GitHub Actions Demystified

Data Club on 29.04.2025

by Henry Webel

Thanks to Pieter Verschaffelt for sharing his slides which these are built upon.

GitHub Actions Demystified

In this DataClub we will look at GitHub Actions. GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. Therefore, it can help bioinformaticians and data scientists to keep their software working by identifying bugs or dependency conflicts when your code or code you depend on changes. It also avoids that you to accidently forget certain checks you would normally perform. Therefore, if you are tired of repetitive task in your daily coding work or need more security, GitHub Actions can be the solution.

We will focus on introducing the core concepts of workflows and jobs in a hands-on manner using a simple hello-world example. Finally, we will show some more elaborated examples as an outlook for interested users.

Prerequisite is that you have a GitHub account and some familiarity with the command line. We do the tutorial on GitHub Codespaces using VSCode (see intro).

It will make most sense to attend if you want to use GitHub Actions (or similar on GitLab or BitBucket) on repositories where you save your package or analysis code. Then it will be easy for you after this Data club to add some basic checks, e.g. format checkers or lint checkers for common errors or pitfalls of your code.

Expectations (each bullet point)

- Included
  - How to set up and use Actions
  - How to automate
  - General knowledge
- Maybe:
  - Best practices
  - Collaborative spaces
  - Practical examples on how to apply them on programming projects.
  - Template for easy implementation
  - How to integrate into my data collection workflow
  - Which servers can work with GitHub libraries?
  - How to keep the GitHub repositories synchronized in your server?
- Out-of-scope
  - when my collaborators change the code, what should I do? (Pull-Requests?)
  - Testing and deploying scripts to Azure VM

Program overview

1. Introduction to GitHub Actions
2. First example
3. Yaml format (optional)
4. Workflow file in detail
5. *Vuegen repo: extended example (if time allows)*

What is GitHub Actions?

"GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline."

Source: docs.github.com

https://www.youtube.com/watch?v=URmeTqglS58

What is GitHub Actions?

- **Scripts** that run based on an event ('trigger') or at specific times
- Many possibilities
  - Perform unit testing and integration testing
  - Create new releases
  - Publishing python packages to PyPI
  - … everything you would normally do manually on your computer
- Can be used to protect PRs
  - PRs need specific checks to succeed before they can be merged

Is this free? Who can use GitHub Actions?

Pricing plans

Plan	Private repositories, per month	
	Storage	Minutes
GitHub Free	500MB	2 000
GitHub Pro* (Free for students)	1GB	3 000
GitHub Team	2GB	3 000
GitHub Enterprise Cloud	50GB	50 000

Minute multipliers

Operating System	Minute multiplier
Linux	1
Windows	2
macOS	10

”GitHub Actions usage is free for standard GitHub-hosted runners in public repositories, and for self-hosted runners.”

Source: About billing for GitHub Actions

Key concepts and terminology

Workflow

A configurable process that is defined by a YAML file inside of your repository and performs one or more specific actions (jobs) after a trigger.

- A single yaml file (simplified)

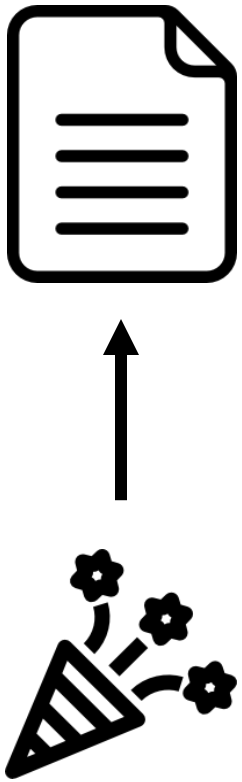
Key concepts and terminology

Workflow

A configurable process that is defined by a YAML file inside of your repository and performs one or more specific actions (jobs) after a trigger.

Trigger or Event

A specific event that automatically initiates a workflow within a GitHub repository.



Key concepts and terminology

Job

A workflow can have many jobs, which can be connected as a directed acyclic graph (DAG).

Step

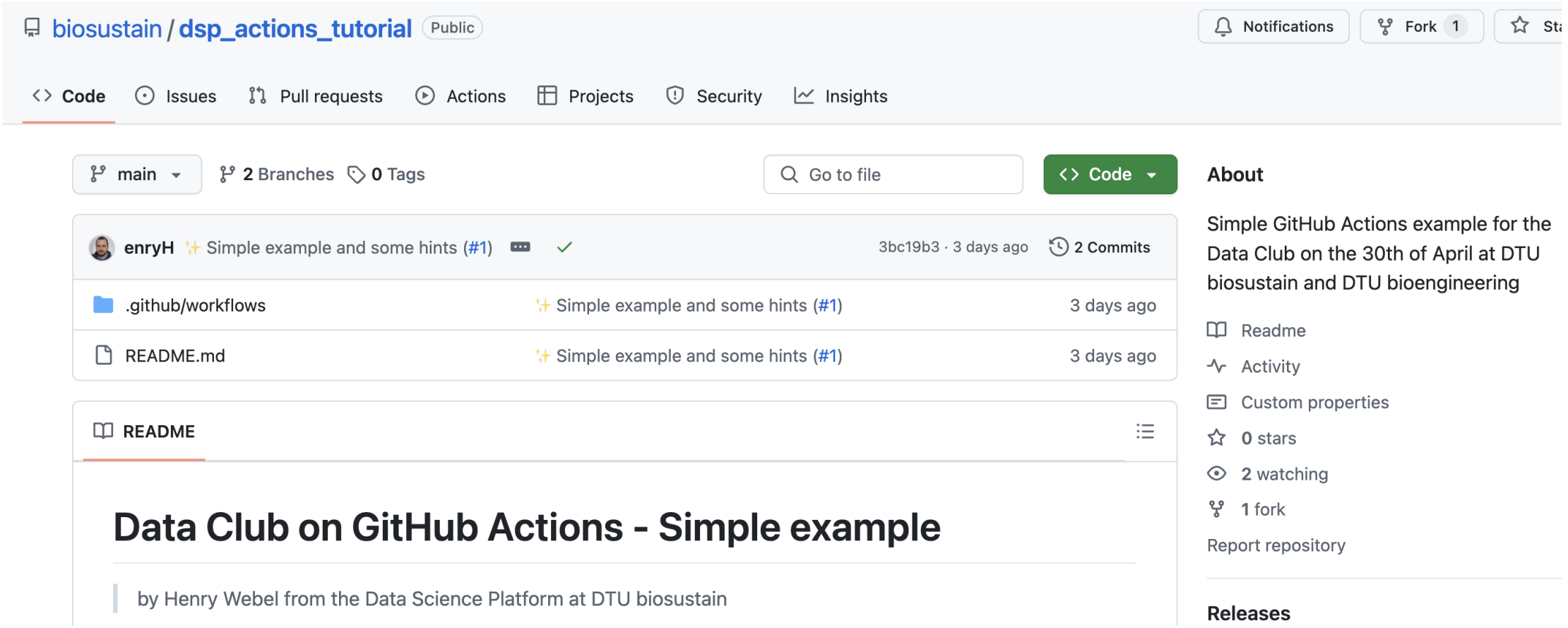
A single command or block of commands (small script) in a **job**.

Program overview

1. Introduction to GitHub Actions
2. First example
3. Yaml format (optional)
4. Workflow file in detail
5. *Vuegen repo: extended example (if time allows)*

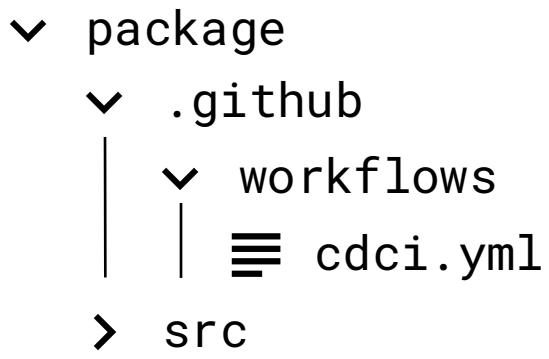
First example

github.com/biosustain/dsp_actions_tutorial



Directory structure of a GitHub repository

- All workflows live in the `.github/workflows` folder
- Each workflow is defined in a separate `.yml` file



Program overview

1. Introduction to GitHub Actions
2. First example
3. Yaml format (optional)
4. Workflow file in detail
5. *Vuegen repo: extended example (if time allows)*

YML?
Yet Another Markup Language

- Simple format used to describe a GitHub Action (workflow)
- Human-readable
  - Designed to be easily read and written by humans
- Use indentation (spaces) to denote structure
- Key-Value pairs
  - Each `key: value` pair represents data
  - Values can be lists of items
- Plus some syntactic sugar for actions, e.g. |

YML?

Lists

```yaml
items:
    - item1
    - item2
```

Key-value pairs and nested structures

```yaml
person:
    name: John Doe
    age: 30
    hobbies:
        - reading
        - cycling
```

Program overview

1. Introduction to GitHub Actions
2. First example
3. Yaml format (optional)
4. Workflow file in detail
5. *Vuegen repo: extended example (if time allows)*

Hello World example

.github > workflows > hello_world.yml

```yaml
# Hello World example for DTU biosustain and DTU bioengineering Data Club

name: Hello World example

on:
  push:

jobs:
  say-hello-world:
    runs-on: ubuntu-latest
    steps:
    - name: biosustain
      run: echo "Hello DTU biosustain!"
    - name: bioengineering
      run: echo "Hello DTU bioengineering!"
```

Hello World example

.github > workflows > hello_world.yml

```yaml
# Hello World example for DTU biosustain and DTU bioengineering Data Club

name: Hello World example

on:
  push:

jobs:
  say-hello-world:
    runs-on: ubuntu-latest
    steps:
    - name: biosustain
      run: echo "Hello DTU biosustain!"
    - name: bioengineering
      run: echo "Hello DTU bioengineering!"
```

Name: (required)
Describe your action in a few words. This will be used by GitHub and help you to easily find a specific action back.

Hello World example

.github > workflows > hello_world.yml

```yaml
# Hello World example for DTU biosustain and DTU bioengineering Data Club

name: Hello World example

on:
  push:

jobs:
  say-hello-world:
    runs-on: ubuntu-latest
    steps:
    - name: biosustain
      run: echo "Hello DTU biosustain!"
    - name: bioengineering
      run: echo "Hello DTU bioengineering!"
```

Triggers: (required)
After which event should this action be executed? Multiple triggers can be provided simultaneously.

Hello World example

.github > workflows > hello_world.yml

```yaml
# Hello World example for DTU biosustain and DTU bioengineering Data Club

name: Hello World example

on:
  push:

jobs:
  say-hello-world:
    runs-on: ubuntu-latest
    steps:
    - name: biosustain
      run: echo "Hello DTU biosustain!"
    - name: bioengineering
      run: echo "Hello DTU bioengineering!"
```

Jobs: (required)
A list of actions that should be executed by this workflow. At least one action is required!

Hello World example

.github > workflows > hello_world.yml

```yaml
# Hello World example for DTU biosustain and DTU bioengineering Data Club

name: Hello World example

on:
  push:

jobs:
  say-hello-world:
    runs-on: ubuntu-latest
    steps:
    - name: biosustain
      run: echo "Hello DTU biosustain!"
    - name: bioengineering
      run: echo "Hello DTU bioengineering!"
```

Job name

Hello World example

.github > workflows > hello_world.yml

```yaml
# Hello World example for DTU biosustain and DTU bioengineering Data Club

name: Hello World example

on:
  push:

jobs:
  say-hello-world:
    runs-on: ubuntu-latest
    steps:
    - name: biosustain
      run: echo "Hello DTU biosustain!"
    - name: bioengineering
      run: echo "Hello DTU bioengineering!"
```

Runner: (required)
Which operating system should this action be executed on? (e.g. ubuntu, windows, macos, ...)

Standard GitHub-hosted runners

about GitHub hosted runners

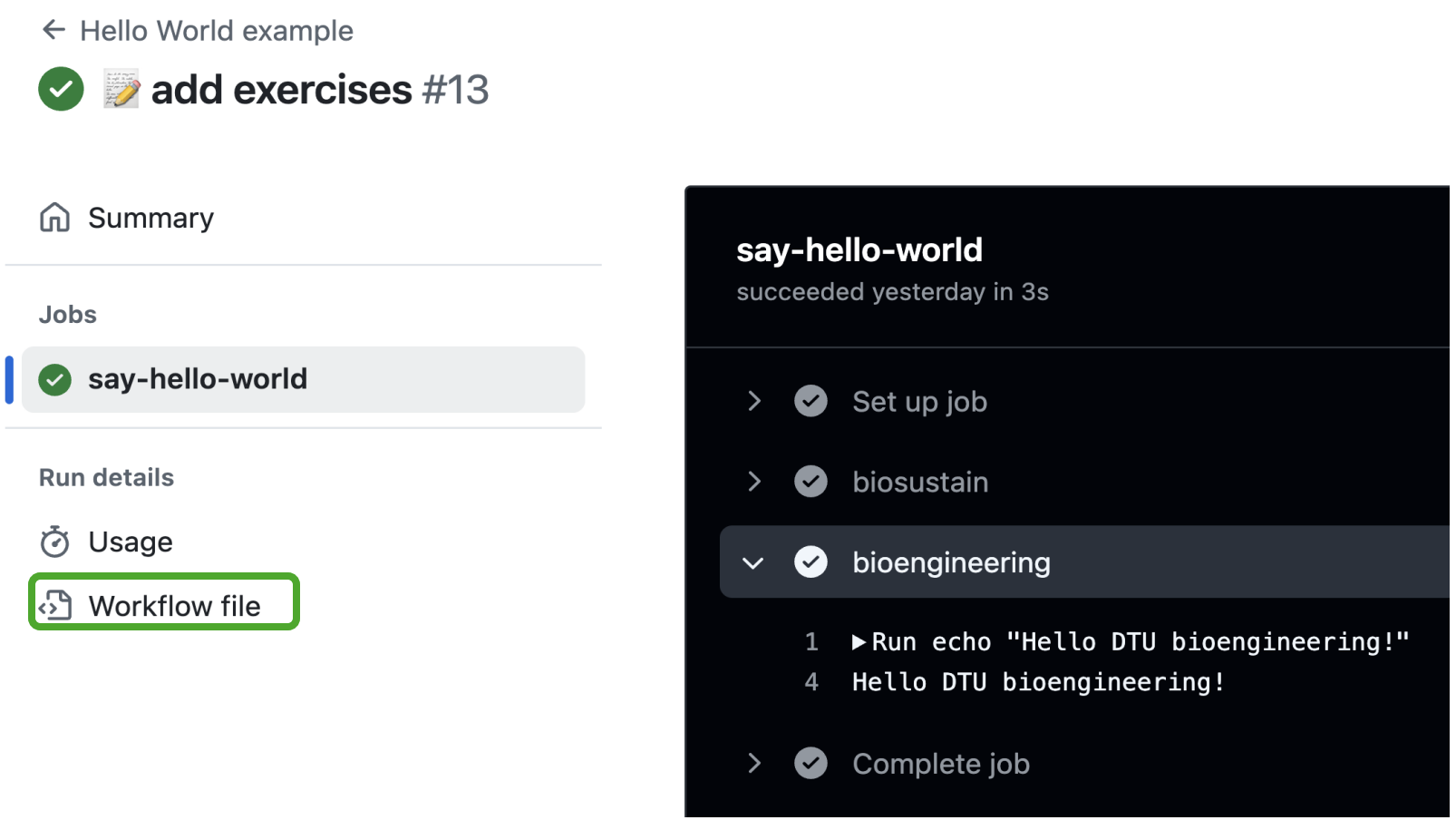
Virtual Machine	Processor (CPU)	Memory (RAM)	Storage (SSD)	Architecture	Workflow label
Linux	4	16 GB	14 GB	x64	`ubuntu-latest`, `ubuntu-24.04`, `ubuntu-22.04`, `ubuntu-20.04`
Windows	4	16 GB	14 GB	x64	`windows-latest`, `windows-2025` [Public preview], `windows-2022`, `windows-2019`
Linux [Public preview]	4	16 GB	14 GB	arm64	`ubuntu-24.04-arm`, `ubuntu-22.04-arm`
macOS	4	14 GB	14 GB	Intel	`macos-13`
macOS	3 (M1)	7 GB	14 GB	arm64	`macos-latest`, `macos-14`, `macos-15` [Public preview]

Hello World example

.github > workflows > hello_world.yml

```yaml
# Hello World example for DTU biosustain and DTU bioengineering Data Club

name: Hello World example

on:
  push:

jobs:
  say-hello-world:
    runs-on: ubuntu-latest
    steps:
    - name: biosustain
      run: echo "Hello DTU biosustain!"
    - name: bioengineering
      run: echo "Hello DTU bioengineering!"
```

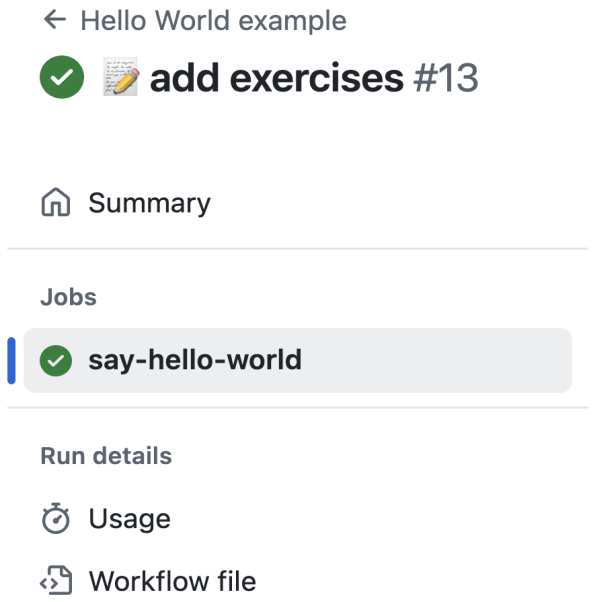
Steps: (required)
What does the action need to do? Steps are executed one-by-one. If a step fails, subsequent steps will not be executed.

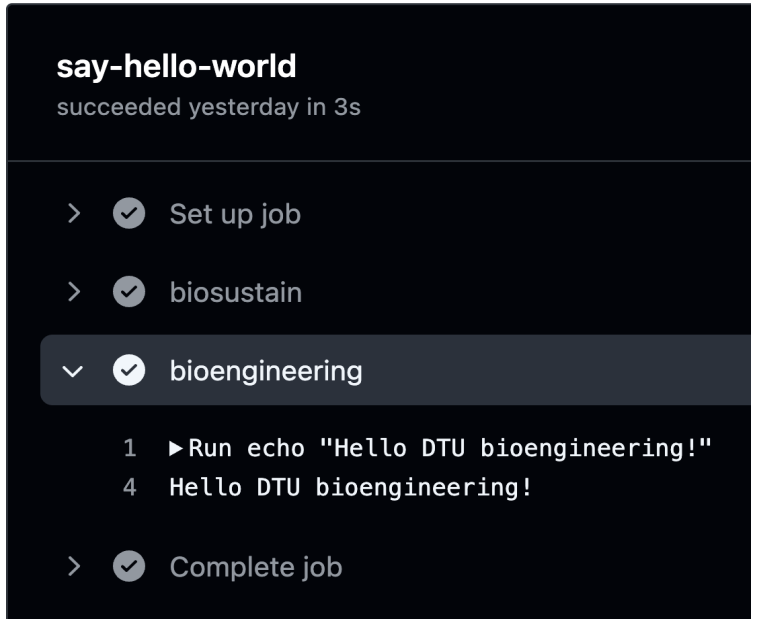

Hello World example
add exercises #13
Summary
Jobs
say-hello-world
Run details
Usage
Workflow file
say-hello-world
succeeded yesterday in 3s
Set up job
biosustain
bioengineering
1 ▶Run echo "Hello DTU bioengineering!"
4 Hello DTU bioengineering!
Complete job

What can or cannot see in run summary?

.github > workflows > hello_world.yml

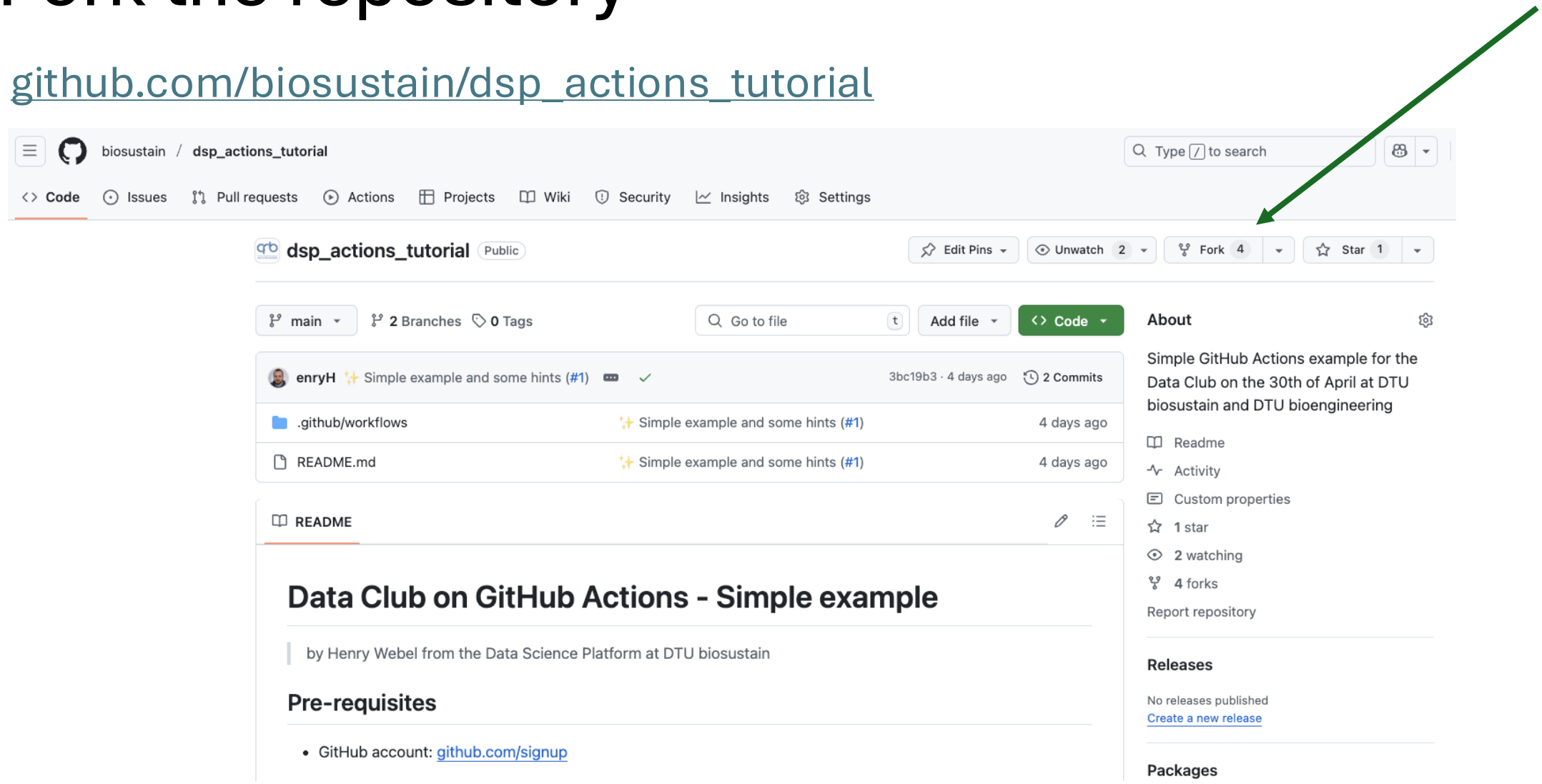
```yaml
# Hello World example for DTU biosustain and DTU bioengineering Data Club

name: Hello World example

on:
  push:

jobs:
  say-hello-world:
    runs-on: ubuntu-latest
    steps:
    - name: biosustain
      run: echo "Hello DTU biosustain!"
    - name: bioengineering
      run: echo "Hello DTU bioengineering!"
```



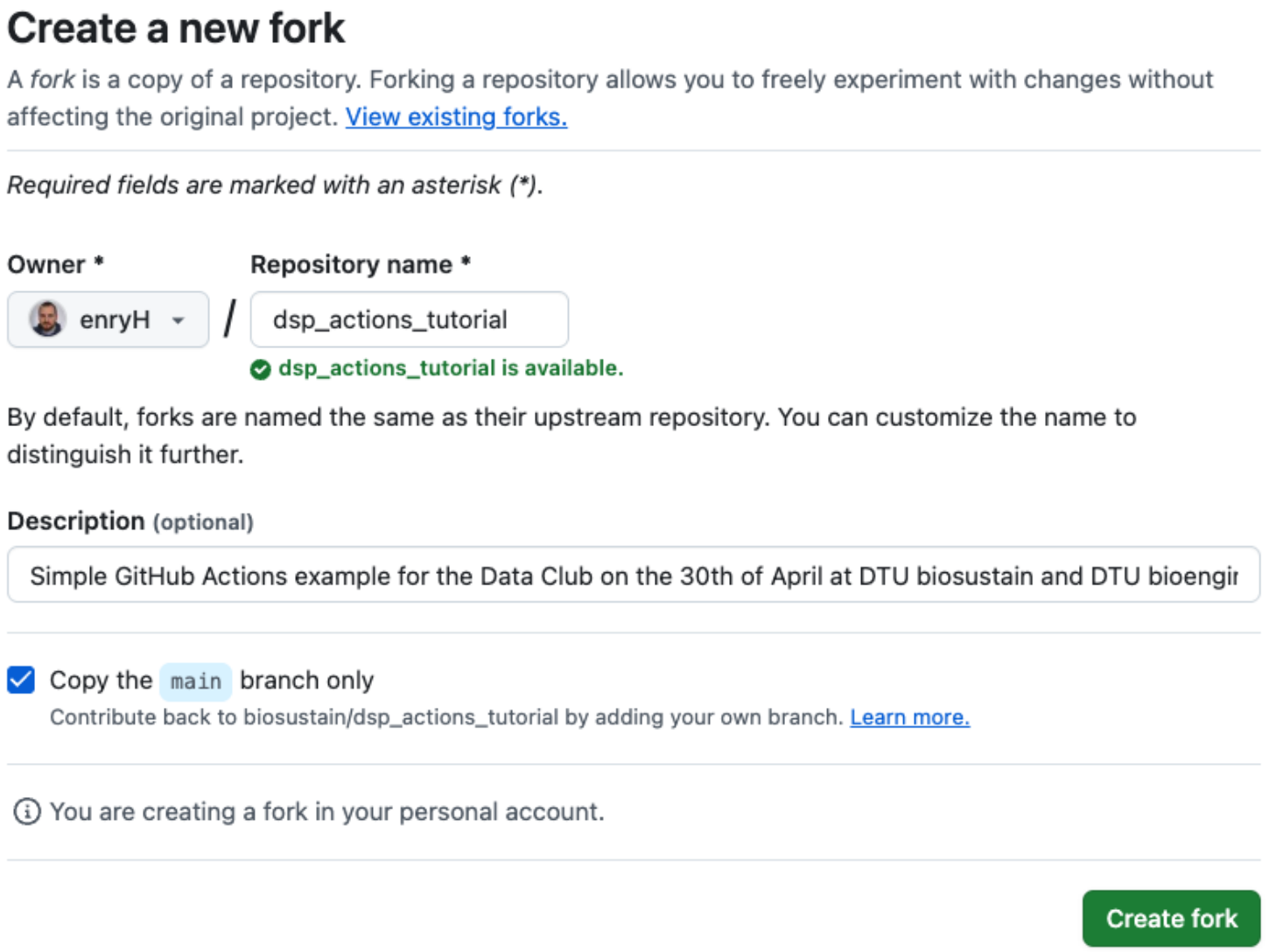


Fork the repository

github.com/biosustain/dsp_actions_tutorial

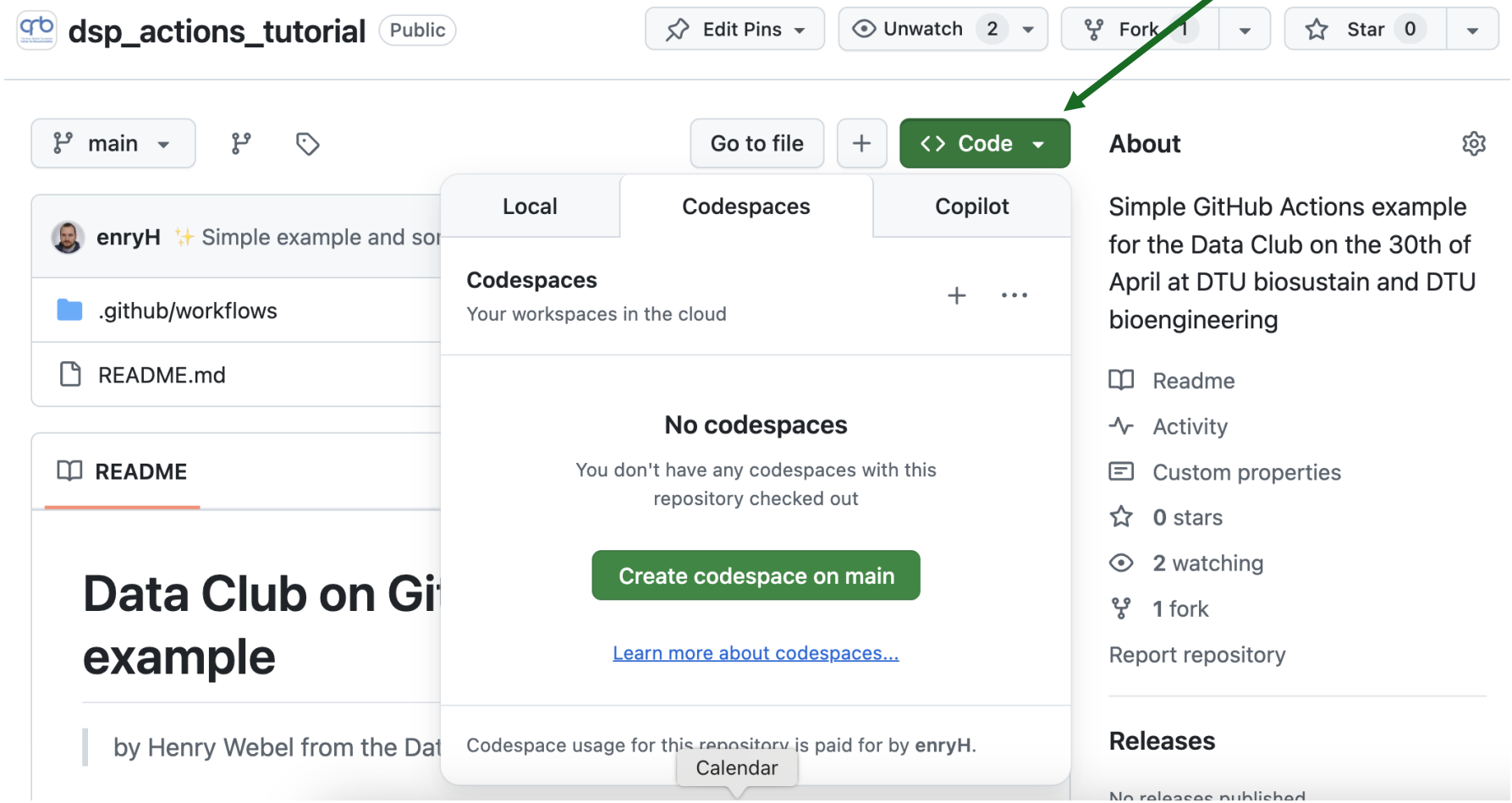


Create fork


Create a new fork
A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. View existing forks.
Required fields are marked with an asterisk (*).
Owner *
Repository name *
enryH
/
dsp_actions_tutorial
dsp_actions_tutorial is available.
By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.
Description (optional)
Simple GitHub Actions example for the Data Club on the 30th of April at DTU biosustain and DTU bioengir
Copy the main branch only
Contribute back to biosustain/dsp_actions_tutorial by adding your own branch. Learn more.
You are creating a fork in your personal account.
Create fork

Hands-on: Setup Code-Space on **your fork**

https://github.com/biosustain/dsp_actions_tutorial



Exercises

Say yes if you see the pop-Up: Install GitHub Actions Extension?
- also to git fetch etc.

```
# first job
## install step:
pip install cowsay
## use it
cowsay -c cheese -t  'hey there’

# second job
## https://github.com/jeffbuttars/cowpy
pip install cowpy

cowpy -c tux hi there
```

- *First*: Add **two jobs** which use either cowsay or cowpy
  - Separate installation and usage into separate **steps**
- *Medium*: Call it from a script (to understand file structure)
  - You will need the files!
  - Copy examples from cowpy README.md
- *Advanced*: Build a directed acyclic graph combining jobs
  - Look at the extended examples in the README to figure this out

One Solution

- Add bin/cowpy_examples.py to repository
  - Need to check repository out
- Does it also run on windows or Mac?
- Caveat: Shell behaviour of (Python) libraries: see cowpy step for command line execution. Check default shell configuration on runners here and warnings here

```yaml
name: Hello World example

on:
  push:
  pull_request:

jobs:
  say-hello-world: ⋯
  cowsay-test:
    runs-on: ubuntu-latest
    steps:
    - name: install cowsay
      run: |
        pip install cowsay
    - name: run cowsay
      run: cowsay -c cheese -t  'hey there'
  cowpy-test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - name: install cowpy
      run: |
        pip install cowpy
    - name: run cowpy
      run: |
       cowpy -c tux "hola mundo"
       echo "hola mundo" | cowpy  -c tux
    - name: run script with examples
      run: |
        python bin/cowpy_examples.py
```

Predefined actions

- Predefined action contain a collection of steps and have arguments (‘with’)
- Most common two:
  - Checkout to get the repository files
  - Setup-python to install a specific Python version for testing

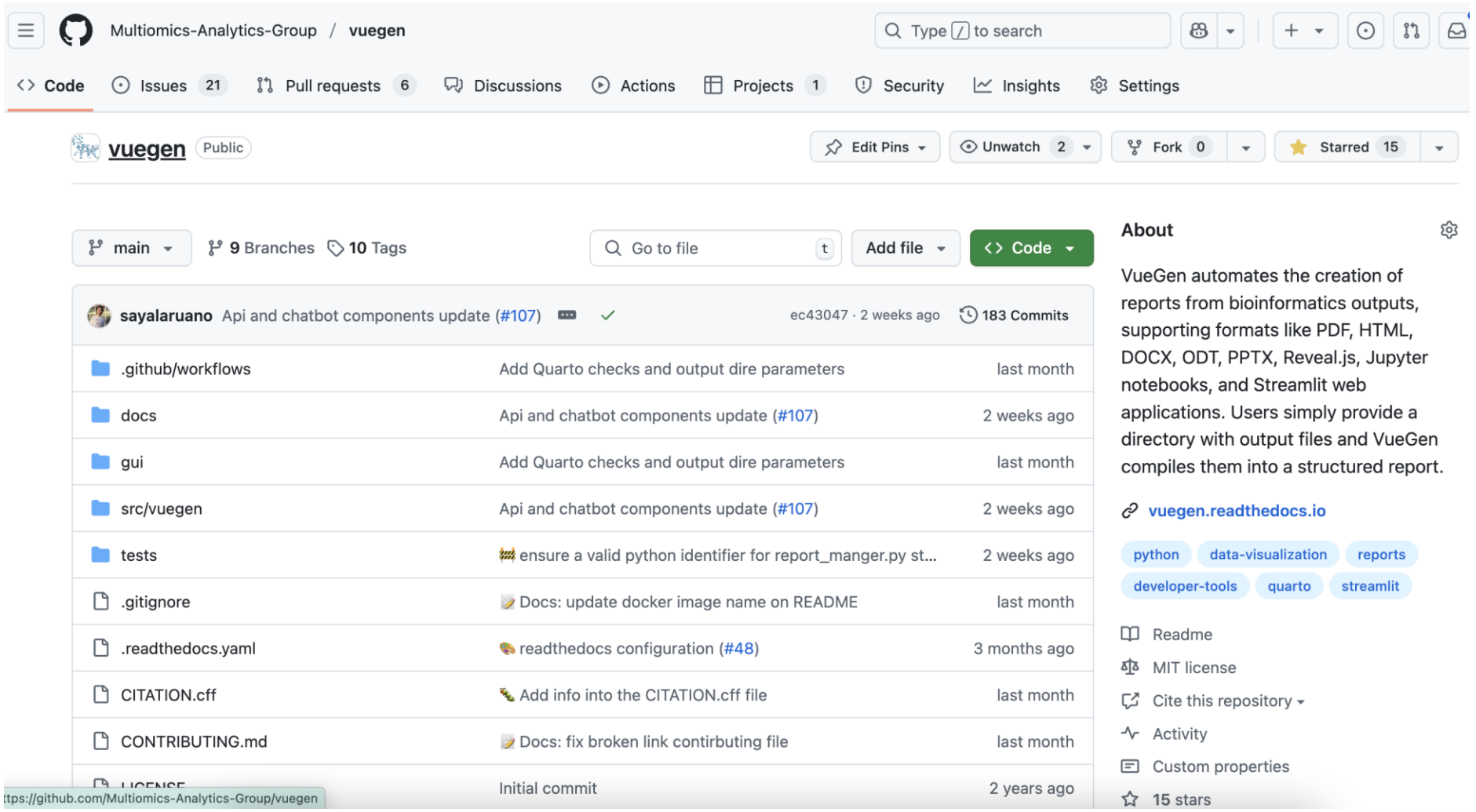
```yaml
cowpy-test:
  runs-on: ubuntu-latest
  defaults:
    run:
      shell: bash -el {0}
  steps:
  - uses: actions/checkout@v4
  - uses: actions/setup-python@v5
    with:
      python-version: '3.12'
  - name: install cowpy
    run: |
      pip install cowpy
  - name: run cowpy
    run: |
     cowpy -c tux hola mundo
     echo "hola mundo" | cowpy  -c tux
  - name: run script with examples
    run: |
      python bin/cowpy_examples.py
```

Program overview

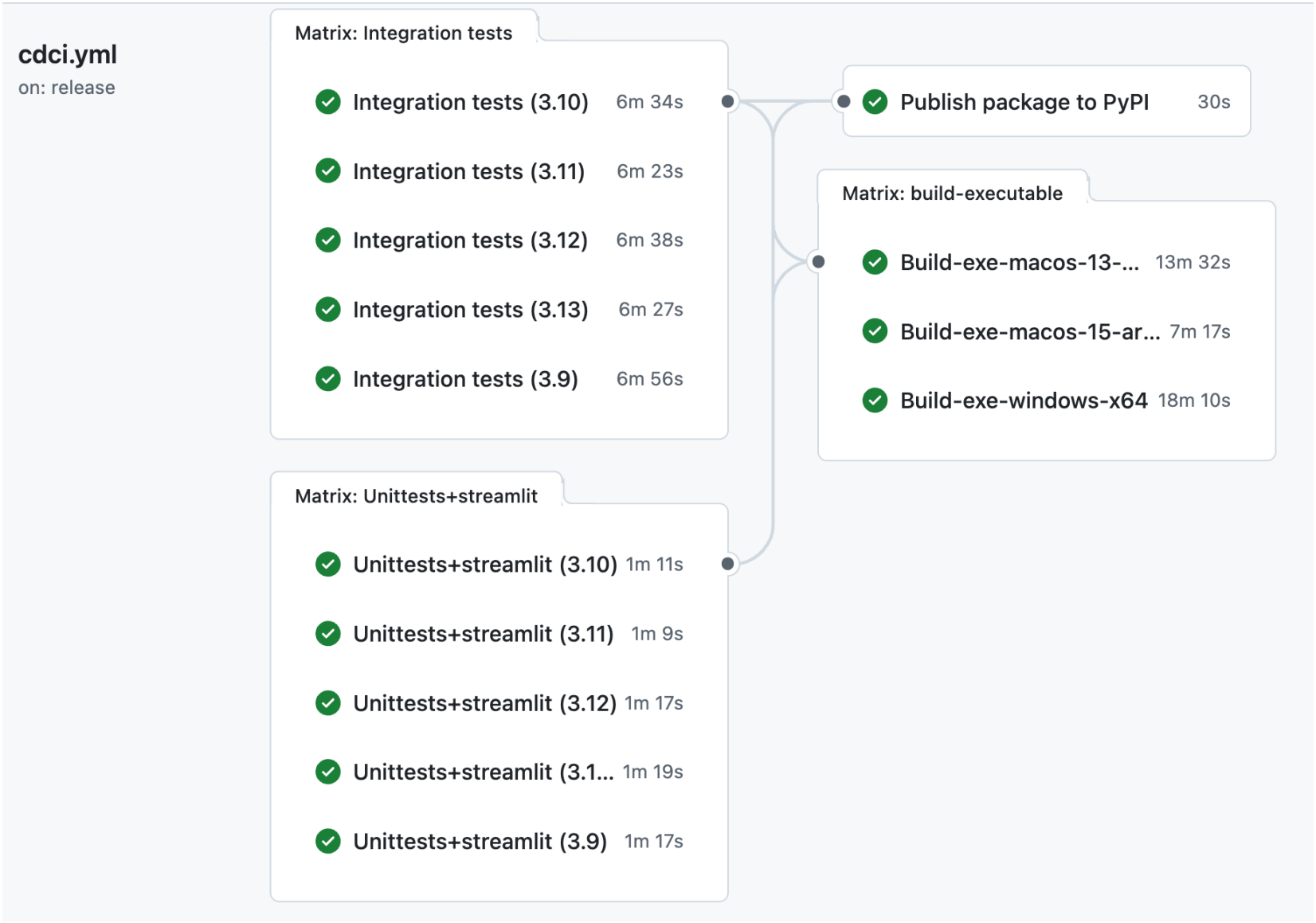
1. Introduction to GitHub Actions
2. First example
3. Yaml format (optional)
4. Workflow file in detail
5. *Vuegen repo: extended example (if time allows)*

VueGen

github.com/Multiomics-Analytics-Group/vuegen



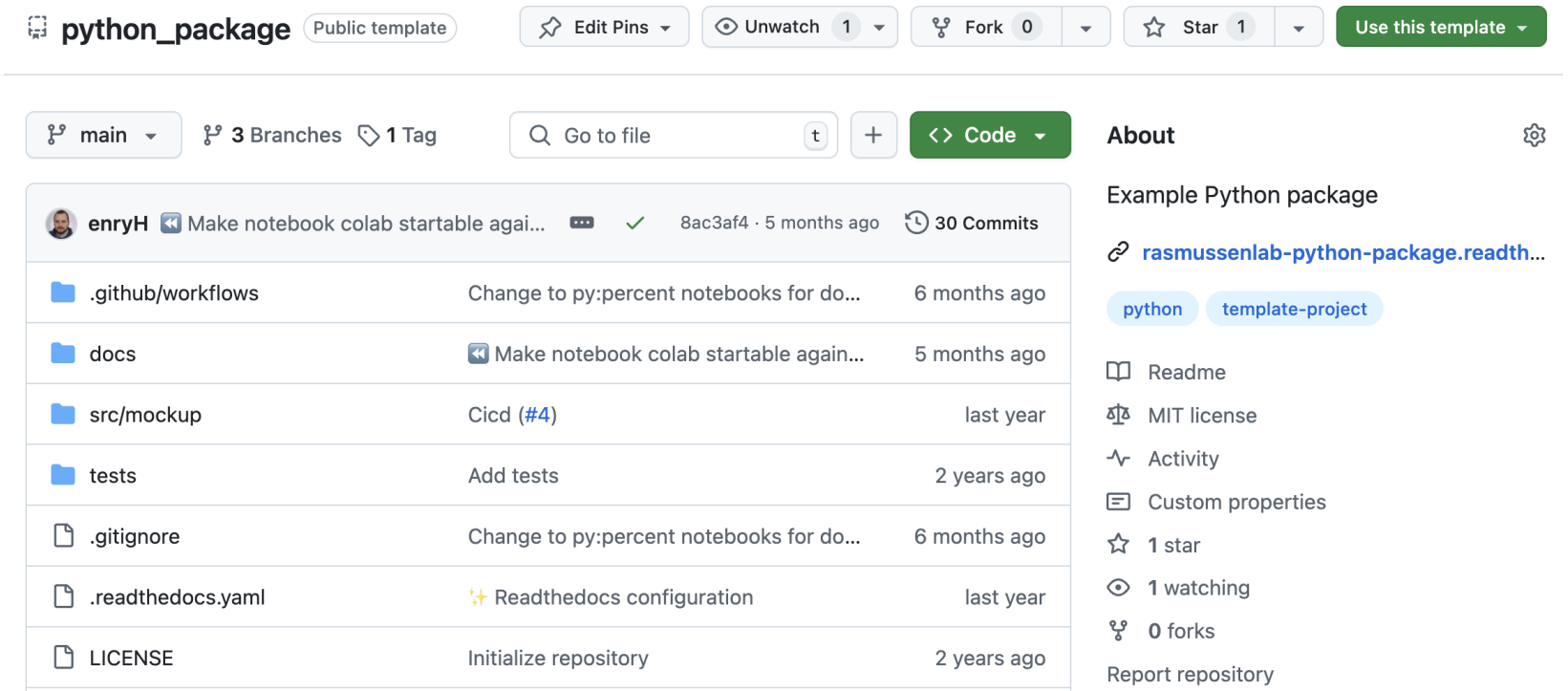
Vuegen release 0.3.1



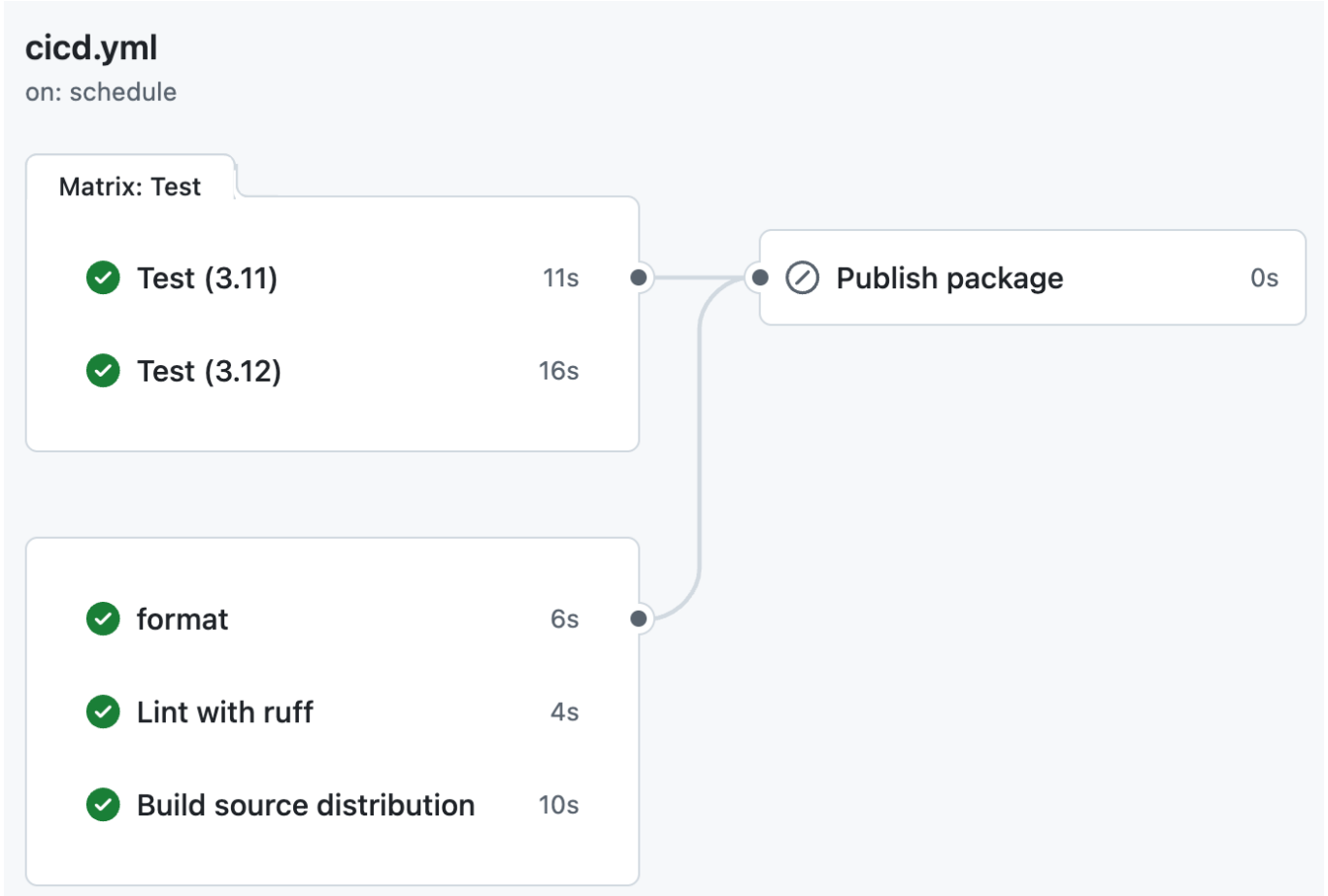
Python package template repository

- Python project example
- Lint and format

github.com/RasmussenLab/python_package



First example



Acknowledgments and Hints

- Pieter Verschaffelt, Workshop on EuBIC Winterschool 2024
  - New Winterschool in 2026 in Czech!
- Check out VueGen!
- Remember to delete your codespaces: github.com/codespaces

More material:

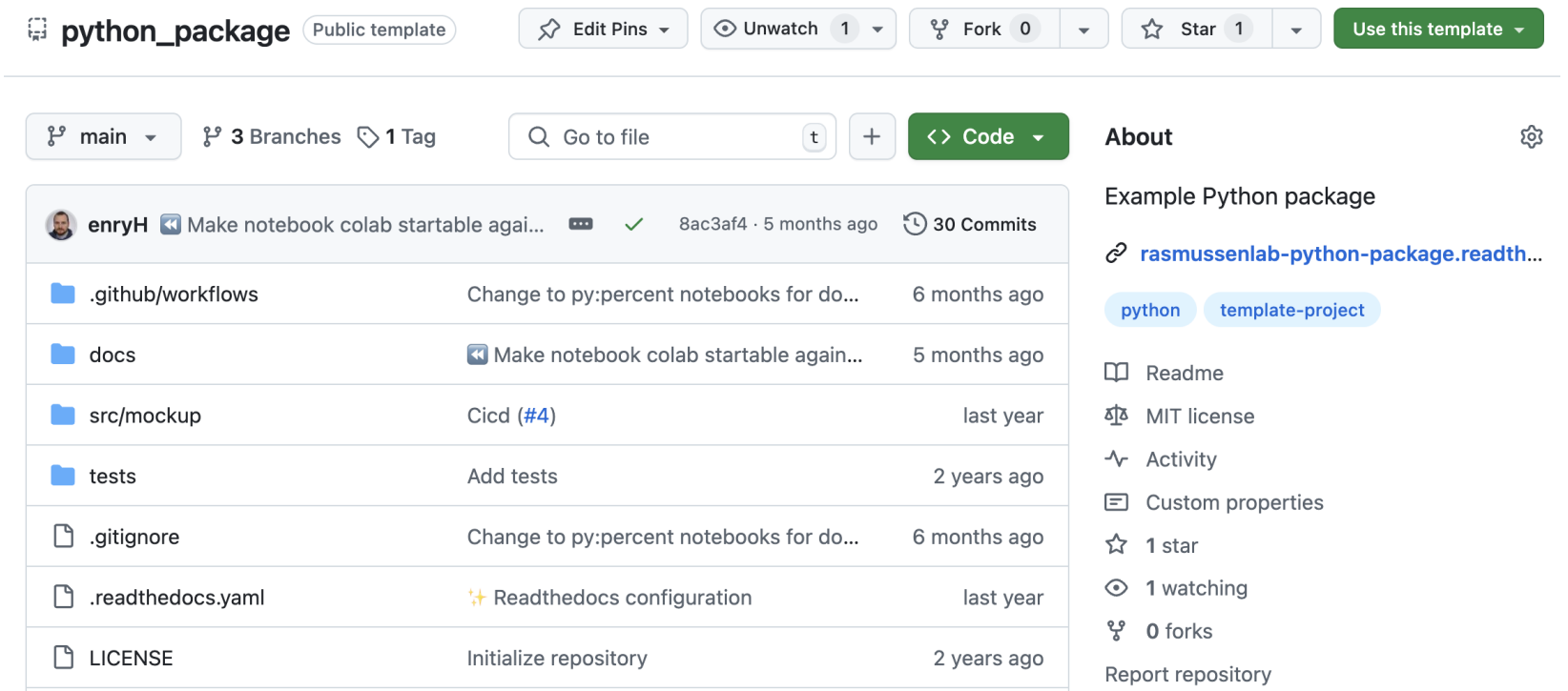
- https://book.the-turing-way.org/reproducible-research/ci

The End – Happy coding!

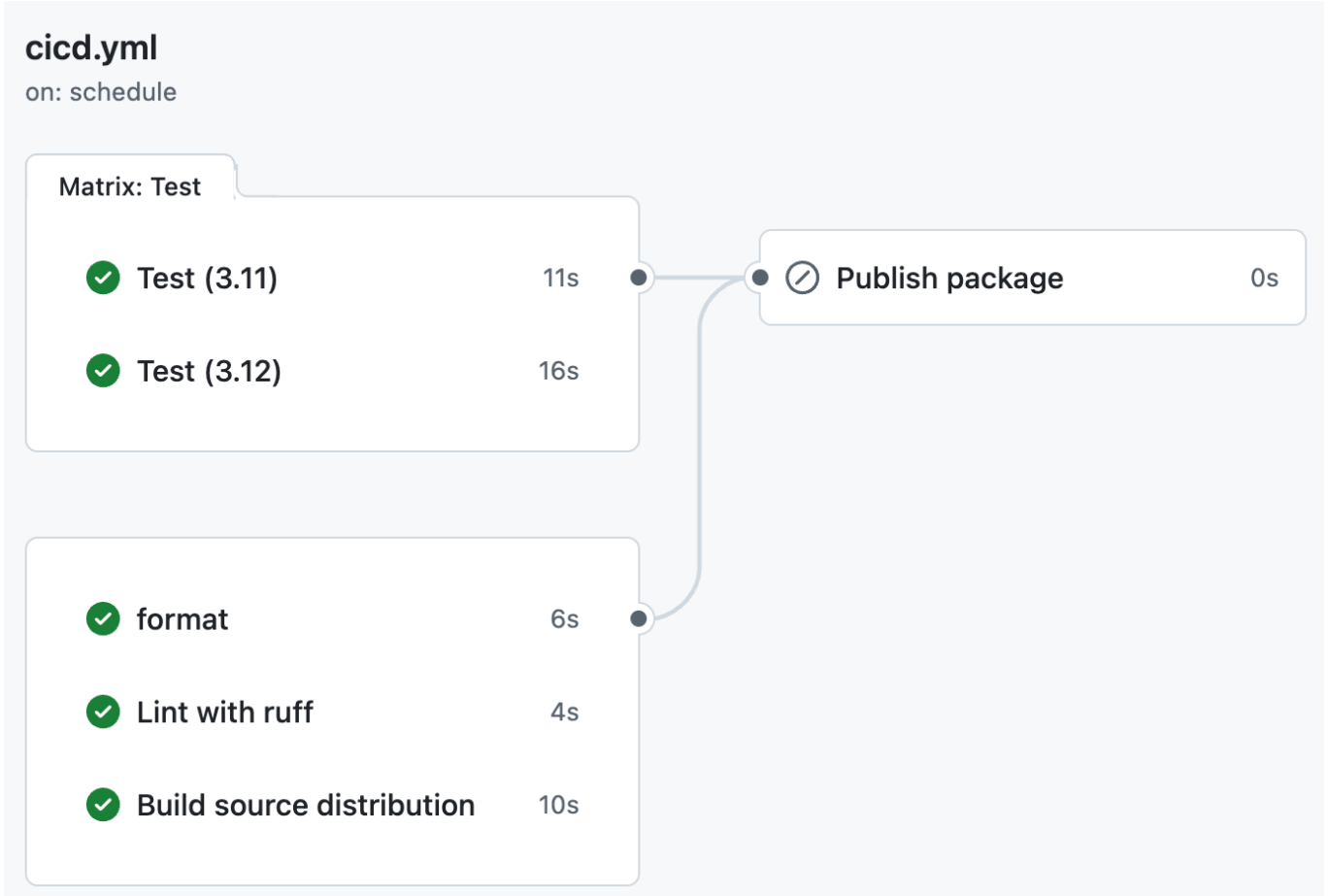
Python package template repository

- Python project example
- Lint and format

RasmussenLab/python_package



CICD action


cicd.yml
on: schedule
Matrix: Test
Test (3.11)
11s
Test (3.12)
16s
format
6s
Lint with ruff
4s
Build source distribution
10s
Publish package
0s

Workflow in Detail

python_package > .github > workflows > cicd.yml

```yaml
# This workflow will install Python dependencies, run tests and lint with a single version of Python
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

name: Python application

on:
  push:
  pull_request:
    branches: [ "main" ]
  schedule:
    - cron: '0 2 * * 3'

permissions:
  contents: read


jobs:
  format:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: psf/black@stable
  lint:
    name: Lint with ruff
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-python@v5
        with:
          python-version: "3.11"
      - name: Install ruff
        run: |
          pip install ruff
      - name: Lint with ruff
        run: |
          # stop the build if there are Python syntax errors or undefined names
          ruff check .
```

Focus on jobs

python_package > .github > workflows > cicd.yml

```yaml
# This workflow will install Python dependencies, run tests and lint with a single version of Python
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

name: Python application

on: ...

permissions: ...

jobs:
  format:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: psf/black@stable
  lint:
    name: Lint with ruff
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-python@v5
        with:
          python-version: "3.11"
      - name: Install ruff
        run: |
          pip install ruff
      - name: Lint with ruff
        run: |
          # stop the build if there are Python syntax errors or undefined names
          ruff check .
```

Workflow in Detail

python_package > .github > workflows > cicd.yml

```yaml
# This workflow will install Python dependencies, run tests and lint with a single version of Python
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

name: Python application

on: ⋯

permissions: ⋯

jobs:
  format:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: psf/black@stable
  lint:
    name: Lint with ruff
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-python@v5
        with:
          python-version: "3.11"
      - name: Install ruff
        run: |
          pip install ruff
      - name: Lint with ruff
        run: |
          # stop the build if there are Python syntax errors or undefined names
          ruff check .
```

Name: (required)
Describe your action in a few words. This will be used by GitHub and help you to easily find a specific action back.

Workflow in Detail

python_package > .github > workflows > cicd.yml

```yaml
# This workflow will install Python dependencies, run tests and lint with a single version of Python
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

name: Python application

on:
  push:
  pull_request:
    branches: [ "main" ]
  schedule:
    - cron: '0 2 * * 3'

permissions:
  contents: read


jobs:
  format:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: psf/black@stable
  lint:
    name: Lint with ruff
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-python@v5
        with:
          python-version: "3.11"
      - name: Install ruff
        run: |
          pip install ruff
      - name: Lint with ruff
        run: |
          # stop the build if there are Python syntax errors or undefined names
          ruff check .
```

Triggers: (required)
After which event should this action be executed? Multiple triggers can be provided simultaneously.

Workflow in Detail

python_package > .github > workflows > cicd.yml

```yaml
# This workflow will install Python dependencies, run tests and lint with a single version of Python
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

name: Python application

on: ⋯

permissions: ⋯

jobs:
  format:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: psf/black@stable
  lint:
    name: Lint with ruff
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-python@v5
        with:
          python-version: "3.11"
      - name: Install ruff
        run: |
          pip install ruff
      - name: Lint with ruff
        run: |
          # stop the build if there are Python syntax errors or undefined names
          ruff check .
```

Jobs: (required)
A list of actions that should be executed by this workflow. At least one action is required!

Workflow in Detail

python_package > .github > workflows > cicd.yml

```yaml
# This workflow will install Python dependencies, run tests and lint with a single version of Python
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

name: Python application

on: ⋯

permissions: ⋯

jobs:
  format:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: psf/black@stable
  lint:
    name: Lint with ruff
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-python@v5
        with:
          python-version: "3.11"
      - name: Install ruff
        run: |
          pip install ruff
      - name: Lint with ruff
        run: |
          # stop the build if there are Python syntax errors or undefined names
          ruff check .
```

Job name

Workflow in Detail

python_package > .github > workflows > cicd.yml

```yaml
# This workflow will install Python dependencies, run tests and lint with a single version of Python
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

name: Python application

on: ⋯

permissions: ⋯

jobs:
  format:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: psf/black@stable
  lint:
    name: Lint with ruff
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-python@v5
        with:
          python-version: "3.11"
      - name: Install ruff
        run: |
          pip install ruff
      - name: Lint with ruff
        run: |
          # stop the build if there are Python syntax errors or undefined names
          ruff check .
```

Runner: (required)
Which operating system should this action be executed on? (e.g. ubuntu, windows, macos, ...)

Standard GitHub-hosted runners

Virtual Machine	Processor (CPU)	Memory (RAM)	Storage (SSD)	Architecture	Workflow label
Linux	4	16 GB	14 GB	x64	`ubuntu-latest`, `ubuntu-24.04`, `ubuntu-22.04`, `ubuntu-20.04`
Windows	4	16 GB	14 GB	x64	`windows-latest`, `windows-2025` [Public preview], `windows-2022`, `windows-2019`
Linux [Public preview]	4	16 GB	14 GB	arm64	`ubuntu-24.04-arm`, `ubuntu-22.04-arm`
macOS	4	14 GB	14 GB	Intel	`macos-13`
macOS	3 (M1)	7 GB	14 GB	arm64	`macos-latest`, `macos-14`, `macos-15` [Public preview]

about GitHub hosted runners

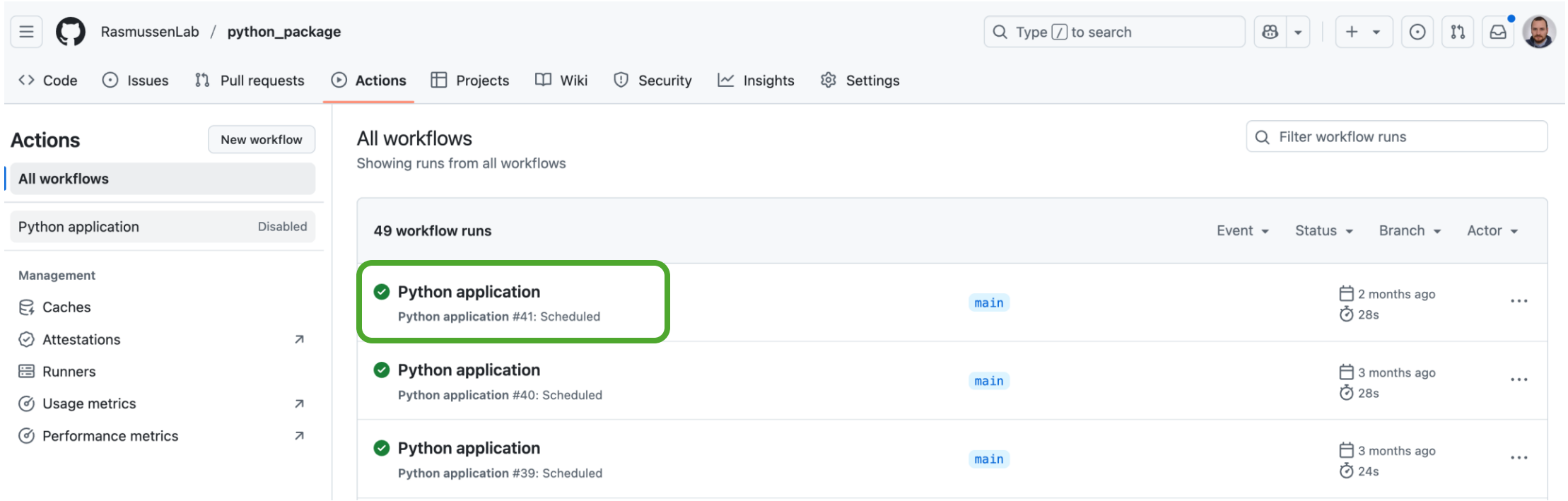
Workflow in Detail

python_package > .github > workflows > cicd.yml

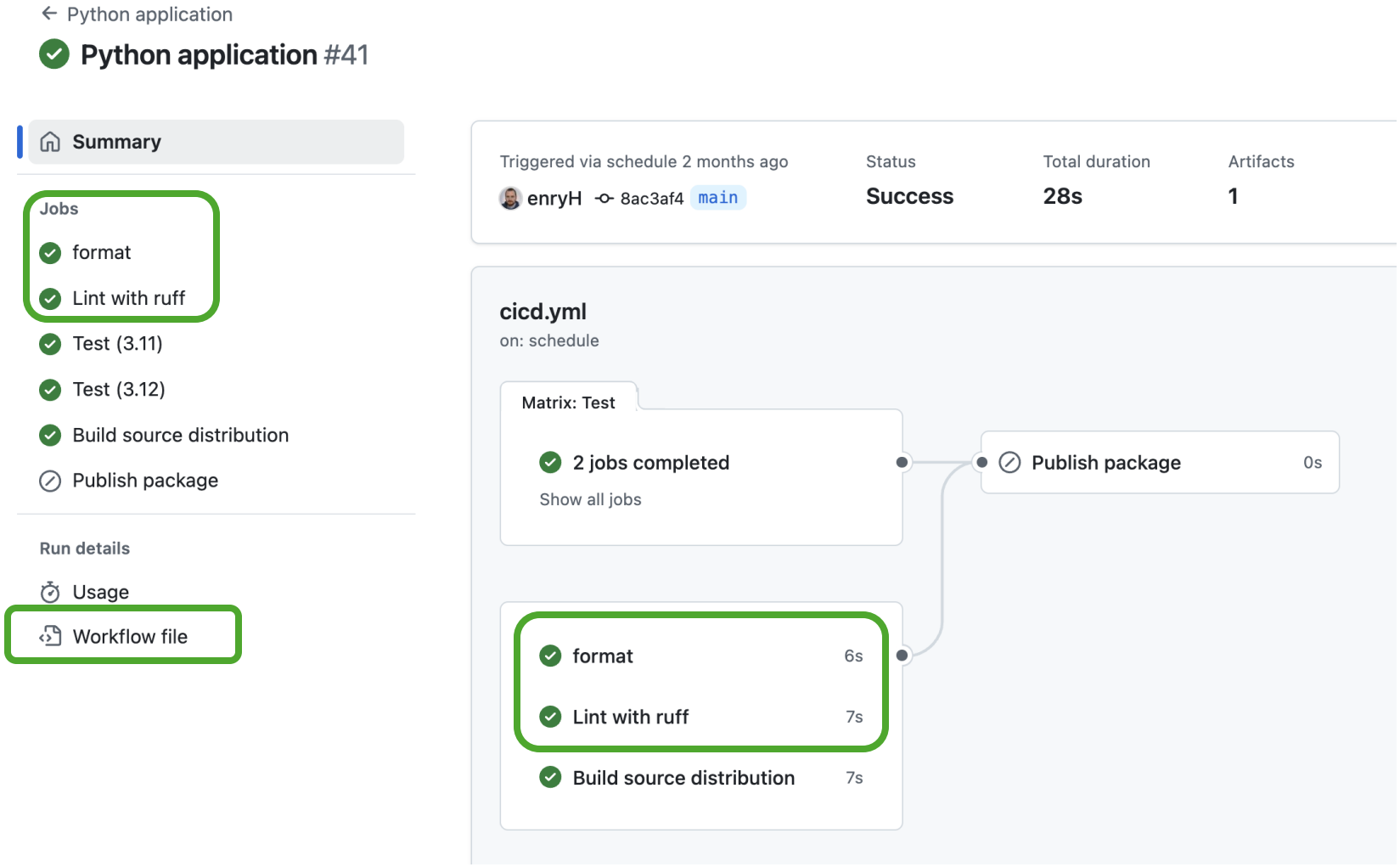
```yaml
# This workflow will install Python dependencies, run tests and lint with a single version of Python
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

name: Python application

on: ⋯

permissions: ⋯

jobs:
  format:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: psf/black@stable
  lint:
    name: Lint with ruff
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-python@v5
        with:
          python-version: "3.11"
      - name: Install ruff
        run: |
          pip install ruff
      - name: Lint with ruff
        run: |
          # stop the build if there are Python syntax errors or undefined names
          ruff check .
```

Steps: (required)
What does the action need to do? Steps are executed one-by-one. If a step fails, subsequent steps will not be executed.

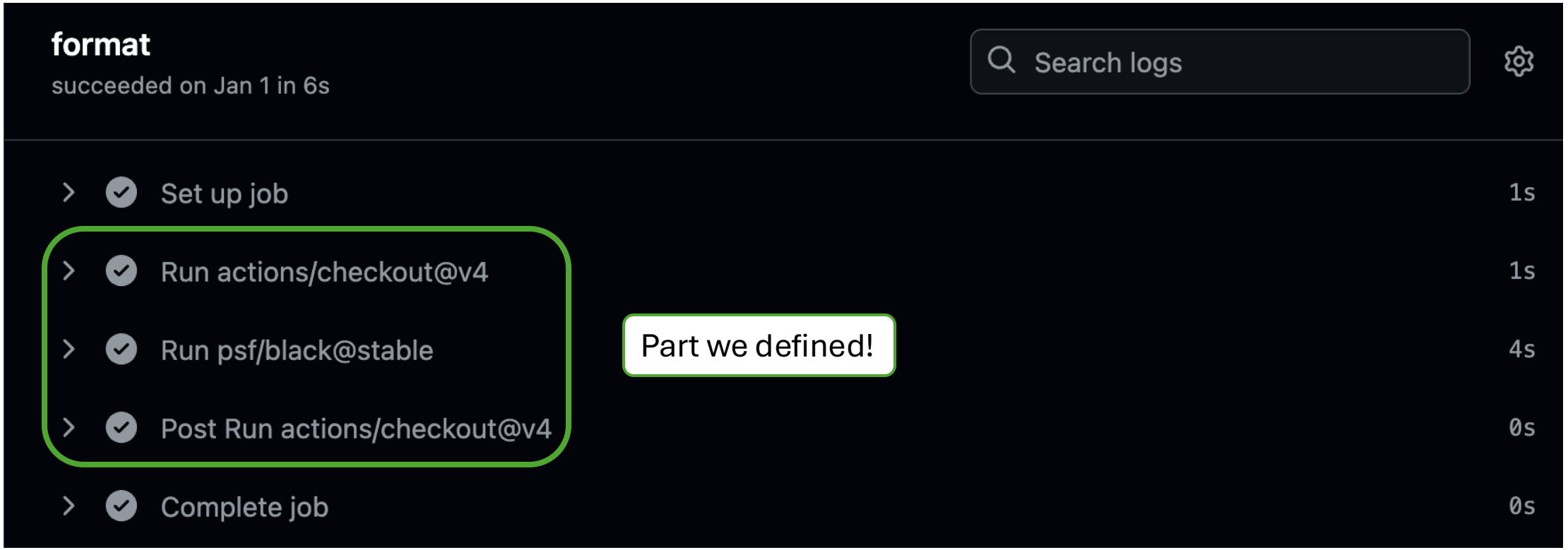
Run overview



GitHub Actions view



Format Action in detail



Workflow in Detail

python_package > .github > workflows > cicd.yml

```yaml
# This workflow will install Python dependencies, run tests and lint with a single version of Python
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

name: Python application

on: ⋯

permissions: ⋯

jobs:
  format:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: psf/black@stable
  lint:
    name: Lint with ruff
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-python@v5
        with:
          python-version: "3.11"
      - name: Install ruff
        run: |
          pip install ruff
      - name: Lint with ruff
        run: |
          # stop the build if there are Python syntax errors or undefined names
          ruff check .
```

Job name

Program overview

1. Introduction to GitHub Actions
2. First example
3. Yaml format (optional)
4. Workflow file in detail
5. PIMMS repo: add first example (if time allows)

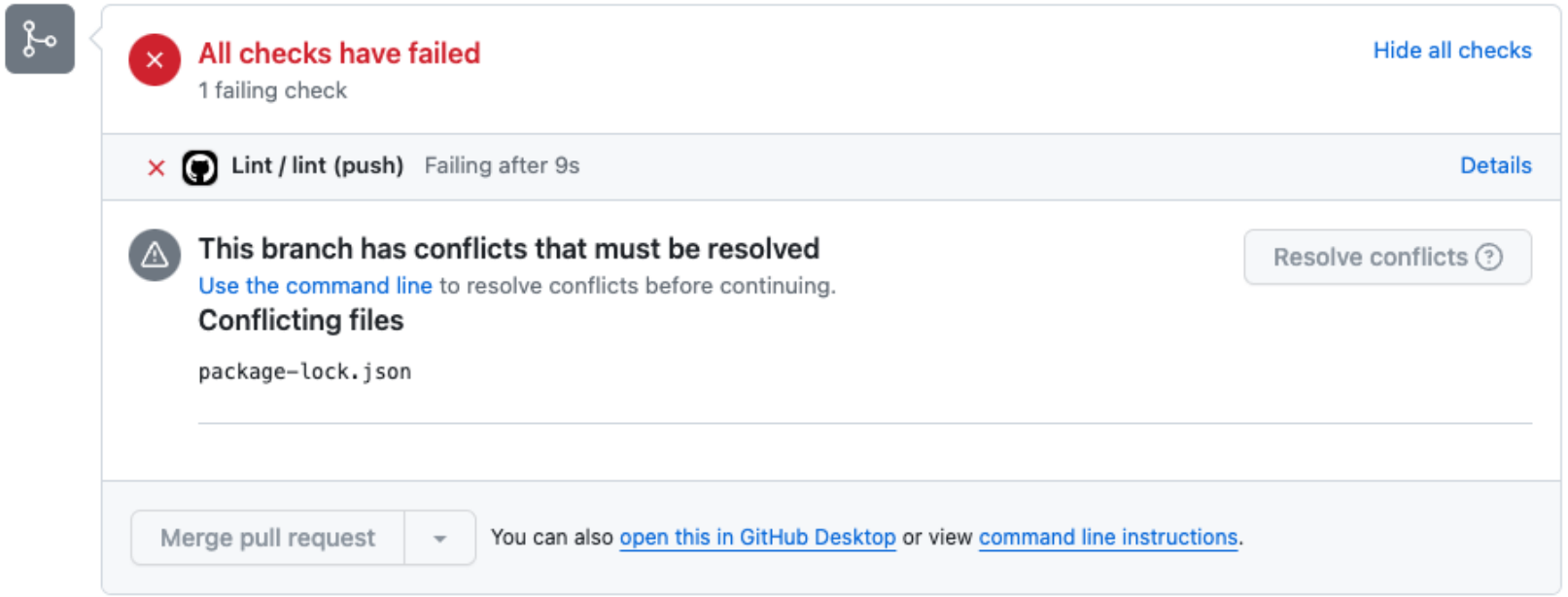
- RasmussenLab/pimms/blob/main/.github/workflows/ci.yaml
  - Run integration test
  - Test pip package works for core models in PIMMS
  - Can publish package to PyPI
- No code formatting checks or linter checks

Workflow triggers

- **Specific event that initiates a workflow**
- Can be categorised in different types
  - **Event-based:** Triggers workflows based on GitHub events
    - e.g. push, pull_request, issue, …
  - **Time-based:** Triggers workflows at a specific moment
    - e.g. each day / week / month / …
  - **Manual:** Triggers workflows after you click a button
  - …
- Can be combined!

Event-based triggers

- push, pull_request, issue, …
  - Workflows are automatically executed after push to repository, creation of new pull request, creation of new issue, …
  - Typically used to perform unit tests or code linting
  - Pull requests **can be blocked** if not all checks pass

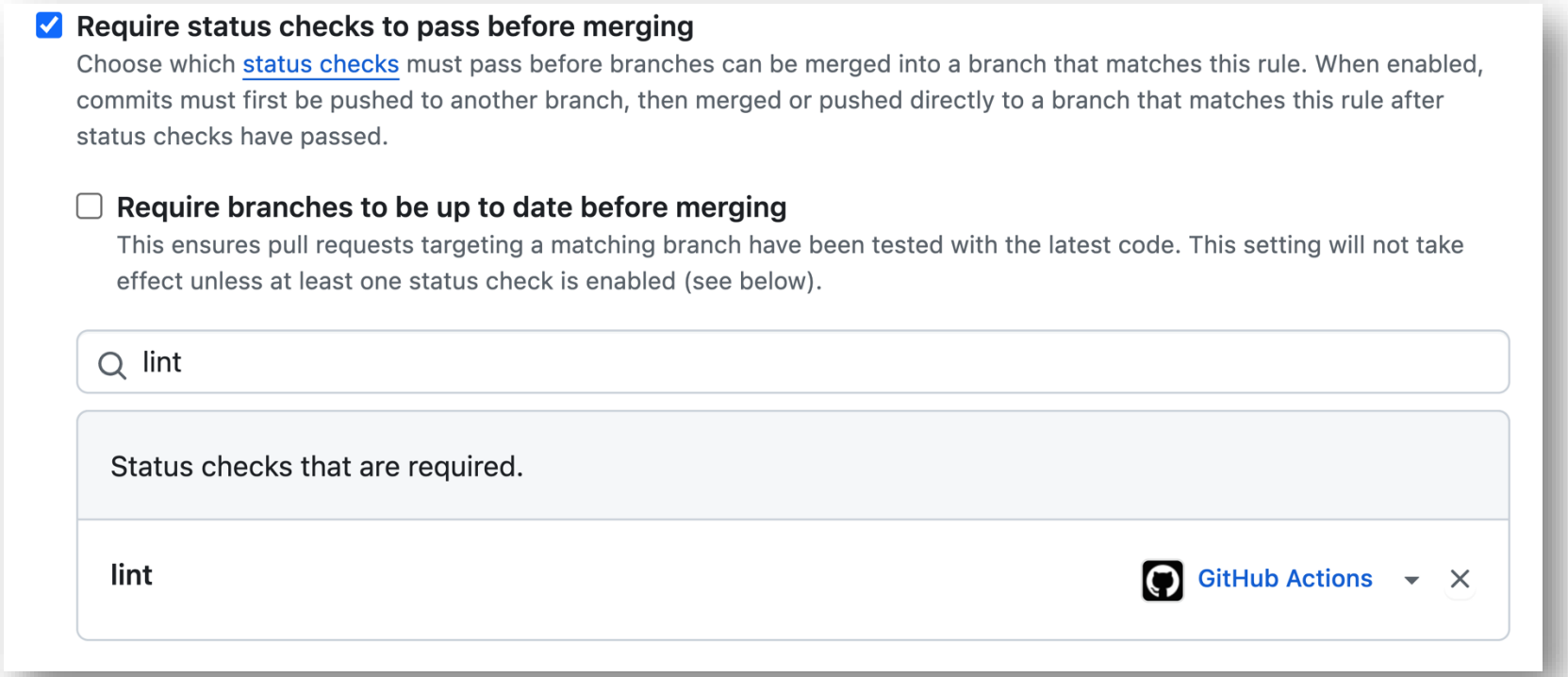


Block merging of pull requests

- Actions can be used to block pull requests if some checks fail
  - e.g. Failing unit tests, linting, automated code quality assessments
- Action is marked as "failed" when one of the steps return non-zero exit code
- Merging can be blocked if one or more actions failed

Block merging of pull requests

- Go to “Settings” > “Branches” > “Add protection rule”
  - Check “Require status checks to pass before merging”
  - Select one or more actions that need to successfully pass



Time-based triggers

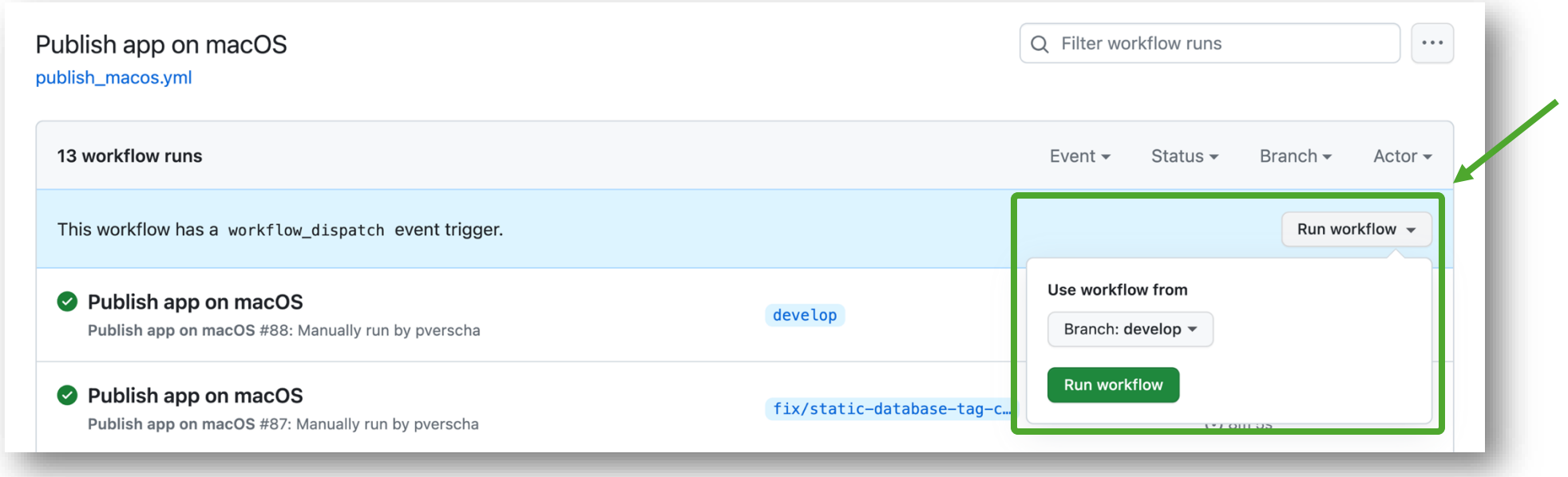
- Automatically triggers the workflow at a specific point in time
- Requires the `schedule` keyword and a cron-compatible expression

```yaml
name: Monthly status report

on:
  schedule:
    # * is a special character in YAML so you have to quote this string
    - cron:  '0 0 1 * *'
```

Tip: use https://crontab.guru/ to easily construct cron-compatible expressions

Manual triggers

- Add a button to GitHub's interface
- Run the action for a specific branch of the repository
- Simply add `workflow_dispatch` as extra trigger to the workflow
- E.g. manually test code from a specific branch



Manual triggers

- Can even be configured to ask user for input

```yaml
name: Deploy Specific Version

on:
  workflow_dispatch:
    inputs:
      version:
        description: 'Version tag to deploy'
        required: true

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout Repository
      uses: actions/checkout@v2
      with:
        ref: ${{ github.event.inputs.version }}

    - name: Deploy
      run: |
        echo "Deploying version ${{ github.event.inputs.version }}"
        # Add deployment scripts or commands here
```

Premade actions can be reused

- Similar actions typically require the same initial steps
- Big actions can be split up into smaller actions
  - Identical steps can be reused by multiple actions
- GitHub Marketplace provides extensive library of reusable actions
  - https://github.com/marketplace?type=actions

Revisiting our code linting example

- Instead of manually installing Python, we can look for premade actions that already do this

- Employ `uses` key when defining step with reusable action

```yaml
- name: Install Python 3.8
  run: |
    sudo apt update
    sudo apt install python3-pip
```

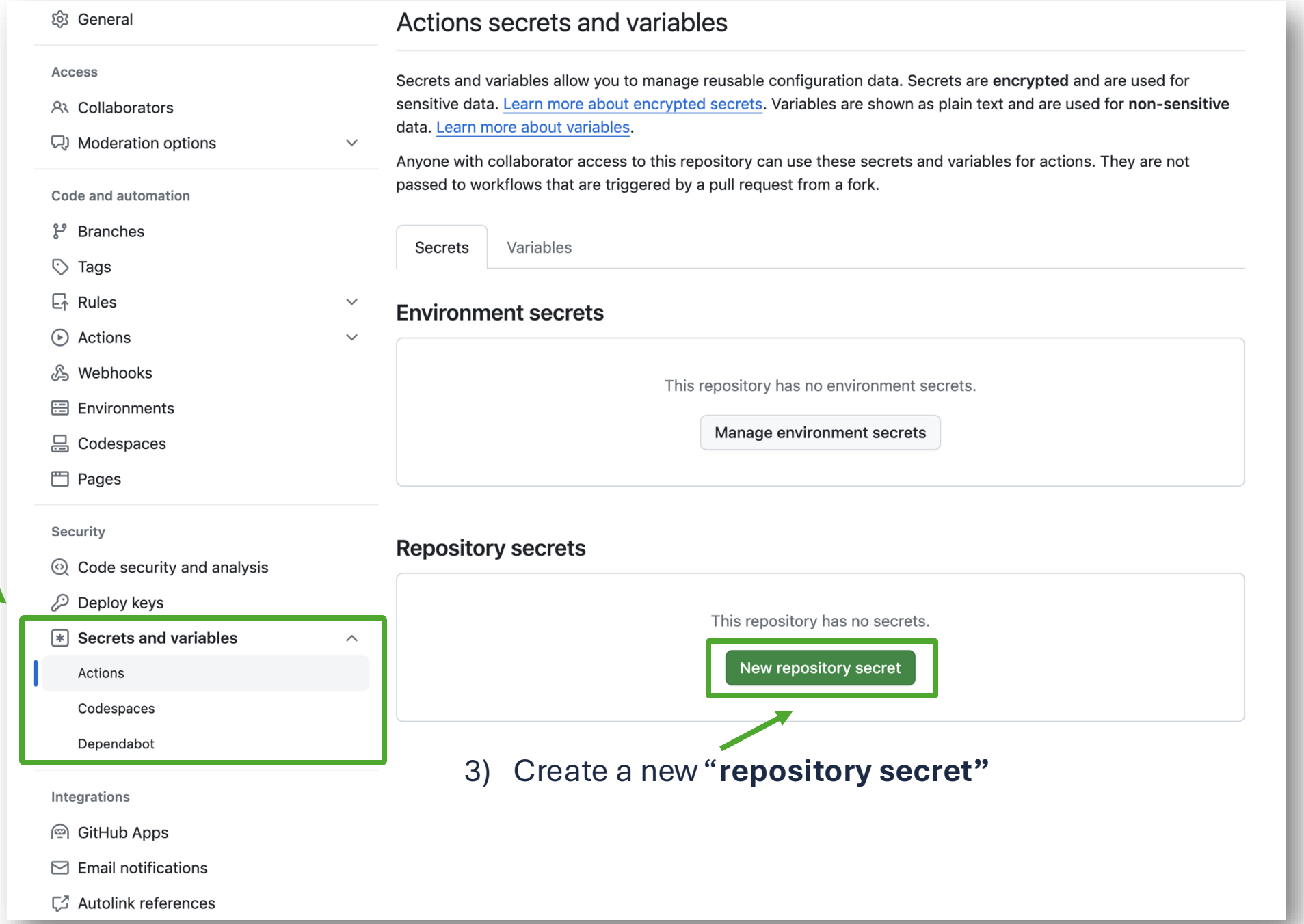
```yaml
- name: Install Python
  uses: actions/setup-python@v2
```

Secrets

- API keys or other “secrets” cannot be added to a repository
  - Effectively exposes the secret to everyone with access to the repository
- Some actions require knowledge about these keys to function
- GitHub allows you to securely store secrets and reuse these in workflows

1) Go to **repository settings**

2) Click on “**Actions**” under “**Secrets and variables**”



3) Create a new “**repository secret**”

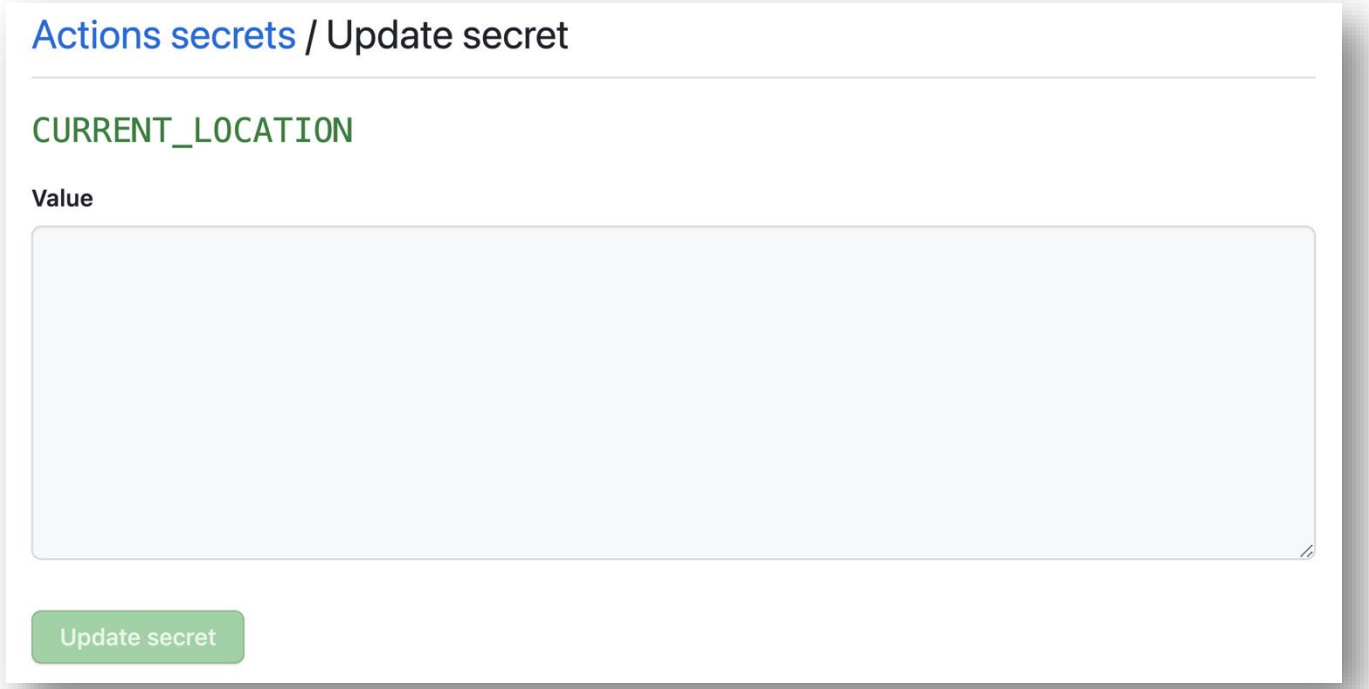
Secrets

- Can be referenced in workflows using `${{ secrets.MY_SECRET }}`

Secrets

- Can only be read by workflows in your repository
  - Is not even visible to you

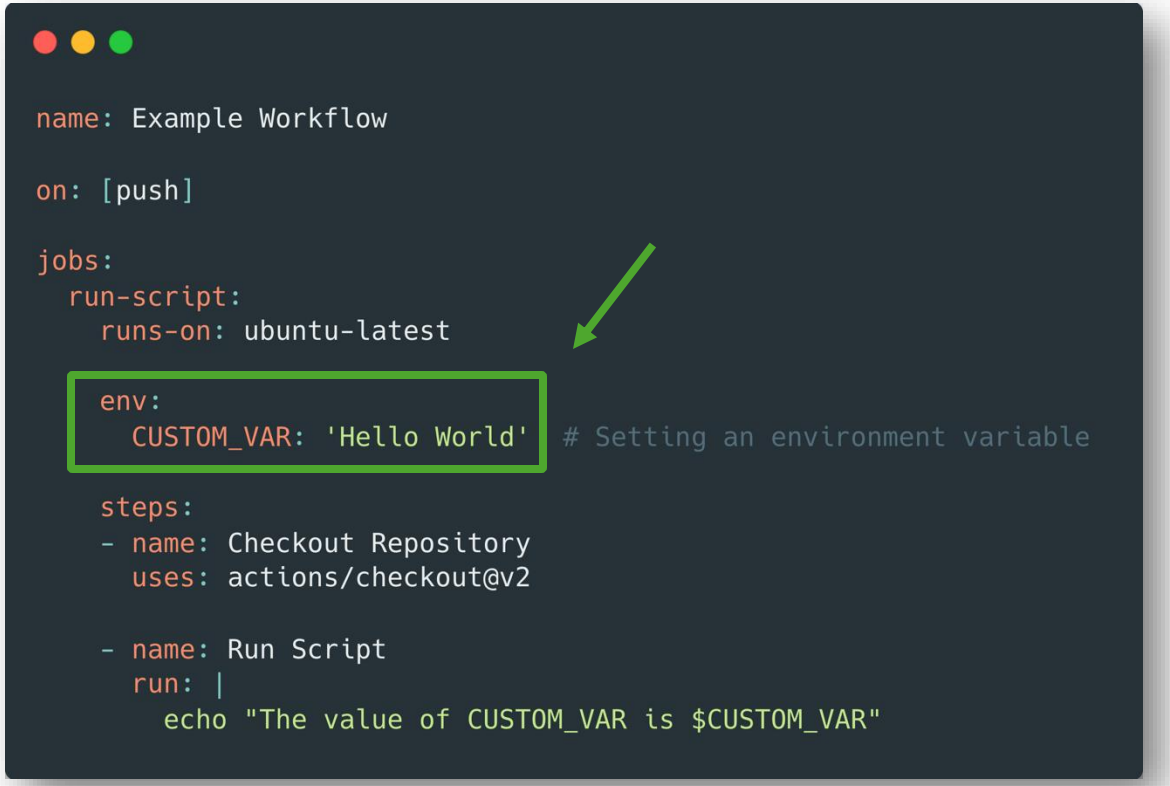
- Value can be updated, but previous values are invisible!



Environment variables

- Small pieces of data that define how a program or script behaves

- Are typically used for paths or settings related to the application
  - E.g. where are configuration files stored?
  - E.g. which URL should be used to request specific pieces of information from?



Continuous Integration (CI)

- The practice of frequently integrating code changes into a shared repository. Ideally, developers will integrate their changes daily, if not multiple times a day.

- Each integration is automatically tested to detect errors as quickly as possible

- GitHub Actions are an ideal candidate for this!

Continuous Development (CD)

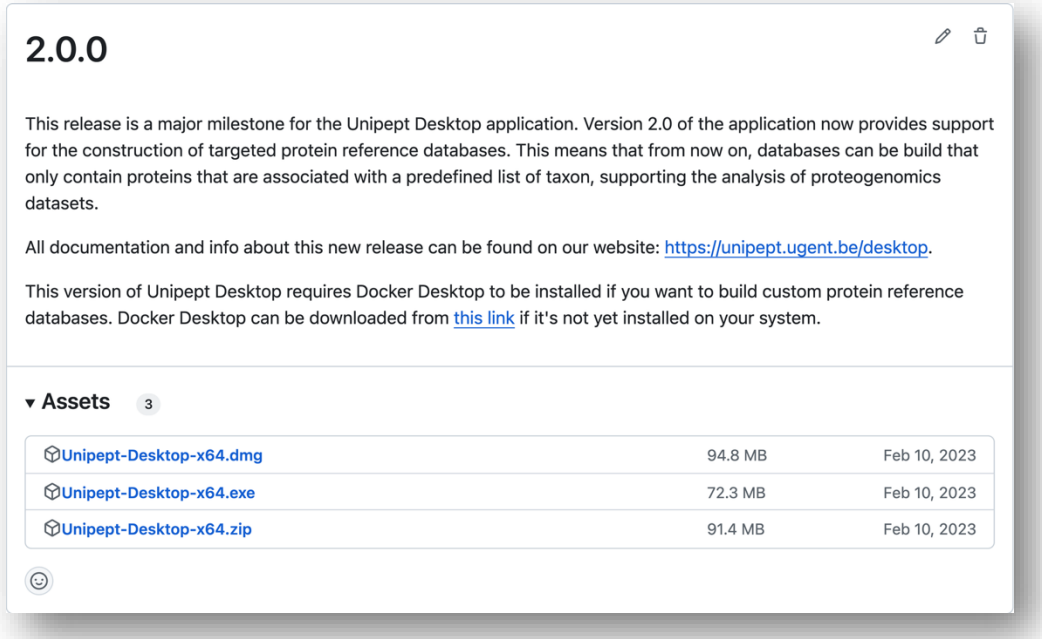
- The practice of continuously developing, testing, integrating, and deploying software

- Release early, release often
  - New features are released as soon as they are ready
  - User feedback can be gathered soon
  - Adjustments can be applied early in the development process

Other world examples (1)

Automatically build new version of Unipept Desktop application

- A different installer is required for each operating system
- Accompanying GitHub release needs to be created with all installers
  - https://github.com/unipept/unipept-desktop/tree/develop/.github/workflows



Other world examples (2)

Reconstruct new taxon database monthly

- New NCBI taxa are added every month
- Our application requires all metadata for these taxa
- A GitHub action automatically parses the new taxa and generates a database file
  - https://github.com/unipept/unipept-database/blob/master/.github/workflows/static_database.yml

Optimizing workflow performance

- Complicated GitHub actions can take a long time to execute

- Here are some tips and tricks to speed up workflow performance
  - Less GitHub credits wasted
  - Less time waiting for checks to pass
  - More convenient in some situations

Cache installed dependencies

- Installing dependencies takes a long time
- Reusing dependencies skips a big step in the workflow process
- Use GitHub Cache Action
  - https://github.com/actions/cache
  - Checks if a cached environment is available
  - If present, check if cache is still valid and reuses previously installed deps
  - If not, installs dependencies and creates new cache entry

```yaml
steps:
- uses: actions/checkout@v4
- uses: actions/setup-python@v5
  with:
    python-version: '3.9'
    cache: 'pip' # caching pip dependencies
- run: pip install -r requirements.txt
```

Test actions locally with `act`

- Updating and testing actions:
  - Make local change, commit change, wait and check for action status
  - Can take a long time
  - Tracking down persistent bugs can be hard
  - Tedious process
- Running GitHub actions locally is possible with `act`
  - Creates a new Docker container reflecting the specified workflow runner
  - https://github.com/nektos/act
  - Has some limitations, but can be very handy