

# 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
```


## Action status visible on repository index

 main ▾


Go to file

Add file ▾

 Code ▾


 **thierry-martinez** Remove newline at end of file ...

✓ 3 minutes ago ⌚ 2



 .github/w

**All checks have passed**  
1 successful check


6 minutes ago

 .gitignore

6 minutes ago

 ✓  .github/workflows/example.yml... [Details](#)

3 minutes ago

 hello\_world.c

Initial commit

6 minutes ago



# Access to logs in action details

← .github/workflows/example.yml

✓ Remove newline at end of file #2

Re-run all jobs

...

Summary

Jobs

✓ build-example

Run details

Usage

Workflow file

## build-example

succeeded 7 minutes ago in 16s

Search logs



> ✓ Set up job

1s

> ✓ Checkout

1s

▼ ✓ Compile

0s

```
1 ▶ Run gcc -o hello_word hello_world.c
4 hello_world.c: In function 'main':
5 hello_world.c:7:3: warning: implicit declaration of function 'exit' [-Wimplicit-
  function-declaration]
6     7 |   exit(0);
7       |   ^~~~
8 hello_world.c:2:1: note: include '<stdlib.h>' or provide a declaration of 'exit'
9     1 | #include <stdio.h>
```

## ci.inria.fr

- A cloud of virtual machines dedicated for continuous integration: VMs are created and maintained by the users. Linux, Windows and MacOS VMs are supported, but MacOS VMs are quite fragile for the moment (but we are working on it!).
- Provides yet another continuous integration system, Jenkins, and a part of it can be configured by a file-based workflow specification: `Jenkinsfile`, difficult to run locally (except by installing and maintaining a local instance of Jenkins...).
- Very extensible through plugins but fragile and hard to maintain. Most plugins should be configured via the web interface; there is an API covering many operations, but it is not well documented and there is no easy tool to call it (excepting `curl`!).

## GitHub-hosted runners

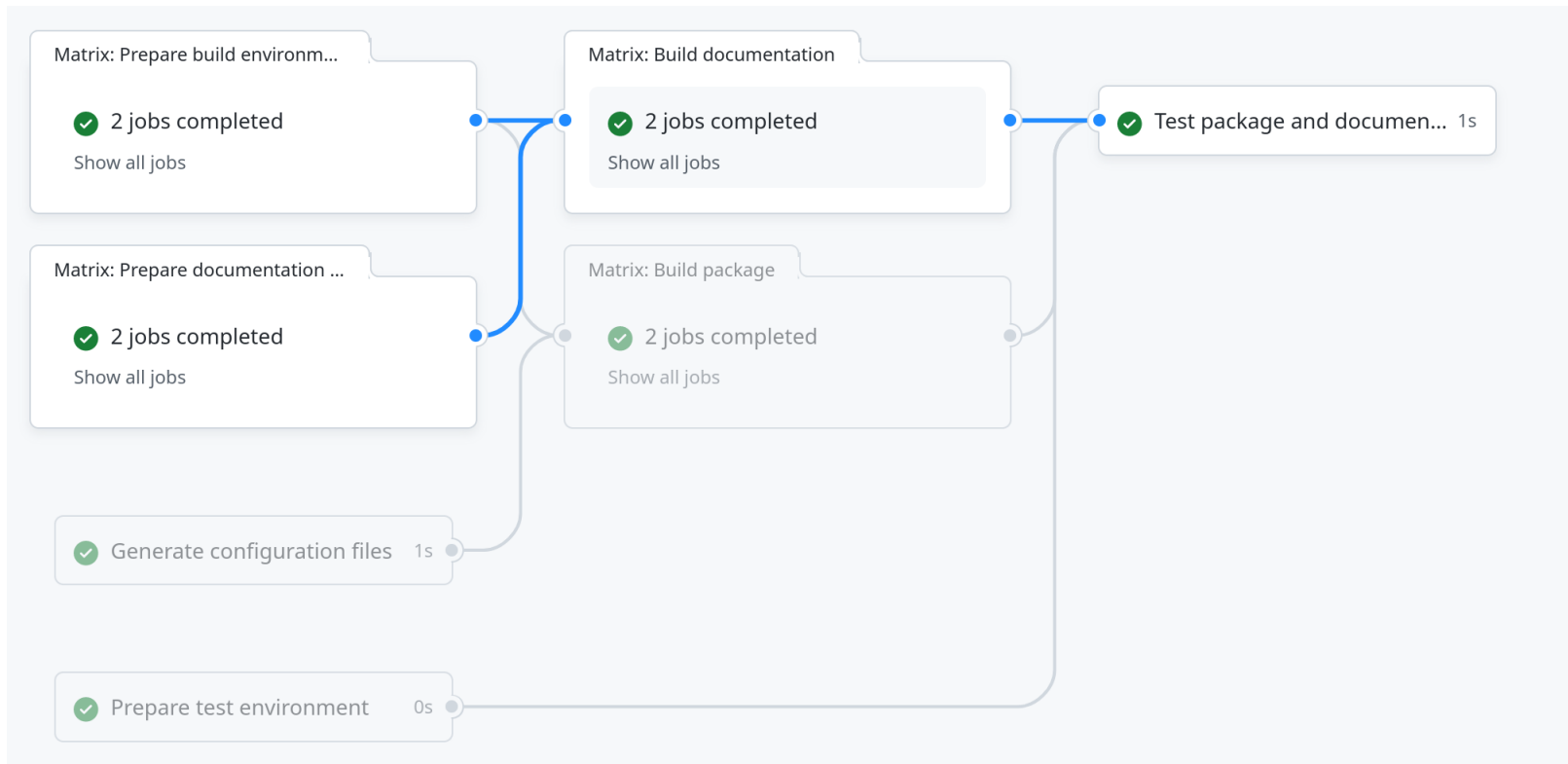
- Hardware specification for Windows and Linux virtual machines:
  - 2-core CPU (x86\_64)
  - 7 GB of RAM
  - 14 GB of SSD space
- Hardware specification for macOS virtual machines:
  - 3-core CPU (x86\_64)
  - 14 GB of RAM
  - 14 GB of SSD space

## **Self-hosted runners**

Self-hosted runners

## Dependency graph

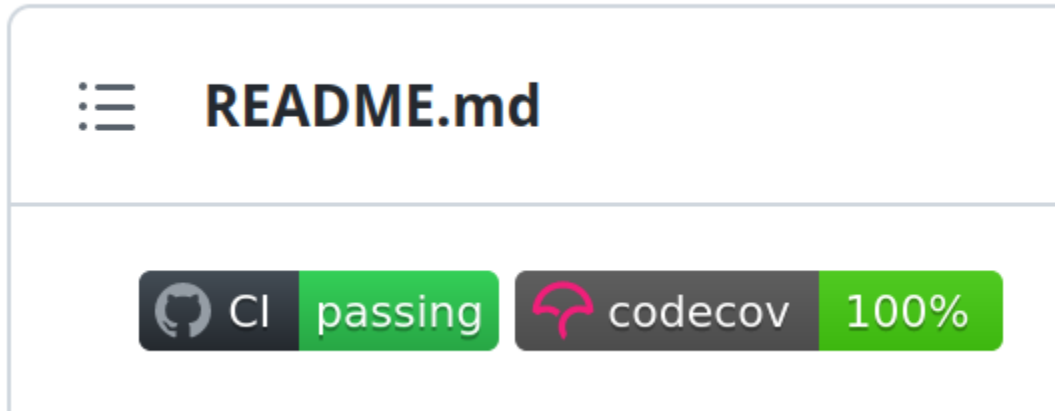
- Workflow can have arbitrary complex directed acyclic graph as dependency graph.



setup.ci	✓ CI / python-tests (macos-latest, 3.9) (push) Succ...	Details	17 days ago
test-req	✓ CI / python-tests (macos-latest, 3.9) (push) Succ...	Details	17 days ago
test_myfuncs.py	✓ CI / python-tests (macos-latest, 3.10) (push) Succ...	Details	17 days ago

- In `README.md` badges:

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



- 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

# **A primer on GitHub Action**

# Repository initialization

- We will use GitHub command line: <https://cli.github.com/>
- GitHub Actions run every workflow specified in files `.github/workflows/*.yaml` in a GitHub repository.
- `gh repo create github-actions-primer --public --clone`
- Put some contents in `github-actions-primer/.github/workflows/main.yaml`

```
name: main
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Preparing the environment
        run: |
          <enter some shell commands>
```

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.

- Use `gh run list` to check the status of workflow runs on the command-line.



# Run workflow locally

```
$ act
```

For platforms not supported out-of-box, one can provide a Docker image: for instance, to support `ubuntu-22.04`

```
act -P ubuntu-22.04=local-ubuntu-22.04
```

where `local-ubuntu-22.04` is a tag for an image built with a `Dockerfile` such as

```
FROM ubuntu:22.04
RUN apt-get update
RUN apt-get upgrade --yes
RUN apt-get install --yes sudo curl psmisc
```

⚠ Using versions of Ubuntu 21.10 in Docker images requires Docker 20.10.9 ([issue with syscall clone3](#)).

⚠ GitHub-hosted runners reduce interactions much more than `act` knows to do locally: think about adding options `--yes` and passing `DEBIAN_FRONTEND=noninteractive` in `apt-get` environment...

# 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>`

## 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`

## Some notes on Windows runner

- `windows-latest` runners come with `choco` and [some pre-installed tools](#), such as `7z`.

## 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.

## 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.



```
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.).

## 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

See [publishing on GitHub Marketplace]:

<https://docs.github.com/en/actions/creating-actions/publishing-actions-in-github-marketplace>

- 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
- Choose the GitHub release to publish to the Marketplace
- The name of the action corresponds to the name of the repo, e.g. orga / repo name