Using jobs in a workflow

In this article

Use workflows to run multiple jobs.

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.

Overview

A workflow run is made up of one or more `jobs`, which run in parallel by default. To run jobs sequentially, you can define dependencies on other jobs using the `jobs.<job_id>.needs` keyword.

Each job runs in a runner environment specified by `runs-on`.

You can run an unlimited number of jobs as long as you are within the workflow usage limits. For more information, see "Usage limits, billing, and administration" for GitHub-hosted runners and "About self-hosted runners" for self-hosted runner usage limits.

If you need to find the unique identifier of a job running in a workflow run, you can use the GitHub Enterprise Server API. For more information, see "Actions."

Setting an ID for a job

Use `jobs.<job_id>` to give your job a unique identifier. The key `job_id` is a string and its value is a map of the job's configuration data. You must replace `<job_id>` with a string that is unique to the `jobs` object. The `<job_id>` must start with a letter or `_` and contain only alphanumeric characters, `-`, or `_`.

Example: Creating jobs

In this example, two jobs have been created, and their `job_id` values are `my_first_job` and `my_second_job`.

```yaml
jobs:
  my_first_job:
    name: My first job
  my_second_job:
    name: My second job
```

Setting a name for a job

Use `jobs.<job_id>.name` to set a name for the job, which is displayed in the GitHub UI.

Defining prerequisite jobs

Use `jobs.<job_id>.needs` to identify any jobs that must complete successfully before this job will run. It can be a string or array of strings. If a job fails or is skipped, all jobs that need it are skipped unless the jobs use a conditional expression that causes the job to continue. If a run contains a series of jobs that need each other, a failure or skip applies to all jobs in the dependency chain from the point of failure or skip onwards. If you would like a job to run even if a job it is dependent on did not succeed, use the `always()` conditional expression in `jobs.<job_id>.if`.

Example: Requiring successful dependent jobs

```yaml
jobs:
  job1:
  job2:
    needs: job1
  job3:
    needs: [job1, job2]
```

In this example, `job1` must complete successfully before `job2` begins, and `job3` waits for both `job1` and `job2` to complete.

The jobs in this example run sequentially:

1. `job1`
2. `job2`
3. `job3`

Example: Not requiring successful dependent jobs

```yaml
jobs:
  job1:
  job2:
    needs: job1
  job3:
    if: ${{ always() }}
    needs: [job1, job2]
```

In this example, `job3` uses the `always()` conditional expression so that it always runs after `job1` and `job2` have completed, regardless of whether they were successful. For more information, see "Expressions."