

GitHub Actions

Alexandre Abadie, Thierry Martinez (SED)

Two-hours training, 15 december 2022

GitHub Actions

- A [continuous integration](#) (CI) platform for GitHub-hosted projects, [launched on 16 October 2018](#).
- Providing [GitHub-hosted runners](#) for Linux, MacOS and Windows.
- File-based workflow specification: `.github/workflows/*.yaml`.
A command-line tool, `act` is available for running workflows locally (or from other continuous integration platforms):
<https://github.com/nektos/act>
- Very easy to extend (new [reusable actions](#) can be defined in git repositories), Linux runners can run docker containers, user-provided runners can be used.

Continuous integration

Continuous integration (CI): practice of short-lived development cycles, automatically tested and shared regularly between developers involved in a project.

Continuous integration platforms: [Github Actions](#), [ci.inria.fr](#), [gitlab.inria.fr](#).

Automating testing (and CI in general) relies on [version control](#) and automated builds.

- speed up development process,
- ease collaboration
- allow programmers to be more confident for not introducing regression and bugs.

This is a step towards broader goals such as [reproducible builds](#) and [reproducible research](#).

About [version control](#)

Version control systems are software dedicated for managing

- history and
 - collaborative edition
- of source code or any other kind of documents.

The prominent software for version control is now [git](#), initially developed in 2005 by Linus Torvalds to manage the Linux source code.

[git](#) is a decentralized tool (where versions are directly exchanged between peers) but most uses of it now rely on [software forges](#), like [GitHub] or [gitlab.inria.fr](#) for instance. Software forges provide other services related to version control, such as [CI/CD] facilities.

Keeping the history of a code is central

- to make change in the code without losing information and
- to identify where regressions have been introduced ([bisection](#)).
- to allow code to be modified concurrently by offering merging facilities ([three-way merge](#)).

GitHub-hosted runners

- Hardware specification for Windows and Linux virtual machines:
 - 2-core CPU (x86_64)
 - 7 GB of RAM
 - 14 GB of SSD space
- Hardware specification for macOS virtual machines:
 - 3-core CPU (x86_64)
 - 14 GB of RAM
 - 14 GB of SSD space

[Usage limits, billing](#): available for free for public repositories,

- up to 20 concurrent jobs (Linux/Windows),
- 5 concurrent jobs for macOS.

Self-hosted runners

<> Code Issues Pull requests Actions Projects Wiki Security Insights **Settings**

General

Access

- Collaborators
- Moderation options

Code and automation

- Branches
- Tags
- Actions

General

Runners

Webhooks

Environments

Codespaces

Pages

Security

- Code security and analysis
- Deploy keys
- Secrets

Runners / Create self-hosted runner

Adding a self-hosted runner requires that you download, configure, and execute the GitHub Actions Runner. By downloading and configuring the GitHub Actions Runner, you agree to the [GitHub Terms of Service](#) or [GitHub Corporate Terms of Service](#), as applicable.

Runner image

☐ macOS

☒ Linux

☐ Windows

Architecture

x64

Download

```
# Create a folder
$ mkdir actions-runner && cd actions-runner

# Download the latest runner package
$ curl -o actions-runner-linux-x64-2.299.1.tar.gz -L https://github.com/actions
/runner/releases/download/v2.299.1/actions-runner-linux-x64-2.299.1.tar.gz

# Optional: Validate the hash
$ echo "147c14700c6cb997421b9a239c012197f11ea9854cd901ee88ead6fe73a72c74  actions-
runner-linux-x64-2.299.1.tar.gz" | shasum -a 256 -c

# Extract the installer
$ tar xzf ./actions-runner-linux-x64-2.299.1.tar.gz
```

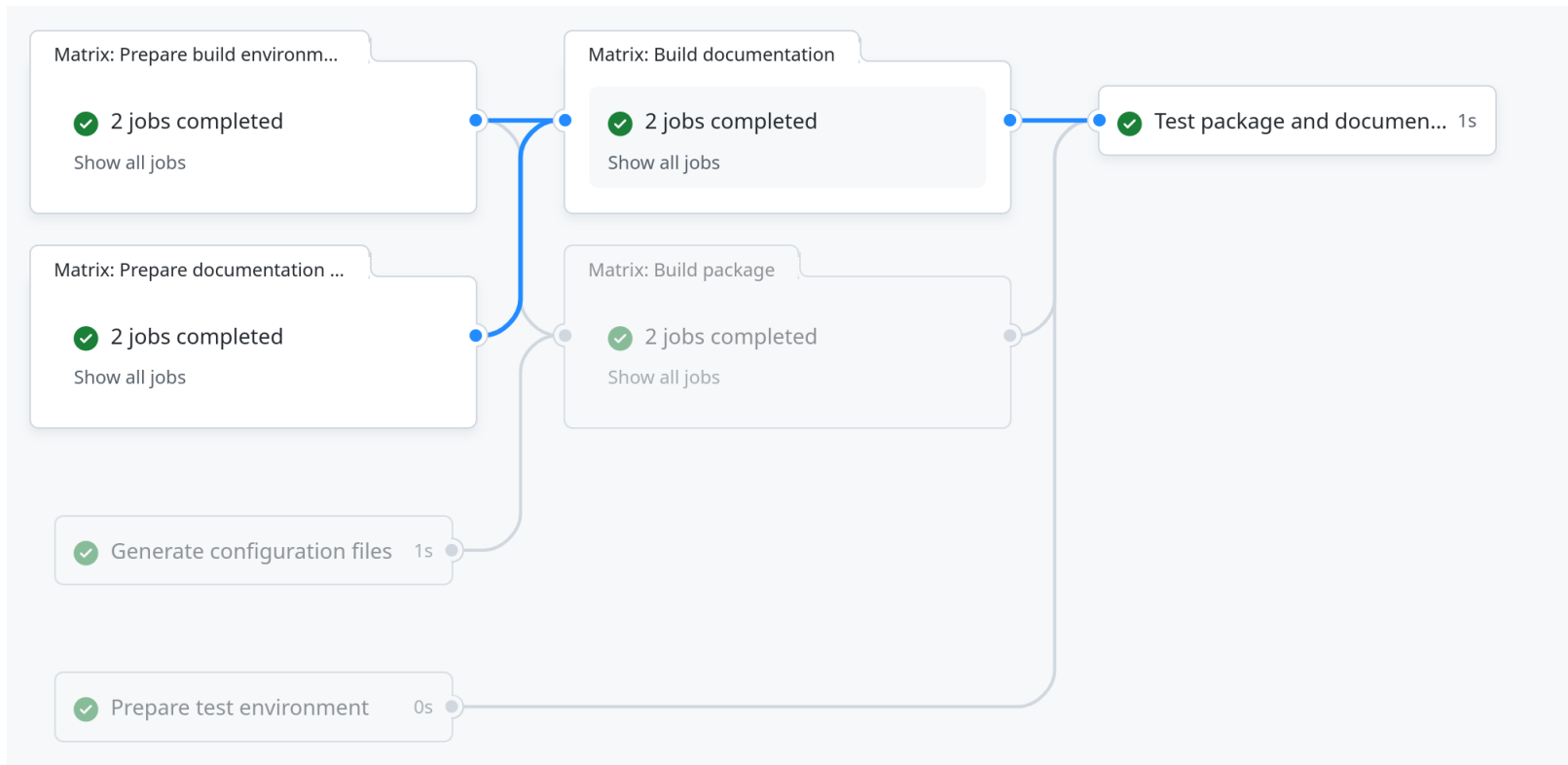
Example of workflow

In `.github/workflows/example.yml`:

```
on: [push]
jobs:
  build-example:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Compile
        run: |
          gcc -o hello_word hello_world.c
      - name: Test
        run: |
          ./hello_word > output.txt
          diff output.txt expected.txt
```


Dependency graph


- Workflow can have arbitrary complex directed acyclic graph as dependency graph.





Status feedback


- On repository index


 **thierry-martinez** Fix syntax ✓ abd185a 17 minutes ago 🕒 30 commits


 .github/


 .gitigno

 README


 myfunc


 setup.cf


 test-req


 test_myfuncs.py


All checks have passed
20 successful checks

✓  CI / Generate configuration files (push) Successf... [Details](#)

✓  CI / python-tests (ubuntu-latest, 3.9) (push) Succ... [Details](#)

✓  CI / python-tests (ubuntu-latest, 3.10) (push) Suc... [Details](#)

✓  CI / python-tests (macos-latest, 3.9) (push) Succe... [Details](#)

✓  CI / python-tests (macos-latest, 3.10) (push) Succ... [Details](#)

17 minutes ago

17 days ago

17 days ago

17 days ago

17 days ago

17 days ago

17 days ago

- In `README.md` badges:


```
[![CI][ci-badge]][ci-link]
```




☰ README.md


 CI passing  codecov 100%

Status feedback

- In pull requests (in addition, posts can generated by bots invoked from CI)

 **thierry-martinez** and others added 3 commits 17 days ago

-  Update `.github/workflows/continuous-integration.yml` ... Verified ✓ 59aaacf
-  Use artifact to communicate coverage.xml ✓ 6cffbdb
-  Add checkout to code coverage ✓ a6e534d


 **codecov** bot commented 17 days ago • edited ▾ 😊 ...


Codecov Report

Base: **100.00%** // Head: **100.00%** // No change to project coverage 👍

Coverage data is based on head ([a6e534d](#)) compared to base ([cae081e](#)).
Patch has no changes to coverable lines.

► Additional details and impacted files

 [View full report at Codecov.](#)

 Do you have feedback about the report comment? [Let us know in this issue.](#)

Repository initialization

- We will use GitHub command line: <https://cli.github.com/>
- Fork training course repository:

```
gh repo fork --clone aabadie/github-actions-python-example
```

(to create a new repository instead: `gh repo create <repo name> --public --clone`)

- Edit workflow in `github-actions-primer/.github/workflows/continuous-integration.yml`

See [Choosing GitHub hosted runners](#) for a list of available platforms for `runs-on` entry. Note that `ubuntu-latest` is currently Ubuntu 20.04. There is `ubuntu-22.04` available in beta.

Running jobs in a container

Build environments can be prepared once for all in a Docker image to reduce build times:

- `docker build -t ghcr.io/<user>/<image name> .`
- [create a personal access token](#) with scope `write:packages`, save it in a file
- `docker login ghcr.io -u <user> --password-stdin < <token path>`
- `docker push ghcr.io/<user>/<image name>`
- create a personal access token with scope `read:packages`, store it in a [secret](#) (using `gh secret set`)
- reference the container in the job

```
container:  
  image: ghcr.io/<user>/<image name>  
  credentials:  
    username: ${github.actor}  
    password: ${secrets.<secret name>}
```

- to run the workflow locally, use `act --secret-file <file name>`

Matrix job

- Use `strategy.matrix` to build the same job with different combinations of parameters.
- Set `strategy.fail-fast: false` to continue the build of other combinations when a combination failed.

Use a job to build the environment

- Store a personal access token] with scope `write:packages` in a secret.
- Check out the repository! Add the following action

```
- name: Checkout  
  uses: actions/checkout@v3
```

- Steps for `docker build` and `docker push` .



`checkout` action wipes out the current directory!

Should be run before any actions writing useful things in it (local setup, etc.).

Run a job only if a file has changed

- Checkout with the input `fetch-depth: 2` to get the two last commits (by default, only the last commit is checked out, i.e. `git fetch --depth=1`)
- Use `git diff --quiet --exit-code HEAD^ HEAD -- <path>` to check if a file changed.

⚠️ Commands should succeed (with return code 0). Use `if-then-else-fi` to control the result of `git diff`.

- Can be done in another job, using [job outputs](#) and [conditions](#).
- We only want to build the image if `Dockerfile` has changed, but the main job should be run even if the build job has been skipped: use `always()` and check `needs.<job_id>.result` for `success` or `skipped`.

Using artifacts and deploy release

- [Storing workflow data as artifacts](#): `actions/upload-artifact@v3` with inputs `name` and `path`, `actions/download-artifact@v3` with input `name`.
- [softprops/action-gh-release](#) with input `files`

Adding self-hosted runners

- In Project Settings › Actions › Runners, button *New self-hosted runner*. Follow the instructions. Tags match the values of `runs-on:` field.
- `./run.sh` can be run in `tmux` or as a service.

Workflow commands

- Annotations: error/notice/warning

```
echo "::error file=app.js,line=1::Missing semicolon"  
echo "::add-matcher::matcher.json"
```

Matcher for gcc: `ammaraskar/gcc-problem-matcher@master`

- Grouping log lines

```
::group::{title}  
::endgroup::
```

- Masking a value

```
::add-mask::{value}
```

Environment files

- Setting an environment variable

```
echo "{environment_variable_name}={value}" >> $GITHUB_ENV
```

- Output parameter (steps setting outputs should have `id:`)

```
echo "{name}={value}" >> $GITHUB_OUTPUT
```

In subsequent steps, refer to `\${{ steps.{id}.outputs.{name} }}`

- Job summary

```
echo "### Hello world! :rocket:" >> $GITHUB_STEP_SUMMARY
```

- Adding a system path

```
echo "{path}" >> $GITHUB_PATH
```

Docker and registry

Docker is a tool to run programs in *containers*, that is to say

- sandboxes that share the same kernel as the running OS,
- but in a `chroot` (isolated filesystem),
- with limited access to resources (process groups, network, and more generally limited system calls).

Running jobs in a container

`container:` key specifies a Docker image (otherwise, the job runs directly on the VM).

Pushing a Docker image to GitHub registry

Build environments can be prepared once for all in a Docker image to reduce build times:

- `docker build -t ghcr.io/<user>/<image name> .`
- [create a personal access token](#) with scope `write:packages`, save it in a file
- `docker login ghcr.io -u <user> --password-stdin < <token path>`
- `docker push ghcr.io/<user>/<image name>`
- create a personal access token with scope `read:packages`, store it in a [secret](#) (using `gh secret set`)
- reference the container in the job

```
container:  
  image: ghcr.io/<user>/<image name>  
  credentials:  
    username: ${github.actor}  
    password: ${secrets.<secret name>}
```

- to run the workflow locally, use `act --secret-file <file name>`

Use a job to build the environment

- Store a personal access token with scope `write:packages` in a secret.
- Check out the repository! Add the following action

```
- name: Checkout  
  uses: actions/checkout@v3
```

- Steps for `docker build` and `docker push` .



`checkout` action wipes out the current directory!

Should be run before any actions writing useful things in it (local setup, etc.).

Continuous Delivery/Continuous Deployment

Continuous Delivery: Preparing new releases

- [Storing workflow data as artifacts](#): `actions/upload-artifact@v3` with inputs `name` and `path`, `actions/download-artifact@v3` with input `name`.
- [softprops/action-gh-release](#) with input `files`

Continuous Deployment: Pushing on pypi

- [pypa/gh-action-pypi-publish](#)

Publishing documentation

- [JamesIves/github-pages-deploy-action](#)

```
- name: Deploy
  uses: JamesIves/github-pages-deploy-action@v4
  with:
    folder: build
```

Creating custom actions

- 3 types of actions can be created: Docker, Javascript and Composite
- Docker actions can only be used on Linux runners
- Composite actions combines multiple workflow steps in a single action

Describing an action

- An action is described by a single `action.yml` file
- One can define the inputs, outputs and environment variables of an action
- If the action is designed to be reusable and public, use a dedicated public repository for the action

The name of the action corresponds to the name of the repo, e.g `orga / repo name @ branch/tag name`

- If the action is local to a repository, place the yml file in `.github/actions/<action name>/action.yml`
- Local actions are used in a workflow as follows, the checkout action must be called before:

```
- uses: actions/checkout@master
- name: Run local custom action
  uses: ../.github/actions/local-action
  ...
```

Example: a Docker action

- [Docker action example](#)
- [Docker image example](#)
- The Docker action can use a `Dockerfile` => GitHub will build the image when the action is run
- [Docker actions doc](#)

Example: a Javascript action

- [javascript action example](#)
- javascript actions require `node_modules/` to be committed
- [Javascript actions doc](#)
- [Actions toolkit](#)

Example: a composite action

- [composite action example](#)
- [Composite actions doc](#)

Example: an action published to the marketplace

- Repository to create an action
- Published action on the Marketplace
- GitHub automatically detect that the repo contain an action