, you must have `admin` access. Lastly, to create secrets or variables for a personal account repository or an organization repository through the REST API, you must have collaborator access.

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**. If you cannot see the "Settings" tab, select the ··· dropdown menu, then click **Settings**.



3. In the "Security" section of the sidebar, select **Secrets and variables**, then click **Actions**.
4. Click the **Variables** tab.

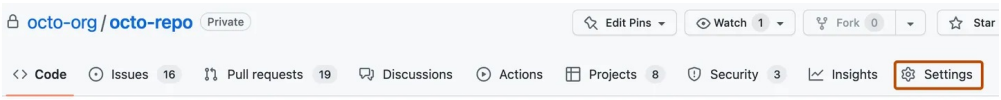


5. Click **New repository variable**.
6. In the **Name** field, enter a name for your variable.
7. In the **Value** field, enter the value for your variable.
8. Click **Add variable**.

Creating configuration variables for an environment

To create secrets or variables for an environment in a personal account repository, you must be the repository owner. To create secrets or variables for an environment in an organization repository, you must have `admin` access. For more information on environments, see "Using environments for deployment."

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Settings**. If you cannot see the "Settings" tab, select the ··· dropdown menu, then click **Settings**.



3. In the left sidebar, click **Environments**.
4. Click on the environment that you want to add a variable to.
5. Under **Environment variables**, click **Add variable**.
6. In the **Name** field, enter a name for your variable.
7. In the **Value** field, enter the value for your variable.
8. Click **Add variable**.

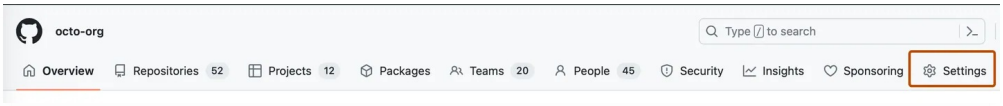
Creating configuration variables for an organization

> **Note:** Organization-level secrets and variables are not available to be used by private repositories for your plan. For more information on upgrading your GitHub subscription, see "Upgrading your account's plan".

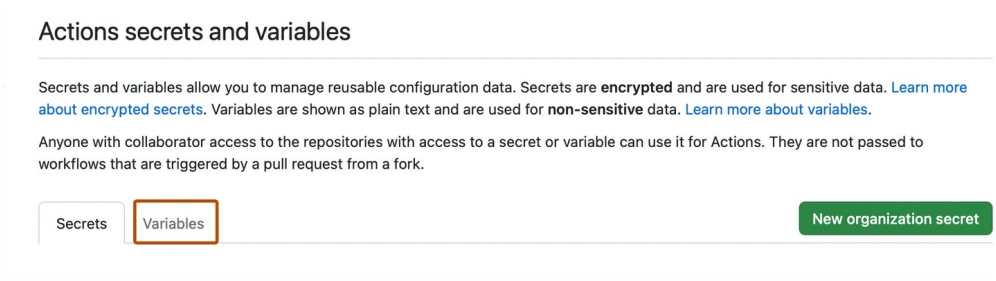
When creating a secret or variable in an organization, you can use a policy to limit access by repository. For example, you can grant access to all repositories, or limit access to only private repositories or a specified list of repositories.

To create secrets or variables at the organization level, you must be an organization owner.

1. On GitHub.com, navigate to the main page of the organization.
2. Under your organization name, click **Settings**. If you cannot see the "Settings" tab, select the ··· dropdown menu, then click **Settings**.



3. In the "Security" section of the sidebar, select **Secrets and variables**, then click **Actions**.
4. Click the **Variables** tab.



5. Click **New organization variable**.
6. In the **Name** field, enter a name for your variable.
7. In the **Value** field, enter the value for your variable.
8. From the **Repository access** dropdown list, choose an access policy.
9. Click **Add variable**.

Limits for configuration variables

Individual variables are limited to 48 KB in size.

You can store up to 1,000 organization variables, 500 variables per repository, and 100 variables per environment. The total combined size limit for organization and repository variables is 256 KB per workflow run.

A workflow created in a repository can access the following number of variables:

- Up to 500 repository variables, if the total size of repository variables is less than 256 KB. If the total size of repository variables exceeds 256 KB, only the repository variables that fall below the limit will be available (as sorted alphabetically by variable name).
- Up to 1,000 organization variables, if the total combined size of repository and organization variables is less than 256 KB. If the total combined size of organization and repository variables exceeds 256 KB, only the organization variables that fall below that limit will be available (after accounting for repository variables and as sorted alphabetically by variable name).
- Up to 100 environment-level variables.

Note: Environment-level variables do not count toward the 256 KB total size limit. If you exceed

the combined size limit for repository and organization variables and still need additional variables, you can use an environment and define additional variables in the environment.

Using contexts to access variable values

Contexts are a way to access information about workflow runs, variables, runner environments, jobs, and steps. For more information, see "Contexts". There are many other contexts that you can use for a variety of purposes in your workflows. For details of where you can use specific contexts within a workflow, see "Contexts."

You can access environment variable values using the `env` context and configuration variable values using the `vars` context.

Using the `env` context to access environment variable values

In addition to runner environment variables, GitHub Actions allows you to set and read `env` key values using contexts. Environment variables and contexts are intended for use at different points in the workflow.

The `run` steps in a workflow, or in a referenced action, are processed by a runner. As a result, you can use runner environment variables here, using the appropriate syntax for the shell you are using on the runner - for example, `$NAME` for the bash shell on a Linux runner, or `$env:NAME` for PowerShell on a Windows runner. In most cases you can also use contexts, with the syntax `${{ CONTEXT.PROPERTY }}`, to access the same value. The difference is that the context will be interpolated and replaced by a string before the job is sent to a runner.

However, you cannot use runner environment variables in parts of a workflow that are processed by GitHub Actions and are not sent to the runner. Instead, you must use contexts. For example, an `if` conditional, which determines whether a job or step is sent to the runner, is always processed by GitHub Actions. You must therefore use a context in an `if` conditional statement to access the value of an variable.

YAML

```yaml
env:
  DAY_OF_WEEK: Monday

jobs:
  greeting_job:
    runs-on: ubuntu-latest
    env:
      Greeting: Hello
    steps:
      - name: "Say Hello Mona it's Monday"
        if: ${{ env.DAY_OF_WEEK == 'Monday' }}
        run: echo "$Greeting $First_Name. Today is $DAY_OF_WEEK!"
        env:
          First_Name: Mona
```

In this modification of the earlier example, we've introduced an `if` conditional. The workflow step is now only run if `DAY_OF_WEEK` is set to "Monday". We access this value from the `if` conditional statement by using the `env` context. The `env` context is not required for the variables referenced within the `run` command. They are referenced as runner environment variables and are interpolated after the job is received by the runner. We could, however, have chosen to interpolate those variables before sending the job to the runner, by using contexts. The resulting output would be the same.

```yaml
run: echo "${{ env.Greeting }} ${{ env.First_Name }}. Today is ${{
```

```
env.DAY_OF_WEEK }}!"
```

Note: Contexts are usually denoted using the dollar sign and curly braces, as `${{ context.property }}`. In an `if` conditional, the `${{` and `}}` are optional, but if you use them they must enclose the entire comparison statement, as shown above.

You will commonly use either the `env` or `github` context to access variable values in parts of the workflow that are processed before jobs are sent to runners.

Context	Use case	Example
`env`	Reference custom variables defined in the workflow.	`${{ env.MY_VARIABLE }}`
`github`	Reference information about the workflow run and the event that triggered the run.	`${{ github.repository }}`

Warning: When creating workflows and actions, you should always consider whether your code might execute untrusted input from possible attackers. Certain contexts should be treated as untrusted input, as an attacker could insert their own malicious content. For more information, see "Security hardening for GitHub Actions."

Using the `vars` context to access configuration variable values

Configuration variables can be accessed across the workflow using `vars` context. For more information, see "Contexts".

If a configuration variable has not been set, the return value of a context referencing the variable will be an empty string.

The following example shows using configuration variables with the `vars` context across a workflow. Each of the following configuration variables have been defined at the repository, organization, or environment levels.

YAML

```yaml
on:
  workflow_dispatch:
env:
  # Setting an environment variable with the value of a configuration variable
  env_var: ${{ vars.ENV_CONTEXT_VAR }}

jobs:
  display-variables:
    name: ${{ vars.JOB_NAME }}
    # You can use configuration variables with the `vars` context for dynamic
jobs
    if: ${{ vars.USE_VARIABLES == 'true' }}
    runs-on: ${{ vars.RUNNER }}
    environment: ${{ vars.ENVIRONMENT_STAGE }}
    steps:
    - name: Use variables
      run: |
        echo "repository variable : $REPOSITORY_VAR"
        echo "organization variable : $ORGANIZATION_VAR"
        echo "overridden variable : $OVERRIDE_VAR"
        echo "variable from shell environment : $env_var"
      env:
        REPOSITORY_VAR: ${{ vars.REPOSITORY_VAR }}
```

```
        ORGANIZATION_VAR: ${{ vars.ORGANIZATION_VAR }}
        OVERRIDE_VAR: ${{ vars.OVERRIDE_VAR }}

    - name: ${{ vars.HELLO_WORLD_STEP }}
      if: ${{ vars.HELLO_WORLD_ENABLED == 'true' }}
      uses: actions/hello-world-javascript-action@main
      with:
        who-to-greet: ${{ vars.GREET_NAME }}
```

Default environment variables

The default environment variables that GitHub sets are available to every step in a workflow.

Because default environment variables are set by GitHub and not defined in a workflow, they are not accessible through the `env` context. However, most of the default variables have a corresponding, and similarly named, context property. For example, the value of the `GITHUB_REF` variable can be read during workflow processing using the `${{ github.ref }}` context property.

You can't overwrite the value of the default environment variables named `GITHUB_*` and `RUNNER_*`. Currently you can overwrite the value of the `CI` variable. However, it's not guaranteed that this will always be possible. For more information about setting environment variables, see "Defining environment variables for a single workflow" and "Workflow commands for GitHub Actions."

We strongly recommend that actions use variables to access the filesystem rather than using hardcoded file paths. GitHub sets variables for actions to use in all runner environments.

Variable	Description
`CI`	Always set to `true`.
`GITHUB_ACTION`	The name of the action currently running, or the `id` of a step. For example, for an action, `__repo-owner_name-of-action-repo`. GitHub removes special characters, and uses the name `__run` when the current step runs a script without an `id`. If you use the same script or action more than once in the same job, the name will include a suffix that consists of the sequence number preceded by an underscore. For example, the first script you run will have the name `__run`, and the second script will be named `__run_2`. Similarly, the second invocation of `actions/checkout` will be `actionscheckout2`.
`GITHUB_ACTION_PATH`	The path where an action is located. This property is only supported in composite actions. You can use this path to access files located in the same repository as the action. For example, `/home/runner/work/_actions/repo-owner/name-of-action-repo/v1`.
`GITHUB_ACTION_REPOSITORY`	For a step executing an action, this is the owner and repository name of the action. For example, `actions/checkout`.
`GITHUB_ACTIONS`	Always set to `true` when GitHub Actions is

`GITHUB_ACTIONS`	Always set to `true` when GitHub Actions is running the workflow. You can use this variable to differentiate when tests are being run locally or by GitHub Actions.
`GITHUB_ACTOR`	The name of the person or app that initiated the workflow. For example, `octocat`.
`GITHUB_ACTOR_ID`	The account ID of the person or app that triggered the initial workflow run. For example, `1234567`. Note that this is different from the actor username.
`GITHUB_API_URL`	Returns the API URL. For example: `https://api.github.com`.
`GITHUB_BASE_REF`	The name of the base ref or target branch of the pull request in a workflow run. This is only set when the event that triggers a workflow run is either `pull_request` or `pull_request_target`. For example, `main`.
`GITHUB_ENV`	The path on the runner to the file that sets variables from workflow commands. This file is unique to the current step and changes for each step in a job. For example, `/home/runner/work/_temp/_runner_file_commands/set_env_87406d6e-4979-4d42-98e1-3dab1f48b13a`. For more information, see "Workflow commands for GitHub Actions."
`GITHUB_EVENT_NAME`	The name of the event that triggered the workflow. For example, `workflow_dispatch`.
`GITHUB_EVENT_PATH`	The path to the file on the runner that contains the full event webhook payload. For example, `/github/workflow/event.json`.
`GITHUB_GRAPHQL_URL`	Returns the GraphQL API URL. For example: `https://api.github.com/graphql`.
`GITHUB_HEAD_REF`	The head ref or source branch of the pull request in a workflow run. This property is only set when the event that triggers a workflow run is either `pull_request` or `pull_request_target`. For example, `feature-branch-1`.
`GITHUB_JOB`	The job_id of the current job. For example, `greeting_job`.
`GITHUB_OUTPUT`	The path on the runner to the file that sets the current step's outputs from workflow commands. This file is unique to the current step and changes for each step in a job. For example, `/home/runner/work/_temp/_runner_file_commands/set_output_a50ef383-b063-46d9-9157-57953fc9f3f0`. For more information, see "Workflow commands for GitHub Actions."
`GITHUB_PATH`	The path on the runner to the file that sets system `PATH` variables from workflow commands. This file is unique to the current step and changes for each step in a job. For example, `/home/runner/work/_temp/_runner_file_commands/`

	`add_path_899b9445-ad4a-400c-aa89-249f18632cf5`. For more information, see "Workflow commands for GitHub Actions."
`GITHUB_REF`	The fully-formed ref of the branch or tag that triggered the workflow run. For workflows triggered by `push`, this is the branch or tag ref that was pushed. For workflows triggered by `pull_request`, this is the pull request merge branch. For workflows triggered by `release`, this is the release tag created. For other triggers, this is the branch or tag ref that triggered the workflow run. This is only set if a branch or tag is available for the event type. The ref given is fully-formed, meaning that for branches the format is `refs/heads/<branch_name>`, for pull requests it is `refs/pull/<pr_number>/merge`, and for tags it is `refs/tags/<tag_name>`. For example, `refs/heads/feature-branch-1`.
`GITHUB_REF_NAME`	The short ref name of the branch or tag that triggered the workflow run. This value matches the branch or tag name shown on GitHub. For example, `feature-branch-1`.
`GITHUB_REF_PROTECTED`	`true` if branch protections or rulesets are configured for the ref that triggered the workflow run.
`GITHUB_REF_TYPE`	The type of ref that triggered the workflow run. Valid values are `branch` or `tag`.
`GITHUB_REPOSITORY`	The owner and repository name. For example, `octocat/Hello-World`.
`GITHUB_REPOSITORY_ID`	The ID of the repository. For example, `123456789`. Note that this is different from the repository name.
`GITHUB_REPOSITORY_OWNER`	The repository owner's name. For example, `octocat`.
`GITHUB_REPOSITORY_OWNER_ID`	The repository owner's account ID. For example, `1234567`. Note that this is different from the owner's name.
`GITHUB_RETENTION_DAYS`	The number of days that workflow run logs and artifacts are kept. For example, `90`.
`GITHUB_RUN_ATTEMPT`	A unique number for each attempt of a particular workflow run in a repository. This number begins at 1 for the workflow run's first attempt, and increments with each re-run. For example, `3`.
`GITHUB_RUN_ID`	A unique number for each workflow run within a repository. This number does not change if you re-run the workflow run. For example, `1658821493`.
`GITHUB_RUN_NUMBER`	A unique number for each run of a particular workflow in a repository. This number begins at 1 for the workflow's first run, and increments with each new run. This number does not

	change if you re-run the workflow run. For example, `3`.
`GITHUB_SERVER_URL`	The URL of the GitHub server. For example: `https://github.com`.
`GITHUB_SHA`	The commit SHA that triggered the workflow. The value of this commit SHA depends on the event that triggered the workflow. For more information, see "Events that trigger workflows." For example, `ffac537e6cbbf934b08745a378932722df287a53`.
`GITHUB_STEP_SUMMARY`	The path on the runner to the file that contains job summaries from workflow commands. This file is unique to the current step and changes for each step in a job. For example, `/home/runner/_layout/_work/_temp/_runner_file_commands/step_summary_1cb22d7f-5663-41a8-9ffc-13472605c76c`. For more information, see "Workflow commands for GitHub Actions."
`GITHUB_WORKFLOW`	The name of the workflow. For example, `My test workflow`. If the workflow file doesn't specify a `name`, the value of this variable is the full path of the workflow file in the repository.
`GITHUB_WORKFLOW_REF`	The ref path to the workflow. For example, `octocat/hello-world/.github/workflows/my-workflow.yml@refs/heads/my_branch`.
`GITHUB_WORKFLOW_SHA`	The commit SHA for the workflow file.
`GITHUB_WORKSPACE`	The default working directory on the runner for steps, and the default location of your repository when using the `checkout` action. For example, `/home/runner/work/my-repo-name/my-repo-name`.
`RUNNER_ARCH`	The architecture of the runner executing the job. Possible values are `X86`, `X64`, `ARM`, or `ARM64`.
`RUNNER_DEBUG`	This is set only if debug logging is enabled, and always has the value of `1`. It can be useful as an indicator to enable additional debugging or verbose logging in your own job steps.
`RUNNER_NAME`	The name of the runner executing the job. This name may not be unique in a workflow run as runners at the repository and organization levels could use the same name. For example, `Hosted Agent`
`RUNNER_OS`	The operating system of the runner executing the job. Possible values are `Linux`, `Windows`, or `macOS`. For example, `Windows`
`RUNNER_TEMP`	The path to a temporary directory on the runner. This directory is emptied at the beginning and end of each job. Note that files will not be removed if the runner's user account does not have permission to delete them. For example, `D:\a\_temp`
`RUNNER_TOOL_CACHE`	The path to the directory containing preinstalled tools for GitHub-hosted runners. For more

information, see "Using GitHub-hosted runners".
For example, `C:\hostedtoolcache\windows`

Note: If you need to use a workflow run's URL from within a job, you can combine these variables: `$GITHUB_SERVER_URL/$GITHUB_REPOSITORY/actions/runs/$GITHUB_RUN_ID`

Detecting the operating system

You can write a single workflow file that can be used for different operating systems by using the `RUNNER_OS` default environment variable and the corresponding context property `${{ runner.os }}`. For example, the following workflow could be run successfully if you changed the operating system from `macos-latest` to `windows-latest` without having to alter the syntax of the environment variables, which differs depending on the shell being used by the runner.

YAML

```yaml
jobs:
  if-Windows-else:
    runs-on: macos-latest
    steps:
      - name: condition 1
        if: runner.os == 'Windows'
        run: echo "The operating system on the runner is $env:RUNNER_OS."
      - name: condition 2
        if: runner.os != 'Windows'
        run: echo "The operating system on the runner is not Windows, it's
$RUNNER_OS."
```

In this example, the two `if` statements check the `os` property of the `runner` context to determine the operating system of the runner. `if` conditionals are processed by GitHub Actions, and only steps where the check resolves as `true` are sent to the runner. Here one of the checks will always be `true` and the other `false`, so only one of these steps is sent to the runner. Once the job is sent to the runner, the step is executed and the environment variable in the `echo` command is interpolated using the appropriate syntax (`$env:NAME` for PowerShell on Windows, and `$NAME` for bash and sh on Linux and MacOS). In this example, the statement `runs-on: macos-latest` means that the second step will be run.

Passing values between steps and jobs in a workflow

If you generate a value in one step of a job, you can use the value in subsequent steps of the same job by assigning the value to an existing or new environment variable and then writing this to the `GITHUB_ENV` environment file. The environment file can be used directly by an action, or from a shell command in the workflow file by using the `run` keyword. For more information, see "Workflow commands for GitHub Actions."

If you want to pass a value from a step in one job in a workflow to a step in another job in the workflow, you can define the value as a job output. You can then reference this job output from a step in another job. For more information, see "Workflow syntax for GitHub Actions."