

# Outiller son projet open-source Python avec les Github actions

---

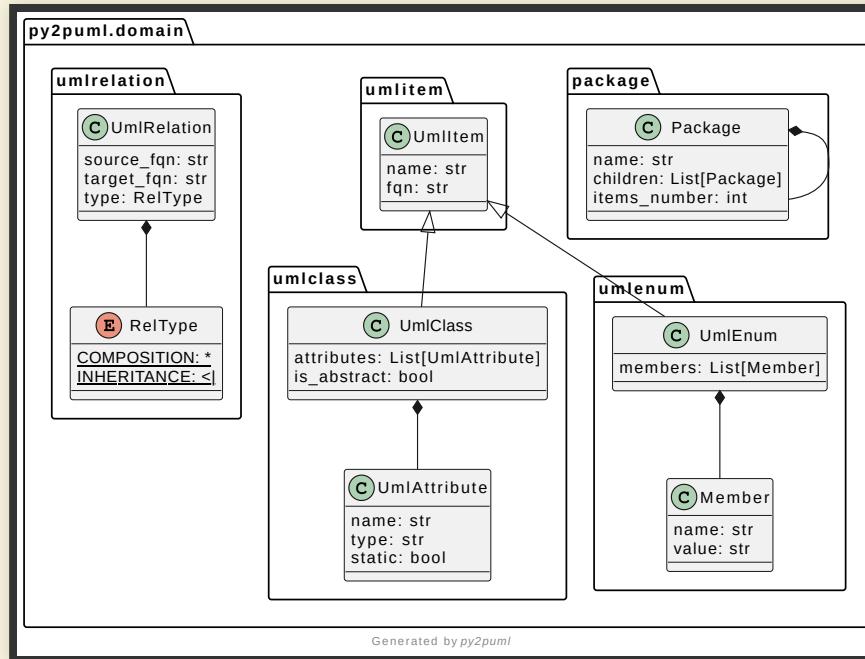
## Qui suis-je ?

---

- 💙💛 dev Python @ Purecontrol
- 🐱 github.com/lucsorel (outils code-to-doc)
- 🐘 @lucsorelgiffo@floss.social

# Le projet open-source Python

<https://github.com/lucsorel/py2puml> : génère un diagramme de classes en scannant les structures de données de la base de code.



# Syndrôme de Spiderman

---

## Spiderman

- 1 piqûre : 😷
- grimpe aux murs : 💪
- anime des soirées mousse  
avec les poignets : 🎉



# Syndrôme de Spiderman

## Spiderman

- 1 piqûre : 😷
- grimpe aux murs : 💪
- anime des soirées mousse avec les poignets : 🎉



py2puml

- 140+ ⭐ : 🎉
- 37 issues (24 fermées) : 😷
- 25 PR (21 fermées) : 😢

*"De nouvelles responsabilités demandent de nouveaux pouvoirs."*

Luc (du 35) 🕸️

## **Accueillir et cadrer les contributions**

---

- non-régression des fonctionnalités
- homogénéité des pratiques
  - formatage de code
  - qualité de code

## **CONTRIBUTING.md dans le dépôt ?**

---



Des conventions non outillées finissent dans l'oubli.

## Intégration continue

- non-régression → **tests automatisés** (+ couverture de code testé)
- homogénéité de la base de code
  - formatage de code → **formateur**
  - qualité de code → **linter**

# Mise en place des Github actions

Syntaxe **déclarative** permettant d'exécuter des commandes ou des outils au fil du cycle de vie *git* du projet.

```
super-projet/  
  └── .github/  
      └── workflows/  
          └── {nom_de_workflow}.yml # ci.yml, par exemple  
          └── ...
```



# Anatomie progressive d'un workflow d'intégration continue Github

Départ : système d'exploitation + checkout du code source

```
name: Python CI # facultatif

on: push          # évènement(s) git déclencheur(s)

jobs:
  build:           # le nom de l'opération
    runs-on: ubuntu-latest # système d'exploitation utilisé
    steps:            # étapes de l'opération
      - name: Récupération du code # facultatif
        uses: actions/checkout@v4 # utilisation d'une recette existante (! @version)
```

Documentation officielle des Github actions.

## Actions : les recettes de base

---

- action = brique réutilisable et adaptable à votre besoin
  - chacune a son dépôt
  - configuration décrite dans `action.yaml`
  - parfois accompagnée de scripts référencés dans `action.yaml`
- 62 actions "officielles" proposées par Github (septembre 2023)
- nombreuses actions communautaires

Exemples : [github.com/actions/checkout](https://github.com/actions/checkout), [github.com/abatilo/actions-poetry](https://github.com/abatilo/actions-poetry)

# **Opération de build du projet**

---

## Opération de build du projet

1. récupérer les sources du projet 

## Opération de build du projet

1. récupérer les sources du projet 
2. installer python

## Opération de build du projet

1. récupérer les sources du projet 
2. installer python
3. installer poetry

## Opération de build du projet

1. récupérer les sources du projet 
2. installer python
3. installer poetry
4. installer les dépendances

## Opération de build du projet

1. récupérer les sources du projet 
2. installer python
3. installer poetry
4. installer les dépendances
5. lancer les tests automatisés + couverture

## Opération de build du projet

1. récupérer les sources du projet 
2. installer python
3. installer poetry
4. installer les dépendances
5. lancer les tests automatisés + couverture
6. auditer la qualité du code

# Installation de Python - 1/2

Projet utilisant **pyenv** pour définir la version de python.

```
name: Python CI
on: push

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Install Python
        uses: actions/setup-python@v4          # action officielle
        with:                                  # configuration pour pyenv
              python-version-file: '.python-version' # maj transparente de l'IC 
```

# Installation de Python - 2/2

## Configuration alternatives de l'action :

```
steps:  
  # dernière version disponible (⚠️ varie en fonction du runner)  
  - uses: actions/setup-python@v4  
  
  # version spécifiée en dur 🎭  
  - uses: actions/setup-python@v4  
    with:  
      python-version: '3.9'  
  
  # utilisation de PyPy  
  - uses: actions/setup-python@v4  
    with:  
      python-version: 'pypy3.9'
```

Voir d'autres cas d'usage avancés (intervalle ou matrice de versions, caches pour outils spécifiques, etc.).

# Installation des dépendances 1/2

- utilisation de **poetry** pour la gestion des dépendances
- dossier `.venv` / `local` (à mettre en cache 

```
jobs:  
  build:  
    runs-on: ubuntu-latest  
    steps:  
      # [...]  
      - name: Install poetry  
        uses: abatilo/actions-poetry@v2 # action communautaire   
        with:  
          poetry-version: 1.5.1           # optionnel  
  
      - name: Cache the virtual environment  
        uses: actions/cache@v3          # action officielle  
        with:  
          path: ./venv  
          key: venv-${{ hashFiles('poetry.lock') }} # clé d'éviction sur le lock-file  
  
      - name: Install project dependencies  
        run: poetry install             # exécution explicite d'une commande 
```

## Installation des dépendances 2/2

Alternative avec pip + requirements.txt directement :

```
steps:  
  - uses: actions/setup-python@v4  
    with:  
      python-version: '3.9'  
      cache: 'pip' # utilisation d'un cache pour pip  
  
  - name: Install project dependencies  
    run: pip install -r requirements.txt # pas besoin d'environnement virtuel  
        # au sein d'un runner
```

# Exécution des tests

---

```
env:  
  MIN_CODE_COVERAGE_PERCENT: 93 # ➡️ définition d'une variable  
  
jobs:  
  build:  
    runs-on: ubuntu-latest  
    steps:  
      # [...]  
      - name: Run automated tests (with code coverage)  
        run: > # '>' permet de séparer une commande sur plusieurs lignes  
          poetry run pytest -v --cov=py2puml --cov-branch  
          --cov-report term-missing  
          # l'opération sera en échec si le taux de couverture descend sous le seuil  
          --cov-fail-under $MIN_CODE_COVERAGE_PERCENT # utilisation ➡️
```

# Consultation des exécutions de workflow dans Github

---

1. cliquer sur le menu Actions de votre dépôt sur Github
2. cliquer sur une des exécutions de workflow

← Python package

## feat: handle union type and add pre-commit lint hooks (#51) #18

Summary

Jobs

build

Run details

Usage

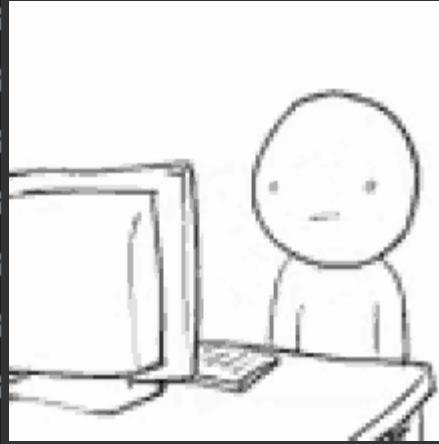
Workflow file

build

succeeded on Aug 28 in 41s

Search logs



- >  Set up job 3s
- >  Run actions/checkout@v3 1s
- >  Install Python 10s
- >  Install poetry 14s
- >   based on the dependencies lock file 1s
- >  4s
- >  3s
- >  0s
- >  component based on the dependencies lock file 1s
- >  0s
- >  0s

Démo !

## Et les hooks de pre-commit ?

---

L'outil pre-commit utilise les mécanismes de hook de git pour lancer des outils à différentes étapes du cycle de vie git d'un projet.



Figure 1. Rediffusion : <https://www.youtube.com/watch?v=l0HrTE45RVM>

# Exemple de configuration pre-commit ( .pre-commit-config.yaml)

```
repos:  
- repo: https://github.com/pre-commit/pre-commit-hooks  
  rev: v3.3.3  
  hooks:  
    - id: trailing-whitespace  
    - id: end-of-file-fixer  
    - id: double-quote-string-fixer  
  
- repo: https://github.com/google/yapf  
  rev: v0.40.2  
  hooks:  
    - id: yapf  
      additional_dependencies: [toml]  
  
- repo: https://github.com/astral-sh/ruff-pre-commit  
  rev: v0.0.292  
  hooks:  
    - id: ruff
```

Voir [pre-commit.com/hooks.html](https://pre-commit.com/hooks.html).

## pre-commit en intégration continue

- à la dure dans le runner Github d'intégration continue
  - installer git
  - lancer pre-commit

ou

- intégration avec le service [pre-commit.ci](#)

# Intégrer pre-commit.ci dans son projet Github 1/6

1. aller sur <https://pre-commit.ci/>
2. s'identifier avec Github

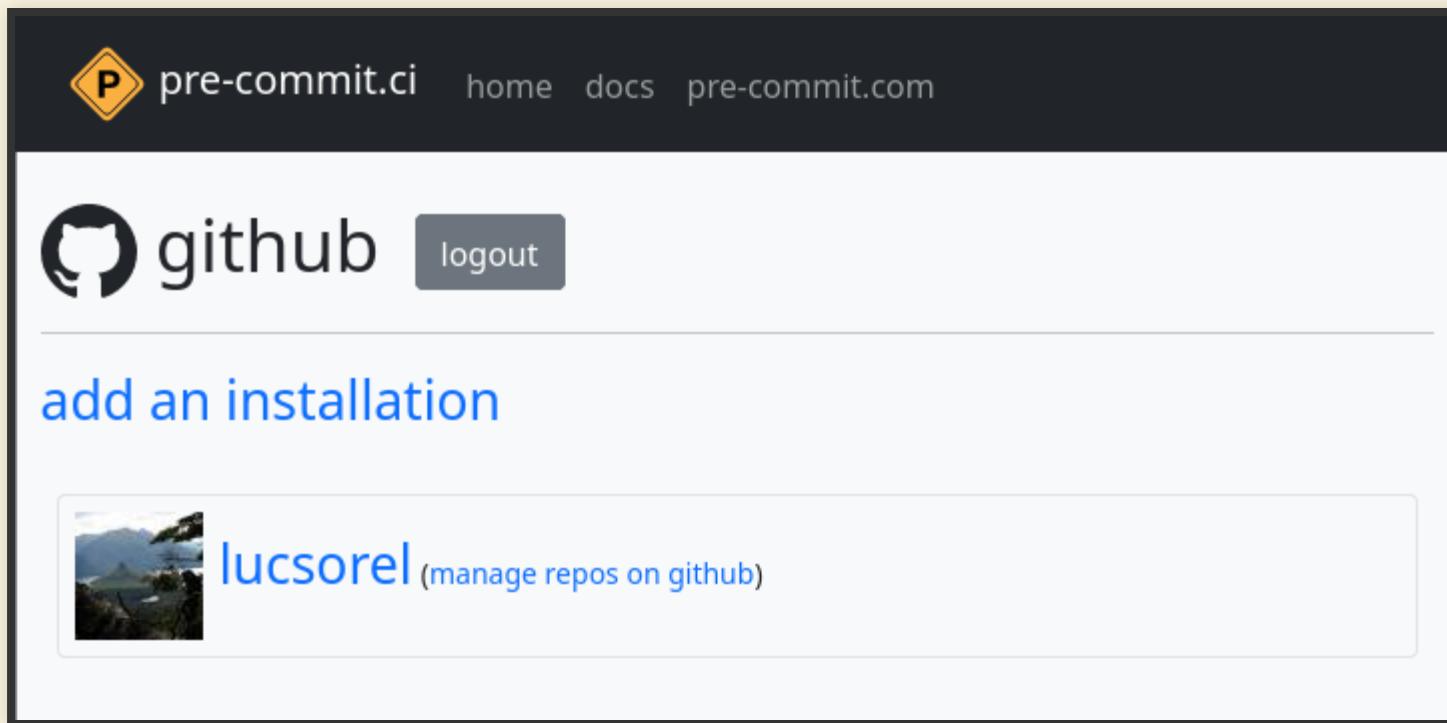
The screenshot shows the homepage of pre-commit.ci. At the top, there's a yellow diamond icon containing a black letter 'P' followed by the text 'pre-commit ci'. Below this, it says 'a continuous integration service for the pre-commit framework'. A large callout box contains text about how the service saves time by fixing trivial code issues automatically. At the bottom, there's a button labeled 'Sign In With GitHub' featuring the GitHub logo.

Developers spend a fair chunk of time during their development flow on fixing relatively trivial problems in their code. **pre-commit.ci** both enforces that these issues are discovered (which is opt-in for each developer's workflow via **pre-commit**) but also fixes the issues automatically, letting developers focus their time on more valuable problems.

 Sign In With GitHub

# Intégrer pre-commit.ci dans son projet Github 2/6

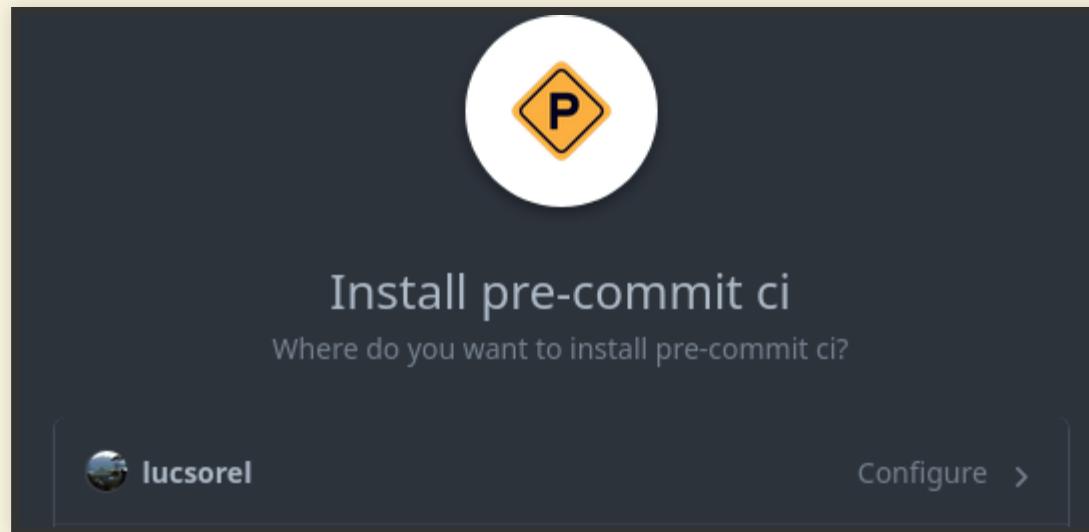
- ajouter une installation



# Intégrer pre-commit.ci dans son projet Github 3/6

---

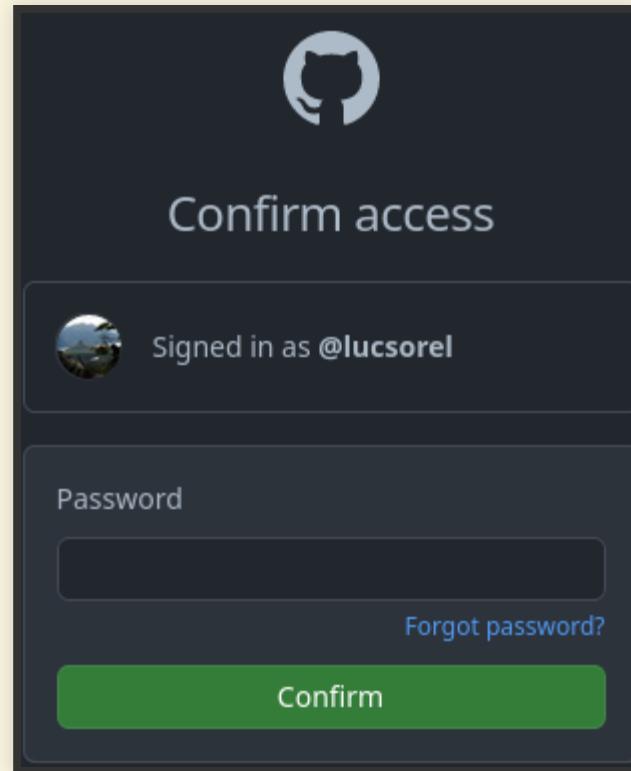
- configurer l'intégration



# Intégrer pre-commit.ci dans son projet Github 4/6

---

- confirmer l'accès au compte



# Intégrer pre-commit.ci dans son projet Github 5/6

- sélectionner le dépôt de code à intégrer

The screenshot shows the GitHub user settings interface for Luc Sorel-Giffo (lucsorel). On the left, there's a sidebar with links like Public profile, Account, Appearance, Accessibility, Notifications, Access (Billing and plans, Emails, Password and authentication, Sessions, SSH and GPG keys, Organizations, Moderation), Code, planning, and automation (Repositories, Codespaces, Packages, Copilot, Pages, Saved replies), and Security. The main area displays the 'pre-commit ci' application card, which was installed last month and developed by pre-commit-ci. It has two permissions listed: 'Read access to issues, merge queues, and metadata' and 'Read and write access to code, commit statuses, pull requests, and workflows'. Below this, the 'Repository access' section is shown, with the 'Only select repositories' option selected. A dropdown menu titled 'Select repositories' contains the entry 'lucsorel/py2puml', which generates PlantUML class diagrams for Python applications.

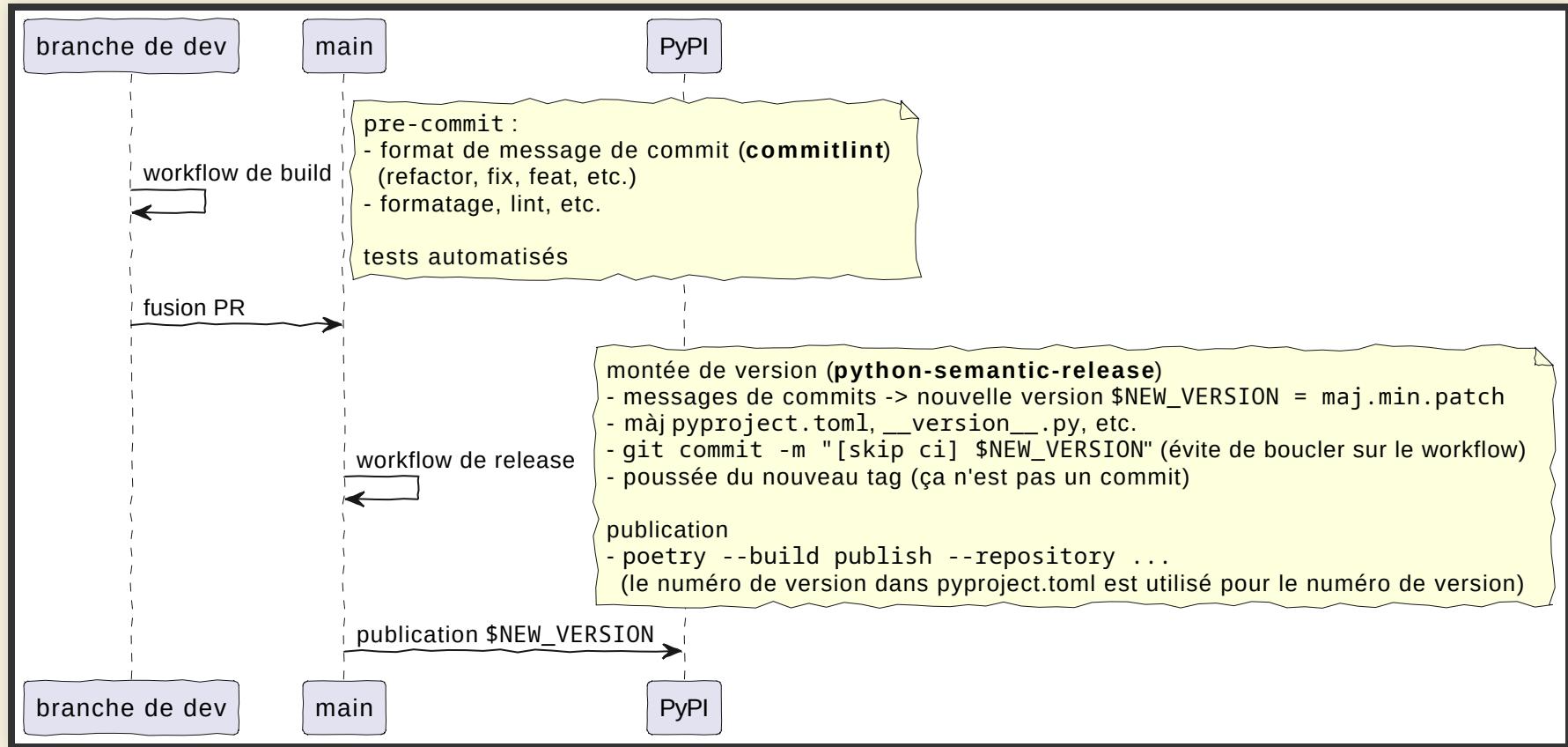
# Intégrer pre-commit.ci dans son projet Github 6/6

👉 Ne se déclenche que dans le cadre d'une pull-request.

The screenshot shows a status page from pre-commit.ci for a pull request. At the top, there's a green header with a checkmark icon and the text "lucsorel / py2puml · #62". Below the header, it says "queued at: 2023-10-02 19:41:48.610909" and "total time to completion: 17.8s". A timeline below shows the stages of the build: queue (79ms), mergeable check (985ms), clone (306ms), ci config (1ms), pull image (303µs), build (13.4s), download (752ms), and run (2.3s). The run section lists various checks: check yaml, check toml, trim trailing whitespace, fix end of files, fix double quoted strings, check for added large files, python tests naming, isort, Yapf, and ruff, all of which have passed.

check	status
check yaml	Passed
check toml	Passed
trim trailing whitespace	Passed
fix end of files	Passed
fix double quoted strings	Passed
check for added large files	Passed
python tests naming	Passed
isort	Passed
Yapf	Passed
ruff	Passed

# Prochaines étapes



- doc skip ci
- hook **commitlint**
- doc Python Semantic Release

# Les Github actions !

---



- système gratuit de CI / CD
- YAML déclaratif
- originalité des actions : briques composables
- s'adapte à une diversité de projets

**Merci !**

---

Des questions ?

Présentation à retrouver sur [github.com/lucsorel/conferences/tree/main/python-rennes-2023.10.10-cicd-projets-python](https://github.com/lucsorel/conferences/tree/main/python-rennes-2023.10.10-cicd-projets-python) 