

# YAML Terms for GitHub Actions

---

## Workflow Syntax

Work flow triggers are events that cause a workflow to run:

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

`name` : The name of the GitHub Action

`on` : When or condition of when the workflow is triggered. There are different filters, branches, labels, that can be used to trigger the workflow.

`jobs` : A jobs runs in a runner environment as determined by `runs-on`

## Course Notes

---

### Workflow Components

**Actions:** Reusable tasks that perform specific jobs within a workflow

**Workflows:** Automated processes defined in your repository that coordinate one or more jobs, triggered by events or on a schedule

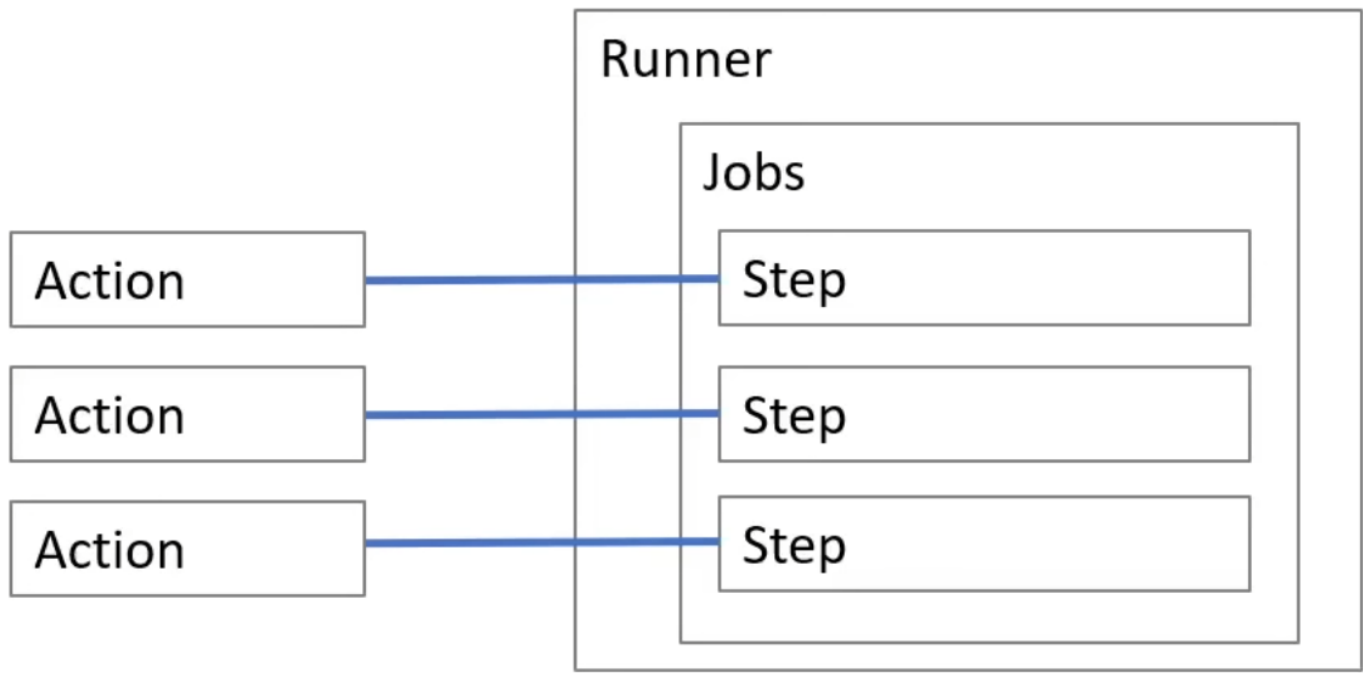
**Jobs:** Groups of steps that execute on the same runner, typically running in parallel unless configured otherwise

**Steps:** Individual tasks within a job that run commands or actions sequentially

**Runs:** Instances of workflow execution triggered by events, representing the complete run-through of a workflow

**Runners:** Servers that host the environment where the jobs are executed, available as GitHub-hosted or self-hosted options

**Marketplace:** A platform to find and share reusable actions



## Scheduling Events

**Schedule** can use a **cron expression** to trigger a workflow at a specific time or day

For example:

```
on:
  schedule:
    - cron: '30 5 * * 1,3'
    - cron: '30 5 * * 2,4'

jobs:
  test_schedule:
    runs-on: ubuntu-latest
    steps:
      - name: Not on Monday or Wednesday
        if: github.event.schedule != '30 5 * * 1,3'
        run: echo "Skip this step on Monday and Wednesday"
      - name: Every time
        run: echo "This step will always run"
```

## Triggering Single or Multiple Events

### Single Event

For example on **Push**

```

name: CI on Push

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Run a one-line script
        run: echo "Hello world!"

```

## Multiple Events

For example on **Push, Pull, Request, Release**

Note: 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

```

name: CI on Multiple Events

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main
  release:
    types: [published, created]

jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python

```

## Manual Events

Workflows can be triggered manually through Github UI, GitHub CLI, or GitHub REST API

```
gh workflow run greet.yml \  
-f name=brian \  
-f greeting=hello \  
-F data=@myfile.txt
```

a **workflow\_dispatch:** and **inputs:** event must be added to run manually

## Webhook Events

Many of the listed GitHub Workflow Triggers are triggered by a **webhook**

Most of these webhooks will be triggered within GitHub when users are interacting with GitHub which will in turn will trigger API actions. Users generally don't have to directly call the API to trigger the workflow

Using **repository\_dispatch** with **webhook** type you can trigger the Workflow via an external HTTP endpoint.

- will only trigger a workflow run if the workflow file is on the default branch

For example:

```
name: Workflow on Repository Dispatch  
  
on:  
  repository_dispatch:  
    types:  
      - webhook  
  
jobs:  
  respond-to-dispatch:  
    runs-on: ubuntu-latest  
    steps:  
      - name: Checkout repository  
        uses: actions/checkout@v2  
      - name: Run a script  
        run: echo "Event of type ${GITHUB_EVENT_NAME}"
```

When you **make the request to the webhook** you must:

- Send a POST request to the repo's dispatches endpoint
- Set the Accept type for application/vnd.github+json
- Provide Authorization to your Personal Access Token
- Pass the event type "webhook"

## Conditional Keyword for Steps

jobs.<job\_id>.if conditional can be used to **prevent a job from running unless a condition is met.**

For Example:

```
on:
  push:
    branches:
      - main

jobs:
  production-deploy:
    if: github.repository == 'brianopao/Github-Actions-Course'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '14'
      - run: npm install -g bats
```

**Note:** You must always use the `${{ }}` expression syntax or escape with `"`, `'`, or `()` when the expression starts with a `!`, since `!` is a special character in YAML

For Example:

```
if: ${{ ! startsWith(github.ref, 'refs/tags/v') }}
```

## Expressions

### Functions

- **contains** : Check if a string contains another string
  - example: `if: contains(github.event.head_commit.message, 'skip ci')`
- **endsWith** : Check if a string ends with another string
  - example: `if: endsWith(github.event.head_commit.message, 'skip ci')`
- **startsWith** : Check if a string starts with another string
  - example: `if: startsWith(github.event.head_commit.message, 'skip ci')`
- **format** : Format a string using a format string
  - example: `if: format('refs/heads/{0}', github.ref) == 'refs/heads/main'`
- **join** : Join a list of strings into a single string
  - example: `if: join(github.event.labels.*.name, ',') == 'bug,enhancement'`
- **toJson** : Convert a value to a JSON string
  - example: `if: toJson(github.event) == '{"key": "value"}'`
- **fromJson** : Convert a JSON string to a value
  - example: `if: fromJson('{"key": "value"}').key == 'value'`

- `hashFiles` : Compute the hash of files
- ◦ example: `if: hashFiles('**/*.js') == '123456'`

## Status Check Functions

- `always` : Always true
- `cancelled` : Always false
- `failure` : True if the previous step failed
- `success` : True if the previous step succeeded
- `skipped` : True if the previous step was skipped
- `timedout` : True if the previous step timed out
- `neutral` : True if the previous step was

## Runners

The Runner **determines the underlying computer and OS** that the workflow will run on

The runner can be:

- GitHub-hosted runner - GitHub provides runtime environments
  - ◦ Standard Size
    - ▪ Linux, ubuntu, Windows
    - ▪ MacOS
  - ◦ Larger Size
    - ▪ Only available for organizations and enterprise using the GitHub Team or Github Enterprise Cloud plans
    - ▪ More RAM, CPU, and storage
- Self-hosted - External compute connected to GitHub using the GitHub Actions self-hosted runner application

```
runs-on: ubuntu-latest
runs-on: windows-latest
runs-on: macos-latest

runs-on: [macos-14, macos-13, macos-12]

runs-on: self-hosted
```

## GitHub-Hosted Runners

- No setup required; fully managed by GitHub
- Free with limits on usage; ch
- Automatically scales based on demand

- Predefined environments with limited control
- Windows, Linux, MacOS
- Secure but runs in a shared environment
- Fixed performance capabilities

## Self-Hosted Runners

- Requires setup and maintenance
- No additional cost; you provide the hardware
- Full control over the environment
- Any OS, any hardware
- Potentially more secure, but you are responsible for security
- Performance depends on your hardware

## Self Hosted Runners Setup

They can be:

- Physical machines
- Virtual machines
- Containers
- On-premises
- Cloud

You can add self-hosted runner at various levels in the management hierarchy:

- Repository Level
  - dedicated to a single repository
- Organization Level
  - shared across multiple repositories
- Enterprise Level
  - shared across multiple organizations

To setup self-hosted you need to add a runner and install the GitHub Actions Runner to connect the external compute to the external self-hosted runner

## Workflow Commands

Actions can communicate with the runner machine to set environment variables, output values used by other actions, add debug messages to the log, and other tasks

**Set Env Vars** set an environment variables that will be available to subsequent actions in the job

```
steps:
  - name: Set an environment variable
    run: echo "::ACTION_ENV=production" >> $GITHUB_ENV
```

**Adding to System Path** Add a directory to the system PATH for subsequent steps in the job

```
steps:
  - name: Add directory to PATH
    run: echo "/pth/to/dir" >> $GITHUB_PATH
```

**Setting Output Variables** Set outputs that can be used by other jobs in a multi-job workflow

```
steps:
  - name: Set output
    id: exmaple-step
    run: echo "result=output_value" >> $GITHUB_OUTPUT

  - name: Use Output
    run: echo "The output was ${ steps.example-step.outputs.result }"
```

**Debug Messages** Add debug messages to the log

```
steps:
  - name: Debug message
    run: echo "::debug::This is a debug message"
```

**Grouping Log Messages** Makes logs easier to read by grouping messages together

```
steps:
  - name: Group messages
    run: echo "::group::My Group"
  - name: Step 1
    run: echo "This is step 1"
  - name: Step 2
    run: echo "This is step 2"
  - name: End Group
    run: echo "::endgroup::"
```

**Masking Values in Logs** Prevent sensitive information from being printed in logs

```
steps:
  - name: Mask a value
    run: echo "::add-mask::${ secrets.SECRET_VALUE }"
```



**Stopping Failing Actions** Can force a workflow to stop and fail using the error command

```
steps:  
  - name: Fail the workflow  
    run: echo "::error::This workflow has failed"
```