



The background features abstract, organic shapes in shades of pink and orange. Concentric, thin gold lines radiate from the center, creating a focal point for the text.

# GitHub Actions

# "Hello, world."

在读专业的书本之前，不妨先做个小实验。

1 在自己的 GitHub 上创建一个 repo，并克隆到本地。

The screenshot shows a GitHub repository page for 'GithubActions'. The repository has one branch ('main') and one commit ('Initial commit' by 'AnhaoROMA'). The README file contains the text 'GithubActions'. On the right side, there are sections for 'About', 'Readme', 'Activity', 'Releases', and 'Packages'. The bottom of the page includes standard GitHub footer links.

The screenshot shows the GitHub Actions interface. A workflow named 'any.yml' is selected in the 'GITHUB ACTIONS' sidebar. The workflow has one run, indicated by a red dot. The main pane displays the workflow configuration and its status. To the right, a terminal window shows the command-line steps taken to set up the repository for GitHub Actions:

```
PS E:\GithubActions> git add .github/*
On branch main
Your branch is up to date with 'origin/main'.
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
  new file: .github/workflows/any.yml

PS E:\GithubActions> git commit -m "启用 Github Actions 功能"
[main 6636525] 启用 Github Actions 功能
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 .github/workflows/any.yml
PS E:\GithubActions> git push origin main
Enumerating objects: 6, done.
```

2 在本地的工作路径下创建文件 `.github/workflows/*.yml`。

3 提交并推送。

4 此时在“Actions”栏发现报错。

The screenshot shows the 'Actions' tab in the GitHub repository. A circled 'Actions' tab is highlighted. The main pane shows a single workflow run for 'any.yml'. An annotation error is visible in the bottom left corner:

Annotations  
1 error

✖ Error  
No event triggers defined in `on`

5 根据报错信息，可知：需定义“何时(事)触发工作流”。

```

any.yml 1 ×
.github > workflows > any.yml
1 on: push
2

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS E:\GithubActions> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   .github/workflows/any.yml

no changes added to commit (use "git add" and/or "git commit -a")
PS E:\GithubActions> git add .github/workflows/any.yml
PS E:\GithubActions> git commit -m "添加 on 对应的事件"
[main 0eda148] 添加 on 对应的事件
• 1 file changed, 1 insertion(+)
PS E:\GithubActions> git push origin main
• Enumerating objects: 9, done.

```

6 定义“何时(事)触发工作流”：  
当有内容 push 到 GitHub 仓库时，  
触发工作流。

## 7 依旧报错。

All workflows

2 workflow runs

Event	Status	Branch	Actor
now	Failure	main	
6 minutes ago	Failure	main	

**Annotations**  
1 error

**Error**  
No jobs defined in `jobs`

8 报错，除了“何时(事)触发工作流”，还需要明确“干什么事情”。

```

.github > workflows > any.yml
1 on: push
2
3 jobs:
4   job1:
5     ...
6   job2:
7     ...

```

9 暂时仅保留位。

All workflows

3 workflow runs

Event	Status	Branch	Actor
1 minute ago	Failure	main	
5 minutes ago	Failure	main	
11 minutes ago	Failure	main	

10 不出意外地收到报错。

↳ .github/workflows/any.yml

### ✖ 添加 jobs #3

The screenshot shows a GitHub Actions workflow run summary. It indicates the workflow was triggered via push 1 minute ago by AnhaoROMA, pushed to branch main, and failed. The total duration is listed as '-' and there are no artifacts. A message states: "This workflow graph cannot be shown. A graph will be generated the next time this workflow is run." Below this, the "Annotations" section shows 1 error: "Invalid workflow file: .github/workflows/any.yml#L1. No steps defined in 'steps' and no workflow called in 'uses' for the following jobs: job1, job2".

11 需在 jobs 中编写步骤.

12

pwd, ls  
echo "hello, CI/CD"

```
.github > workflows > any.yml
1   on: push
2
3   jobs:
4     job1:
5       steps:
6         - run: pwd
7         - run: ls
8     job2:
9       steps:
10      - run: echo "hello, CI/CD"
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

- PS E:\GithubActions> git commit -a -m "添加 job1 和 job2 的步骤 (steps)" [main 970baef] 添加 job1 和 job2 的步骤 (steps)  
1 file changed, 6 insertions(+), 2 deletions(-)
- PS E:\GithubActions> git push origin main

13 还是不行.

The screenshot shows the GitHub Actions Actions page. The left sidebar includes sections for All workflows, Management (Caches, Attestations, Runners, Usage metrics, Performance metrics), and Annotations (1 error). The main area displays "All workflows" showing runs from all workflows. It lists four workflow runs:

- 添加 job1 和 job2 的步骤 (steps) .github/workflows/any.yml #4: Commit 970baef pushed by AnhaoROMA (Status: Success, now, 1s)
- 添加 jobs .github/workflows/any.yml #3: Commit a4d3dd9 pushed by AnhaoROMA (Status: Failure, 6 minutes ago)
- 添加 on 对应的事件 .github/workflows/any.yml #2: Commit 0eda148 pushed by AnhaoROMA (Status: Failure, 10 minutes ago)
- 启用 Github Actions 功能 .github/workflows/any.yml #1: Commit 6636525 pushed by AnhaoROMA (Status: Failure, 16 minutes ago)

The "Annotations" section at the bottom shows 1 error: "Invalid workflow file: .github/workflows/any.yml#L1. The workflow is not valid. .github/workflows/any.yml (Line: 5, Col: 9): Required property is missing: runs-on .github/workflows/any.yml (Line: 9, Col: 9). Required property is missing: runs-on".

## 14 至少得告诉 GitHub 选择什么操作系统嘛！

```
.github > workflows > any.yml
1  on: push
2
3  jobs:
4    job1:
5      runs-on: ubuntu-latest
6      steps:
7        - run: pwd
8        - run: ls
9    job2:
10   runs-on: ubuntu-latest
11   steps:
12     - run: echo "hello, CI/CD"
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS E:\GithubActions> git commit -a -m "指定运行环境 (runs-on)"  
[main d391c6a] 指定运行环境 (runs-on)  
● 1 file changed, 2 insertions(+)  
PS E:\GithubActions> git push origin main  
● Enumerating objects: 9, done.

选择最新的 Ubuntu。

The screenshot shows the GitHub Actions interface. On the left, there's a sidebar with 'Actions' selected. Under 'All workflows', it lists '.github/workflows/any.yml'. The main area shows 'All workflows' with the heading 'All workflow runs' and 'Showing runs from all workflows'. There are five workflow runs listed, each with a status (e.g., main, 1 minute ago, 5 minutes ago, etc.) and a '...' button. The first run is circled in blue and has a green checkmark next to the text '指定运行环境 (runs-on)'. The other four runs have red circles with white numbers indicating steps: 2, 3, 4, and 5 respectively. A search bar at the top right says 'Type ⌘ to search'.

## 15 终于！

## 16 点开看看细节。

This screenshot shows the detailed view of a GitHub Action run for the 'any.yml' workflow. At the top, it says 'Triggered via push 3 minutes ago' and shows 'Status Success' with a duration of '12s'. Below this is a summary table for 'any.yml' triggered by 'on: push'. It shows two jobs: 'job1' and 'job2', both of which completed successfully with a duration of '0s'. In the 'Annotations' section, there are two warning messages: one for 'job2' stating 'ubuntu-latest pipelines will use ubuntu-24.04 soon.' and another for 'job1' with the same message. There are also 'Re-run all jobs' and '...' buttons at the top right.

类似于 Jenkins 界面

```

job1
succeeded 1 minute ago in 0s

Set up job
1 Current runner version: '2.321.0'
2 ▶ Operating System
3   Ubuntu
4     24.04.1
5   LTS
6 ▶ Runner Image
7   Image: ubuntu-24.04
8   Version: 20241215.1
9   Included Software: https://github.com/actions/runner-images/blob/ubuntu24/20241215.1/images/ubuntu/Ubuntu2404-Readme.md
10  Image Release: https://github.com/actions/runner-images/releases/tag/ubuntu24%2F20241215.1
11 ▶ Runner Image Provisioner
12   2.0.404.1
13 ▶ GITHUB_TOKEN Permissions
14   Contents: read
15   Metadata: read
16   Packages: read
17   Secret source: Actions
18   Prepare workflow directory
19   Prepare all required actions
20   Complete job name: job1

Run pwd
1   ▶ Run pwd
2     /home/runner/work/GithubActions/GithubActions

Run ls
1   ▶ Run ls
2     ls
3     shell: /usr/bin/bash -e {0}

Complete job
1   Cleaning up orphan processes

```

```

job1
succeeded 3 minutes ago in 0s

Set up job
1 Current runner version: '2.321.0'
2 ▶ Operating System
3   Ubuntu
4     24.04.1
5   LTS
6 ▶ Runner Image
7   Image: ubuntu-24.04
8   Version: 20241215.1
9   Included Software: https://github.com/actions/runner-images/blob/ubuntu24/20241215.1/images/ubuntu/Ubuntu2404-Readme.md
10  Image Release: https://github.com/actions/runner-images/releases/tag/ubuntu24%2F20241215.1
11 ▶ Runner Image Provisioner
12   2.0.404.1
13 ▶ GITHUB_TOKEN Permissions
14   Contents: read
15   Metadata: read
16   Packages: read
17   Secret source: Actions
18   Prepare workflow directory
19   Prepare all required actions
20   Complete job name: job1

```

由此可查看已预装软件目录

17 在这个项目中写一个 Go 程序并生成对应的可执行文件。

```

EXPLORER
...
any.yml
GO main.go ×
any.yml
GO main.go > ...
2
3 import "fmt"
4
5 func main() {
6   fmt.Println("西山苍苍 东海茫茫")
7 }

```

GITHUB ACTIONS

- .github\workflows
- any.yml
- main.exe**
- main.go
- README.md

因为 .exe 文件是 Windows 系统的可执行文件后缀名，

所以 runs-on: windows-latest。

```

any.yml  X  main.go
.github > workflows > any.yml
1 on: push
2
3 jobs:
4   Tsinghua:
5     runs-on: windows-latest
6     steps:
7       - run: dir
8       - run: ./main.exe

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

no changes added to commit (use "git add" and/or "git commit -a")
PS E:\GithubActions> git add .github/workflows/any.yml
PS E:\GithubActions> git commit -m "放到 Windows 操作系统里面测试"
[main bce3d04] 放到 Windows 操作系统里面测试
1 file changed, 4 insertions(+), 9 deletions(-)
PS E:\GithubActions> git push origin main

```

## 18 看样子没什么问题，然而...

The screenshot shows the GitHub Actions UI for a workflow named 'any.yml'. A single job named 'Tsinghua' has failed. The status bar indicates 'Failure' and a total duration of '16s'. The annotations section shows a single error from the 'Tsinghua' step: 'Process completed with exit code 1.'

## 19 点开看看。

This screenshot provides a detailed view of the 'Run ./main.exe' step in the 'Tsinghua' job. The logs show the command being run and the resulting error message: 'The term './main.exe' is not recognized as a name of a cmdlet, function, script file, or executable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again.' This indicates that the file 'main.exe' was not found in the current directory.

原因：当前路径下没有 main.exe 文件。

解决方法也很显然：把 GitHub 仓库中的文件拷贝到操作系统里面。

## 20 使用一个现成的“代码”。

```
.github > workflows > any.yml
1   on: push
2
3   jobs:
4     Tsinghua:
5       runs-on: windows-latest
6       steps:
7         - uses: actions/checkout@v4
8         - run: dir
9         - run: ./main.exe
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\GithubActions> git commit -a -m "使用现成的 checkout action 仓库"
[main 81eca7d] 使用现成的 checkout action 仓库
 1 file changed, 1 insertion(+)
PS E:\GithubActions> git push origin main
```

## 21 成功了。

← .github/workflows/any.yml  
✓ 使用现成的 checkout action 仓库 #8

Re-run all jobs ...

Summary

Jobs

Tsinghua

Run details

Usage

Workflow file

Triggered via push 1 minute ago

AnhaoROMA pushed → 81eca7d main

Status Success

Total duration 24s

Artifacts —

any.yml

on: push

Tsinghua 11s

Re-run all jobs ...

← .github/workflows/any.yml  
✓ 使用现成的 checkout action 仓库 #8

Re-run all jobs ...

Summary

Jobs

Tsinghua

Run details

Usage

Workflow file

Tsinghua

succeeded 2 minutes ago in 11s

Search logs

Set up job

Run actions/checkout@v4

Run dir

Run ./main.exe

Run ./main.exe

西山苍苍 东海茫茫

Post Run actions/checkout@v4

Complete job

github > workflows > any.yml

```
1 on: push
2
3 jobs:
4   Tsinghua:
5     name: 清华大学
6     runs-on: windows-latest
7
8     steps:
9       - name: 拷贝内容
10      uses: actions/checkout@v4
11
12      - name: 运行程序
13        run: ./main.exe
```

## 22 感觉有点乱...

不妨给每个步骤起上名字吧。

job 的名字

step 的名字

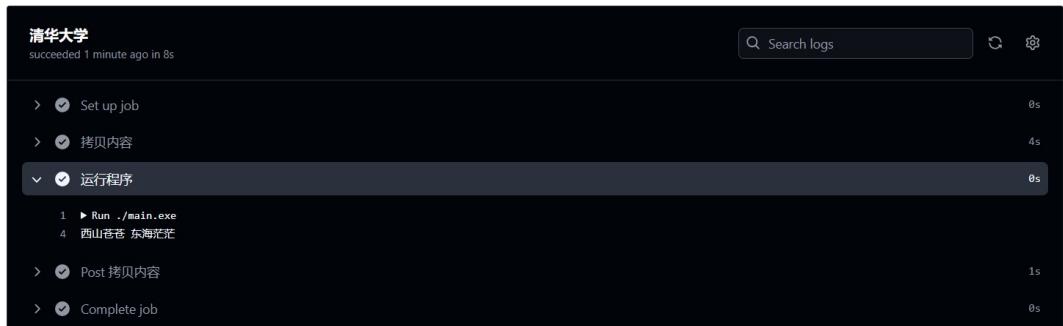
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\GithubActions> git commit -a -m "给每个步骤取名字"
[main cc59be] 给每个步骤取名字
 1 file changed, 9 insertions(+), 3 deletions(-)
PS E:\GithubActions> git push origin main
```

↳ .github/workflows/any.yml  
✓ 给每个步骤取名字 #9

Re-run all jobs ...

Summary  
Jobs  
清华大学  
Run details  
Usage  
Workflow file



这样就好看多了。

## 23 使用 | 运算符可以将多条步骤组合在一起。

.github > workflows > any.yml

```
1   on: push
2
3   jobs:
4     Tsinghua:
5       name: 清华大学
6
7       runs-on: windows-latest
8
9       steps:
10      - name: 复制内容
11        uses: actions/checkout@v4
12
13      - name: 执行两次程序
14        run: |
15          ./main.exe
16          ./main.exe
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS E:\GithubActions> git commit -a -m "多个步骤合到一起"



## 24 还可以给整个工作流起名字。

↳ hello-word

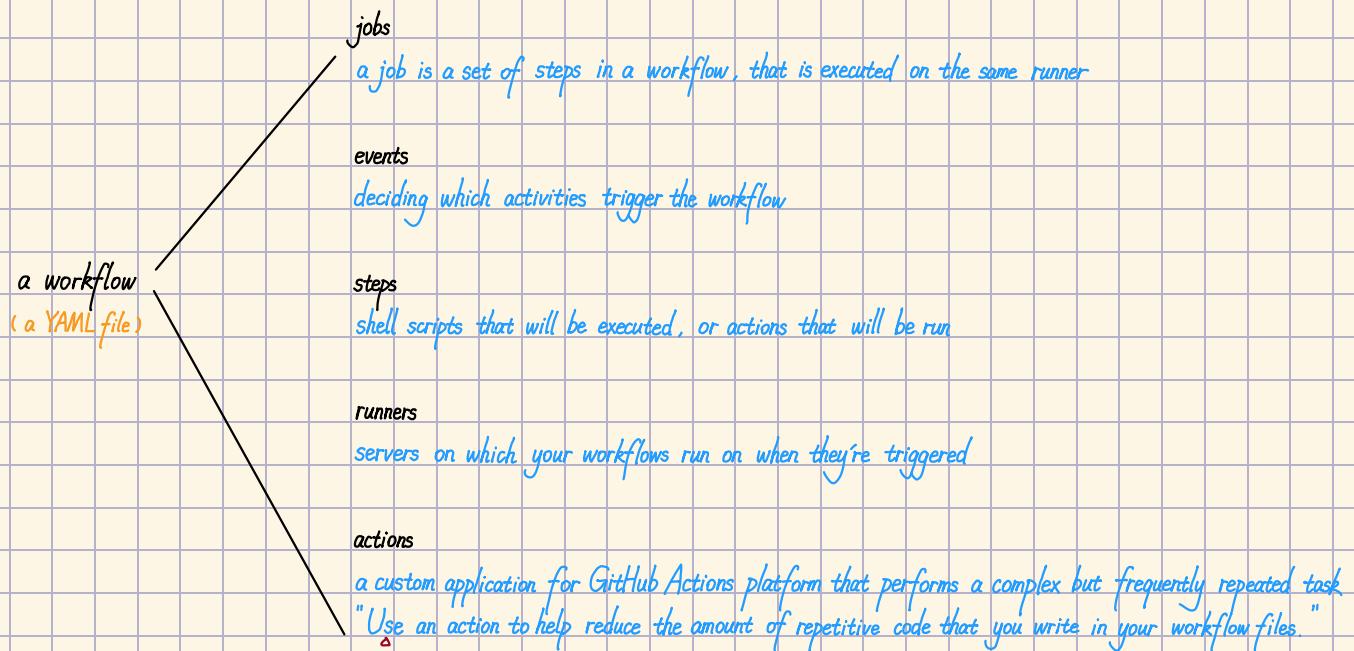
✓ 给整个工作流 (workflow) 起名字 #11

# Getting Started with Workflows

GitHub Actions' workflow features allow users to configure various deployment and delivery pipelines to support different technologies.

This chapter helps me understand GitHub Actions workflows by discussing the components that are important for configuring build and deployment pipelines.

GitHub Actions workflows are configured using preconfigured workflow templates or marketplace actions or self-defined actions.



## # Using Preconfigured Workflow Templates

GitHub has multiple predefined workflow templates to create automated build and deployment processes.

Deployment View all

Deploy Node.js to Azure Web App By Microsoft Azure

Deploy to Amazon ECS By Amazon Web Services

Build and Deploy to GKE By Google Cloud

Terraform By HashiCorp

Deploy to Alibaba Cloud ACK By Alibaba Cloud

Deploy to IBM Cloud Kubernetes Service By IBM

Tencent Kubernetes Engine By Tencent Cloud

OpenShift By Red Hat

There are deployment workflow templates for all the main cloud platforms.

After clicking buttons, a template workflow YAML file is opened, which I can edit to fit my requirements.

[GithubActions/.github/workflows/azure-webapps-node.yml](#) in [main](#)

[Edit](#) [Preview](#) [Code 55% faster with GitHub Copilot](#)

```

1 # This workflow will build and push a node.js application to an Azure Web App when a commit is pushed to your default branch.
2 #
3 # This workflow assumes you have already created the target Azure App Service web app.
4 # For instructions see https://docs.microsoft.com/en-us/azure/app-service/quickstart-nodejs?tabs=linux&pivots=development-environment-cli
5 #
6 # To configure this workflow:
7 #
8 # 1. Download the Publish Profile for your Azure Web App. You can download this file from the Overview page of your Web App in the Azure Portal.
9 #   For more information: https://docs.microsoft.com/en-us/azure/app-service/deploy-github-actions?tabs=applevel#generate-deployment-credentials
10 #
11 # 2. Create a secret in your repository named AZURE_WEBAPP_PUBLISH_PROFILE, paste the publish profile contents as the value of the secret.
12 #   For instructions on obtaining the publish profile see: https://docs.microsoft.com/azure/app-service/deploy-github-actions#configure-the-github-action
13 #
14 # 3. Change the value for the AZURE_WEBAPP_NAME. Optionally, change the AZURE_WEBAPP_PACKAGE_PATH and NODE_VERSION environment variables below.
15 #
16 # For more information on GitHub Actions for Azure: https://github.com/Azure/Actions
17 # For more information on the Azure Web Apps Deploy action: https://github.com/Azure/webapps-deploy
18 # For more samples to get started with GitHub Action workflows to deploy to Azure: https://github.com/Azure/actions-workflow-samples
19 #
20 on:
21   push:
22     branches: [ "main" ]
23   workflow_dispatch:
24   ...
25 ...

```

Use [Control + Shift + m](#) to toggle the [tab](#) key moving focus. Alternatively, use [esc](#) then [tab](#) to move to the next interactive element on the page.

Use [Control + Space](#) to trigger autocomplete in most situations.

[Cancel changes](#) [Commit changes...](#)

### Marketplace Documentation

Search Marketplace for Actions

#### Featured Actions

- [Setup Node.js environment](#) 4k  
By actions  
Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH
- [Upload a Build Artifact](#) 3.3k  
By actions  
Upload a build artifact that can be used by subsequent workflow steps
- [Setup Java JDK](#) 1.6k  
By actions  
Set up a specific version of the Java JDK and add the command-line tools to the PATH
- [Setup .NET Core SDK](#) 958  
By actions  
Used to build and publish .NET source. Set up a specific version of the .NET and authentication to private NuGet repository

## Choose a workflow

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and [set up a workflow yourself](#) →

### Categories

Deployment  
Security  
**Continuous integration**  
Automation  
Pages

[Search workflows](#)

Found 54 workflows

<a href="#">SLSA Go releaser</a> By Open Source Security Foundation (OpenSSF) Compile your Go project using a SLSA compliant builder	<a href="#">Go</a> By GitHub Actions Build a Go project.	<a href="#">SLSA Generic generator</a> By Open Source Security Foundation (OpenSSF) Generate SLSA3 provenance for your existing release workflows
<a href="#">Node.js</a> By GitHub Actions Build and test a Node.js project with npm.	<a href="#">.NET Desktop</a> By GitHub Actions Build, test, sign and publish a desktop application built on .NET.	<a href="#">CMake based, single-platform projects</a> By GitHub Actions Build and test a CMake based project on a single-platform.
<a href="#">Publish Python Package</a> By GitHub Actions Publish a Python Package to PyPI on release.	<a href="#">Publish Node.js Package to GitHub Packages</a> By GitHub Actions Publishes a Node.js package to GitHub Packages.	<a href="#">Java with Maven</a> By GitHub Actions Build and test a Java project with Apache Maven.
<a href="#">Configure</a> <a href="#">JavaScript</a>	<a href="#">Configure</a> <a href="#">C#</a>	<a href="#">Configure</a> <a href="#">Java</a>

In addition to continuous deployment workflow templates, there are continuous integration workflow templates to build applications.

Like the deployment workflow templates, an integration workflow is also a YAML file consisting of basic build steps that I can edit according to my requirements.

[GithubActions/.github/workflows/go.yml](#) in [main](#)

[Edit](#) [Preview](#) [Code 55% faster with GitHub Copilot](#)

```

2 # For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-go
3
4 name: Go
5
6 on:
7   push:
8     branches: [ "main" ]
9   pull_request:
10    branches: [ "main" ]
11
12 jobs:
13
14   build:
15     runs-on: ubuntu-latest
16     steps:
17       - uses: actions/checkout@v4
18
19       - name: Set up Go
20         uses: actions/setup-go@v4
21         with:
22           go-version: '1.20'
23
24       - name: Build

```

Use [Control + Shift + m](#) to toggle the [tab](#) key moving focus. Alternatively, use [esc](#) then [tab](#) to move to the next interactive element on the page.

Use [Control + Space](#) to trigger autocomplete in most situations.

[Cancel changes](#) [Commit changes...](#)

### Marketplace Documentation

Search Marketplace for Actions

#### Featured Actions

- [Cache](#) 4.6k  
By actions  
Cache artifacts like dependencies and build outputs to improve workflow execution time
- [Download a Build Artifact](#) 1.5k  
By actions  
Download a build artifact that was previously uploaded in the workflow by the upload-artifact action
- [Setup Go environment](#) 1.4k  
By actions  
Setup a Go environment and add it to the PATH
- [Close Stale Issues](#) 1.4k  
By actions  
Close issues and pull requests with no recent activity
- [First interaction](#) 774  
By actions

## # Using Marketplace Actions

One place to access Marketplace actions is the workflow editor page.

The screenshot shows the GitHub Actions workflow editor for a file named go.yml. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The current tab is 'Actions'. The main area displays a YAML configuration for a Go workflow, including jobs for building and testing. On the right side, there is a sidebar titled 'Marketplace' which lists several featured actions: Cache, Download a Build Artifact, Setup Go environment, Close Stale Issues, and First interaction. A blue circle highlights the 'Marketplace' tab in the sidebar.

This screenshot shows the 'Marketplace' search results for actions. The search bar at the top contains the placeholder 'Search Marketplace for Actions'. Below it, a section titled 'Featured Actions' lists five items: Cache, Upload a Build Artifact, Download a Build Artifact, Setup Go environment, and Setup .NET Core SDK. Each item has a star icon indicating its popularity (e.g., 4.6k stars for Cache).

Then you can select the actions that need to be added to the workflow.

This screenshot shows the details page for the 'Download a Build Artifact' action. It includes a brief description, a star rating of 1.5k, and a link to 'View full Marketplace listing'. Below this, a section titled 'Installation' provides a snippet of YAML code to copy and paste into a workflow file. A handwritten note next to this section says: 'Then copy & paste the relevant scripts under the Installation section into my workflow.' A blue arrow points from the 'Featured Actions' list in the previous screenshot to this installation section.

## # Using Self-defined Actions

If you have a comprehensive understanding of all components of workflows, then you can write that yaml file from scratch!

## # Understanding the Structure of a Workflow

This screenshot shows the 'Choose a workflow template' page. It features a large button with the text 'Skip this and set up a workflow yourself' enclosed in a yellow box. A handwritten note next to this button says: 'to start with an empty workflow, without using templates'.

Then fill the yaml file as below. Explanation is attached as comments.

Code

Blame

32 lines (27 loc) · 1.02 KB



Code 55% faster with GitHub Copilot

```
1  # This is a basic workflow to help me get started with Github Actions.
2  name: CI
3
4  # To control when the action will run:
5  #   to trigger the workflow on push or pull request, but only the master branch.
6  on:
7    push:
8      branches: [ main ]
9    pull_request:
10      branches: [ main ]
11
12  # A workflow is made up of one or more jobs that can run sequentially or in parallel.
13  jobs:
14    # This workflow contains a single job called "build".
15    build:
16      # The runner (that this job will run on) runs on Ubuntu.
17      runs-on: ubuntu-latest
18
19      # Steps represent a sequence of tasks that will be executed, as part of the job.
20      steps:
21        # Checks-out your repository under $GITHUB_WORKSPACE, so your job(s) can access it.
22        - uses: actions/checkout@v4
23
24        # To run a single command using the runner's shell.
25        - name: to run a single command
26          run: echo ".- - . .... .- ---"
27
28        # To run a set of commands using the runner's shell.
29        - name: to run a set of commands
30          run: |
31            echo "amy"
32            echo "fran"
```

build

succeeded 1 minute ago in 3s

Search logs

> Set up job 1s

> Run actions/checkout@v4 0s

✓ to run a single command 0s

1 ► Run echo ".- - . .... .- ---" 1s

4 .. - - . .... .- ---

✓ to run a set of commands 0s

1 ► Run echo "amy" 0s

5 amy

6 fran

> Post Run actions/checkout@v4 0s

> Complete job 0s

This screenshot shows the GitHub Actions build log for the workflow. The build was successful and completed in 3 seconds. It details the execution of the 'build' job, which includes running a checkout step and two command steps. The first command step runs 'echo ".- - . .... .- ---"'. The second command step runs 'echo "amy"' followed by 'echo "fran"'. All steps are marked as completed successfully.

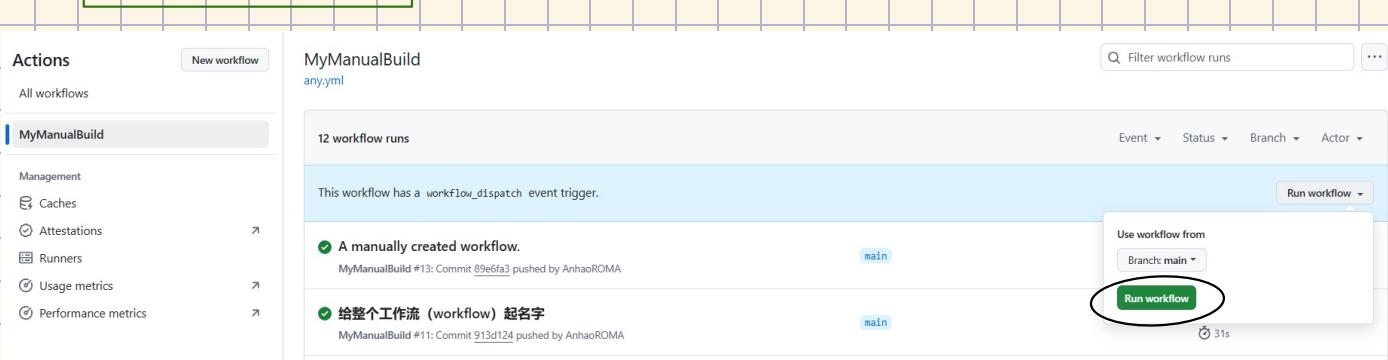
Workflow files should be saved under `.github/workflows/` in the repository root.

I can define the exact triggering condition for each workflow, for example, `event triggers`, `schedule triggers` and `manual triggers`.

For instance, to enable a manual trigger, a `workflow_dispatch` event should be activated in my workflow.

`name: MyManualBuild`  
`on: [ workflow_dispatch ]`

This enables the `Run workflow` button.



The screenshot shows the GitHub Actions interface. On the left, there's a sidebar with 'Actions', 'All workflows', and a section for 'MyManualBuild'. This section contains 'Management' options like 'Caches', 'Attestations', 'Runners', 'Usage metrics', and 'Performance metrics'. The main area shows 'MyManualBuild any.yml' with '12 workflow runs'. A callout highlights the 'Run workflow' button at the top right of the run list, which is circled in red. The button has a tooltip 'Run workflow from Branch: main'.

Another important component is the runner.

A runner is a machine or container that executes the workflow. There are 2 types of runners: `GitHub-hosted runners` and `self-hosted runners`.

A workflow can have one or more jobs.

By default, jobs run in parallel. Hence, if I need to run jobs one after another, dependency `needs` to be specified.

## # Experiment

Let me build a Go application in use of continuous integration.

I will use a Go workflow template, and modify the `YAML` file according to my requirements.

```
1 # This workflow will build a golang project
2 # For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-go
3
4 name: Go
5
6 on:
7   push:
8     branches: [ "main" ]
9   pull_request:
10    branches: [ "main" ]
11
12 jobs:
13
14   build:
15     runs-on: ubuntu-latest
16     steps:
17       - uses: actions/checkout@v4
18
19       - name: Set up Go
20         uses: actions/setup-go@v4
21         with:
22           go-version: '1.20'
23
24       - name: Build
25         run: go build -v ./...
26
27       - name: Test
28         run: go test -v ./...
```

Source code should be downloaded to the runner as the first step before building the code. Therefore, the `checkout` action downloads the source.

Go version is defined after `with`.

ls

1 ► Run ls  
4 README.md  
5 go.mod  
6 go.work  
7 main  
8 main.exe  
9 main.go

the new built

command

source

More following behaviors are "publishing files" and "uploading published files as an artifact to the build pipeline", if you are dealing with .Net Core code.

# Variables, Secrets, Tokens

In any platform or tool facilitating the implementation of CI/CD, it is essential to have a mechanism to configure **variables** in the pipelines. This chapter explores the options for GitHub Actions variables. In GitHub Actions, I can define custom variables in the scope of a **workflow**, **job**, or **step**.

## # Variables in Workflow Scope

.github > workflows > go.yml

```
1 name: GoProgram
2
3 on:
4   push:
5     branches: [ "main" ]
6
7 env:
8   fc_anhao: "Atletico de Madrid"
9   fc_eisen: "Inter Milan"
10  fc_furong: "Barcelona"
11
12 jobs:
13   build:
14     runs-on: ubuntu-latest
15     steps:
16       - name: Workflow Scope
17         run:
18           echo "Anhao's favourite football club is '$fc_anhao'"
19           echo "Eisen's favourite football club is '$fc_eisen'"
20           echo "Furong's favourite football club is '$fc_furong'"
```

To Define

To Use

← GoProgram

workflow scope #8

Summary

Jobs

build

Run details

Usage

Workflow file

Annotations

1 warning

build

succeeded 1 minute ago in 0s

Set up job

Workflow Scope

Run echo "Anhao's favourite football club is '\$fc\_anhao'"

echo "Anhao's favourite football club is 'Atletico de Madrid'"

echo "Eisen's favourite football club is '\$fc\_eisen'"

echo "Furong's favourite football club is '\$fc\_furong'"

shell: /usr/bin/bash -e {0}

env:

fc\_anhao: Atletico de Madrid

fc\_eisen: Inter Milan

fc\_furong: Barcelona

Anhao's favourite football club is 'Atletico de Madrid'

Eisen's favourite football club is 'Inter Milan'

Furong's favourite football club is 'Barcelona'

Complete job

## # Naming Considerations for Variables

### 1. GITHUB\_\* ✗

The "GITHUB\_" prefix is reserved for GitHub.

### 2. Case Sensitivity ✓

GitHub Actions variables are case-sensitive.

### 3. \*\_PATH !

The variables defined to point to a filesystem location should contain the "\_PATH" suffix.  
( Though, the `HOME` and `GITHUB_WORKSPACE` default variables do not use this convention. )

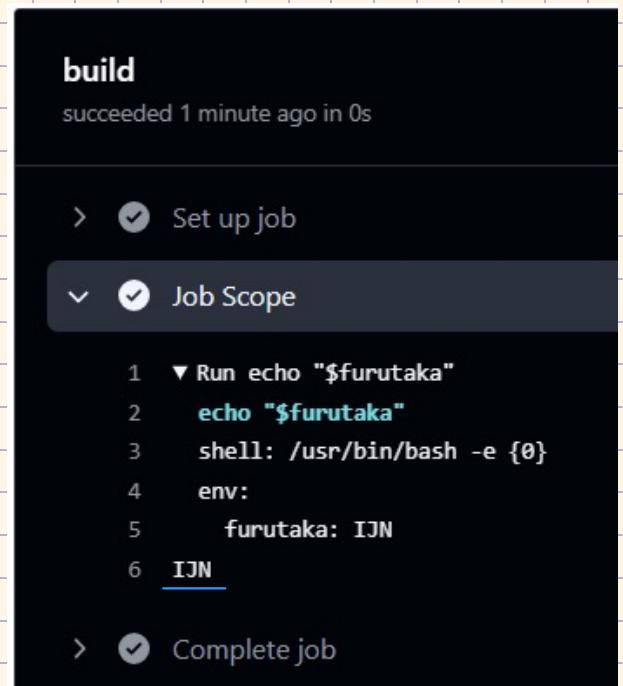
### 4. Special Characters !

Even though there are no syntactical errors caused by using special characters in the middle of a variable name, it is better to avoid them at all costs, other than `_`.

`[a-zA-Z_][a-zA-Z_0-9]`

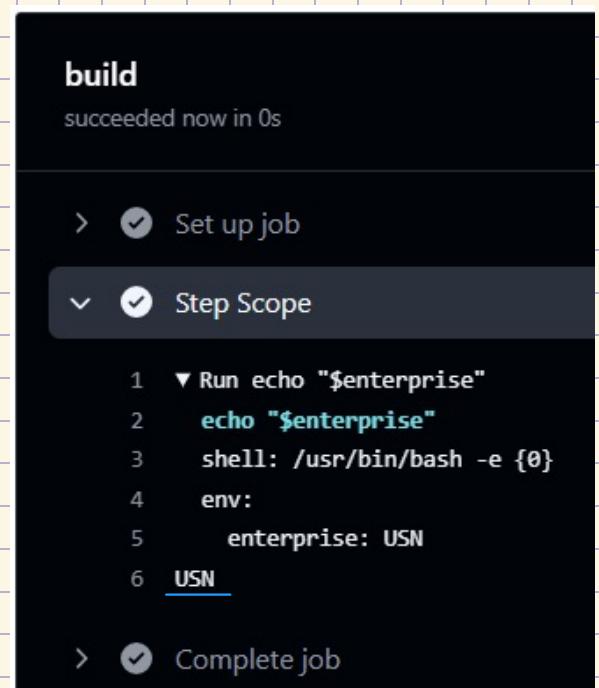
## # Variables in Job Scope

```
.github > workflows > go.yml
1  name: GoProgram
2
3  on:
4    push:
5      branches: [ "main" ]
6
7  jobs:
8    build:
9      runs-on: ubuntu-latest
10
11   env:
12     furutaka: "IJN"
13
14   steps:
15     - name: Job Scope
16       run: echo "$furutaka"
```



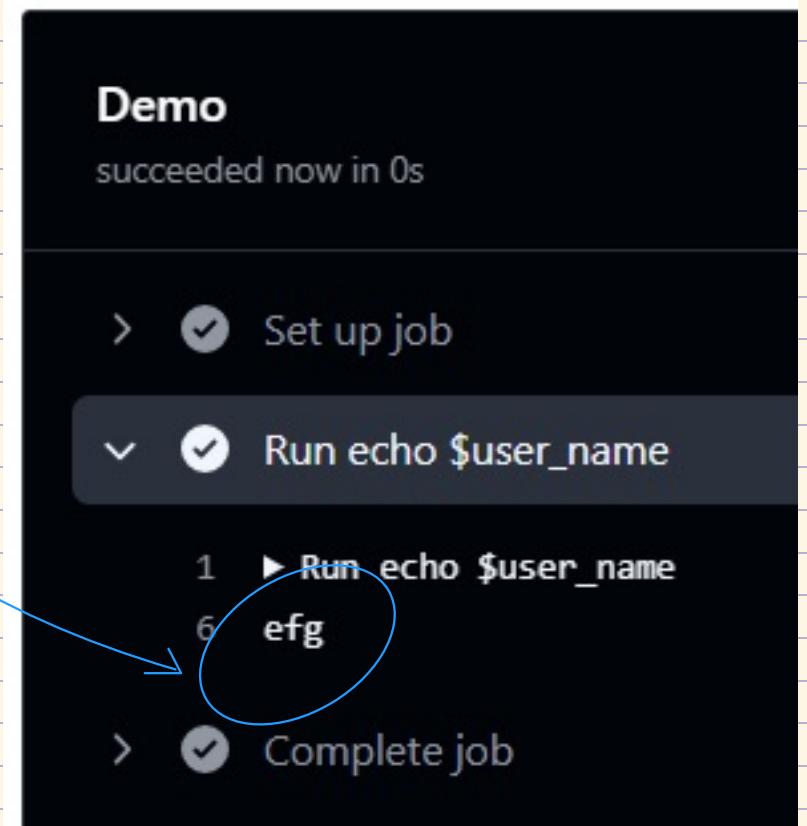
## # Variables in Step Scope

```
.github > workflows > go.yml
1  name: GoProgram
2
3  on:
4    push:
5      branches: [ "main" ]
6
7  jobs:
8    build:
9      runs-on: ubuntu-latest
10
11   steps:
12     - name: Step Scope
13       env:
14         enterprise: "USN"
15       run: echo "$enterprise"
```



## # Override

```
.github > workflows > go.yml
1   name: GoProgram
2
3   on: push
4
5   env:
6     user_name: "abc"
7
8   jobs:
9     Demo:
10    runs-on: ubuntu-latest
11    env:
12      user_name: "efg"
13    steps:
14      - run: echo $user_name
```



## # Default Variables

A GitHub Actions workflow has a set of default variables.

<https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/store-information-in-variables#default-environment-variables>

The diagram illustrates the relationship between the official GitHub Actions documentation and a sample workflow file. A red dashed box encloses the URL from the previous slide. Two arrows point from this box to the image: one arrow points to the word "official" above the workflow file, and another arrow points to the word "using" above the workflow file, indicating that the documentation is being used to explain the file.

.github > workflows > go.yml

```
1 name: GoProgram
2
3 on:
4   push:
5     branches: [ "main" ]
6
7 env:
8   user_name: "paul"
9
10 jobs:
11   build:
12     runs-on: ubuntu-latest
13
14   steps:
15     - name: Default
16       run: echo "$GITHUB_WORKFLOW"
```

build  
succeeded 1 minute ago in 0s

- >  Set up job
- ✓  Default
  - 1 ► Run echo "\$GITHUB\_WORKFLOW"
  - 6 GoProgram
- >  Complete job

The ability to keep secret values is an essential feature in any CI/CD pipeline implementation tool, because some parameters/variables are sensitive information that cannot be stored openly. While programmatically allowing access to third parties may be necessary.

Next, the rest of this chapter explores the options for keeping secrets in GitHub Actions, as well as using them to provide programmatical access to GitHub.

## # Repo - Level Secrets

The screenshot shows the GitHub repository settings page. The left sidebar is collapsed. The main area displays the "Actions secrets and variables" section. It includes a sidebar with navigation links like General, Access, Collaborators, Moderation options, Code and automation, Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, and Pages. The "Secrets and variables" link under "Code and automation" is highlighted with a blue oval. The main content area shows a summary of secrets and variables, a "Manage environment secrets" button, and a "Repository secrets" section with a "New repository secret" button, also highlighted with a blue oval.

## Actions secrets / New secret

Name \*  
ANHAO\_PASSWORD

Secret \*  
Etajima

```
- name: 登录 DockerHub
  uses: docker/login-action@v3
  with:
    username: ${{ secrets.DOCKER_HUB_USERNAME }}
    password: ${{ secrets.DOCKER_HUB_TOKEN }}
```

Add secret

To Define

To Use

## # Organization-Level Secrets

The screenshot shows the GitHub organization settings for 'SI-DevOps-Monitoring'. The left sidebar lists various settings categories like General, Access, and Security. The 'Secrets and variables' section is currently selected, indicated by a blue border. The main content area displays information about secrets and variables, stating 'There are no secrets for this organization.' A green button at the bottom right says 'New organization secret'.

## # Rules

1. To use a secret in the workflow, you need collaborator permission.
2. The secrets created in a GitHub repo are not available to the repo's folks.
3. Once a secret is created, the value cannot be seen again, but it can be utilized in the workflow.  
If required, you can either remove or update the secret to a new value.
4. Organization secrets are available to private repositories with the paid plans.  
Organization secrets are available in public repositories through workflows.
5. GitHub always redacts the secrets printed in workflow logs.  
However, you should take care not to accidentally print the secrets to logs.

## # Naming Considerations for Secrets

1. Alphanumeric + Underscore

[a-zA-Z\_][a-zA-Z\_0-9]

2. Case Sensitivity ✓

3. Unique

Secret names must be unique at the repo or organization level.

If you define a secret name at the organization level and use the same secret name in the organization's repos, precedence is given to the repo-level secrets.

4. GITHUB\_\* ✘

## # GitHub\_Token

Case:

to create an issue if a workflow fails

```
name: GoProgram

on: push

jobs:
  FailJobIssueDemo:
    permissions: write-all
    runs-on: ubuntu-latest
    steps:
      - name: Step is going to pass
        run: echo Passing step
      - name: Step is going to fail
        run: exit 1
      - name: Step To run on failure
        if: ${{ failure() }}
        run:
          curl \
            --request POST \
            --url https://api.github.com/repos/${{ github.repository }}/issues \
            --header 'authorization: Bearer ${{ secrets.GITHUB_TOKEN }}' \
            --header 'content-type: application/json' \
            --data '{
              "title": "Issue created due to workflow failure: ${{ github.run_id }}",
              "body": "This issue was automatically created by the workflow **${{ github.workflow }}**."
            }'
```

The screenshot shows a GitHub Issues page with the following details:

- Filters:** is:issue is:open
- Issues:**
  - 1 Open ✓ 0 Closed
  - Issue created due to workflow failure: 12481786807
- Issue Details:**
  - Title: Issue created due to workflow failure: 12481786807
  - Body: This issue was automatically created by the workflow \*\*\${{ github.workflow }}\*\*.
  - Author: bot
  - Labels: 9
  - Milestones: 0
  - Assignee: None
  - Sort: None

# Artifacts

## # Storing Content in Artifacts

When I execute a build or test run in a GitHub Actions workflow, it will generate binaries or test results as the output of the workflow. These items may be stored for next jobs in the same workflow, or across workflows. GitHub storage is utilized to store artifacts.

```
.github > workflows > go.yml
1   name: Lollipop
2
3   on: push
4
5   jobs:
6     ToBuild:
7       runs-on: ubuntu-latest
8       steps:
9
10      - name: Copy source files
11        uses: actions/checkout@v4
12
13      - name: Set up Go
14        uses: actions/setup-go@v4
15        with:
16          go-version: '1.20'
17
18      - name: Build
19        run: go build main.go
20
21      - name: Check
22        run: ls
23
24      - name: Upload artifact
25        uses: actions/upload-artifact@v4
26        with:
27          name: my-go-app
28          path: ./main # Replace with the actual name of your compiled binary
```

to upload an artifact,  
the default retention  
period is 90 days

You can download artifacts from the workflow once it is completed.

The screenshot shows a GitHub Actions workflow log and a file explorer window. The workflow log details the execution of various steps, including 'Upload artifact'. The file explorer window shows a folder named '.github' containing a file named 'my-go-app.zip'. A blue arrow points from the 'download URL' in the workflow log to the file in the file explorer.

ToBuild

succeeded now in 18s

>  Set up job

>  Copy source files

>  Set up Go

>  Build

>  Check

>  Upload artifact

1 ► Run actions/upload-artifact@v4

9 With the provided path, there will be 1 file uploaded

10 Artifact name is valid!

11 Root directory input is valid!

12 Beginning upload of artifact content to blob storage

13 Uploaded bytes 1121001

14 Finished uploading artifact content to blob storage!

15 SHA256 hash of uploaded artifact zip is 84757678e779719f8659796de2979680c2b9317ea8c2c828135f97f6701f857

16 Finalizing artifact upload

17 Artifact my-go-app.zip successfully finalized. Artifact ID 2365220985

18 Artifact my-go-app has been successfully uploaded! Final size is 1121001 bytes. Artifact ID is 2365220985

19 Artifact download URL: <https://github.com/AnhaoROMA/GithubActions/actions/runs/12513341421/artifacts/2365220985>

>  Post Set up Go

>  Post Copy source files

>  Complete job

Or from browser :

The screenshot shows a GitHub Actions run summary for a workflow named 'Lollipop'. The run was triggered via push 2 minutes ago by user 'AnhaoROMA' to branch 'main'. The status is Success, total duration is 28s, and there is 1 artifact. The job 'ToBuild' completed successfully in 18s. Annotations show two warnings related to 'ToBuild' steps. The artifact 'my-go-app' is listed with a size of 1.07 MB. A blue arrow points from the 'my-go-app' artifact section towards the 'ToRun' code snippet below.

## # Within the Same Workflow

Let's look at an example scenario where artifacts must be passed to another jobs in the same workflow. build steps are done on an Ubuntu runner and the built binaries will be deployed to MacOs.

Append the following snippet to the former :

```
31   ToRun:
32     runs-on: ubuntu-latest
33     needs: ToBuild
34     steps:
35       - name: Download artifact
36         uses: actions/download-artifact@v4
37         with:
38           name: my-go-app
39       - name: Draw
40         run:
41           chmod 777 main
42           ./main
```

waits job "ToBuild" to finish first

corresponding

optionally a path to download artifacts

Succeeded !

The screenshot shows a GitHub Actions run summary for a workflow named 'Lollipop'. The run was triggered via push 2 minutes ago by user 'AnhaoROMA' to branch 'main'. The status is Success, total duration is 3s, and there is 1 artifact. The job 'ToBuild' completed successfully in 19s, followed by the job 'ToRun' which completed successfully in 3s. A blue arrow points from the 'ToRun' job in the run summary towards the 'ToRun' code snippet above.

ToRun  
succeeded 1 minute ago in 3s

Search logs

- > Set up job 0s
- > Download artifact 1s
 

```

1 ► Run actions/download-artifact@v4
7 Downloading single artifact
8 Preparing to download the following artifacts:
9 - my-go-app (ID: 2365270879, Size: 1121001)
10 Redirecting to blob download url: https://productionresultss10.blob.core.windows.net/actions-results/aa9048ce-3cb5-45f8-a195-e5d42c498341/workflow-job-run-0287d5ad-f053-5a28-cb57-79dd564cf74/artifacts/d01cd0b510f524ccabe1108a2b7b32c18a20fef53d0f4c5a1047bc4091ab3d75.zip
11 Starting download of artifact to: /home/runner/work/GithubActions/GithubActions
12 (node:1738) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the Buffer.alloc(), Buffer.allocUnsafe(), or Buffer.from() methods instead.
13 (Use `node --trace-deprecation ...` to show where the warning was created)
14 Artifact download completed successfully.
15 Total of 1 artifact(s) downloaded
16 Download artifact has finished successfully
      
```


- > Draw 0s
- > Complete job 0s

One thing to note is that the version of actions/upload-artifact and actions/download-artifact must be the same (@V3 or @V4).

<https://stackoverflow.com/questions/78238187/unable-to-find-any-artifacts-for-the-associated-workflow-github-actions>

## # Across Different Workflows

I can implement this via REST API.

<https://docs.github.com/en/rest/actions?apiVersion=2022-11-28#artifacts>

but I will not use this method this time.

To make a new workflow (yaml file).

EXPLORER

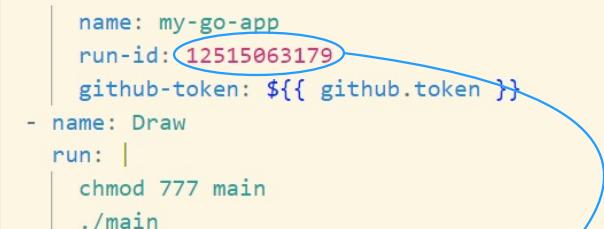
GITHUB ACTIONS

- .github\workflows
  - buildAngular.yml
  - runAngular.yml
  - go.mod
  - go.work
  - main.go
  - README.md

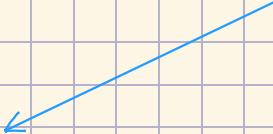
buildAngular.yml runAngular.yml main.go

```

.name: RunAngular
.on: workflow_dispatch
.jobs:
  ToRun:
    runs-on: ubuntu-latest
    steps:
      - name: Download artifact
        uses: actions/download-artifact@v4
        with:
          name: my-go-app
          run-id: 12515063179
          github-token: ${{ github.token }}
      - name: Draw
        run: |
          chmod 777 main
          ./main
      
```



can retrieve from  
the build-angular



https://github.com/AnhaoROMA/GithubActions/actions/runs/12515063179/job/34912055215

Google Lens ⌂ ☆

Supposed that :

- △ build-angular is on push
  - △ run-angular is on workflow\_dispatch
- , so run-angular is manually started, using the artifact(s) generated by build-angular.

Reference :

<https://github.com/marketplace/actions/download-workflow-artifact>

<https://stackoverflow.com/questions/60355925/share-artifacts-between-workflows-github-actions>

" Persisted data in one job can be passed to another subsequent jobs, which may be running on different operating systems. "

# Caching Dependencies

"Reusable files can be cached, which considerably reduces the execution time of a GitHub Actions workflow."

When jobs are executed in GitHub-hosted runners, they always run in a fresh and clean virtual environment. But a clean environment demands downloading all required dependencies in each job run. As a solution, I can use GitHub's capabilities to cache dependencies.

may include files (utilized by package and dependency management tool such as npm, yarn, Gradle)

However, don't cache sensitive values in public repositories because forked repos can obtain cached information.

Prepare a Go program and a workflow.

```
package main

import (
    "fmt"
    "os"
    "time"
)

func pathExists(path string) (bool, error) {
    _, err := os.Stat(path)
    if err == nil {
        return true, nil
    }
    if os.IsNotExist(err) {
        return false, nil
    }
    return false, err
}

func main() {
    ifExist, err := pathExists("/tmp/temp/1.txt")
    if err != nil {
        fmt.Println("Failed!")
    }
    if !ifExist {
        time.Sleep(15 * time.Second)
    }
}
```

```
name: Cache

on: push

jobs:
  CacheTesting:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: cache
        uses: actions/cache@v4
        with:
          path: /tmp/temp/1.txt
          key: signal
      - name: Set up Go
        uses: actions/setup-go@v4
        with:
          go-version: '1.20'
      - name: check_1
        run:
          go run main.go
      - name: create
        run:
          mkdir -p /tmp/temp/
          echo "-----" > /tmp/temp/1.txt
      - name: check_2
        run: ls -l /tmp/temp/1.txt
```

The path(s) on the runner to cache or restore.

The key created when saving a cache and the key used to search for a cache.

如果该文件不存在，则停顿15秒。

When I executed the workflow for the first time, there is no cache available in the repo, so the file `/tmp/temp/1.txt` will be "cached".

The screenshot shows the GitHub Actions logs for a workflow named "CacheTesting". The logs are displayed in a dark-themed interface. The workflow consists of several steps: "Set up job", "Run actions/checkout@v4", "cache", "Set up Go", "check\_1", "create", "check\_2", "Post Set up Go", "Post cache", "Post Run actions/checkout@v4", and "Complete job". The "cache" step includes a command to run `actions/cache@v4` and a note that "Cache not found for input keys: signal". The "Post cache" step includes a command to run `/usr/bin/tar --posix -cf cache.tzst --exclude cache.tzst -P -C /home/runner/work/GithubActions/GithubActions --files-from manifest.txt --use-compress-program zstdmt` and a note that "Cache saved successfully" and "Cache saved with key: signal". The logs also show the creation of files in the temporary directory.

And I can find the cache in the repo.

The screenshot shows the GitHub Actions Caches page. The left sidebar has sections for "Actions", "New workflow", "All workflows", "Cache", "Management", and "Caches" (which is selected). The main area displays a table of caches. It shows three entries: one for "signal" (200 bytes, last used 4 minutes ago), and two for "victoria-metrics-linux-amd64-v1.108.1.tar.gz" (12 MB, last used 31 minutes ago; and another 186 bytes, last used 39 minutes ago). A search bar at the top right says "Filter caches".

Since there is no `/tmp/temp/1.txt` when running "check\_1", the total run time is 33s (has a 15s pause).

In subsequent runs, the cached file(s) will be used.

The screenshot shows the GitHub Actions logs for a subsequent workflow run of "CacheTesting". The logs show the same sequence of steps as the first run, but with different execution times. The "cache" step now includes a note that "Cache restored successfully" and "Cache restored from key: signal". The "Post cache" step includes a note that "Cache hit occurred on the primary key signal, not saving cache". The logs also show the creation of files in the temporary directory.

Since the cache is available, the pipeline doesn't save the cache again.

Since there is `/tmp/temp/1.txt` when running "check\_1", the total run time is 17s (no 15s pause).

GitHub's policy is to remove cached files not accessed for seven days.

I can create many caches, but there is a 5 GB size limit for all caches in the repository.

# Using Self-Hosted Runners

GitHub provides hosted runners, or in other words, Windows, Linux, and MacOS machines, as workflow runners.

Hosted runner information can be found at: <https://github.com/actions/runner-images/tree/main/images>.  
A VM runner-supported software list is specified in the `readme.md` file in each repo folder.

You may have specific software needs to build and deploy your applications. You may want to deploy to an on-premises environment utilizing GitHub Actions workflows. To cater to your needs, you can set up your (virtual) machines as runners for GitHub Actions.

Self-hosted runners can be added at different levels in GitHub.

- △ Repository level
- △ Organization level
- △ Enterprise level

However, be careful not to add a self-hosted runner to a public repository because folks in your public repos can execute malicious code by utilizing your runner.