

# Intégration continue

Nous avons mis en place l'intégration continue de notre projet en utilisant GitHub Actions. Pour faire ainsi, nous avons créé un fichier YAML nommé `test.yaml` qui décrit les étapes nécessaires pour exécuter les tests de notre projet à chaque fois qu'un commit est effectué sur la branche `main` ou `develop`.

Voici une explication de notre fichier de tests '`tests.yaml`' :

## `name`

- Ceci définit le nom du workflow. Dans notre cas, le workflow est nommé "Tests".

## `on`

- Cette section définit les événements déclencheurs qui lanceront le workflow. Nous avons défini trois événements : `pull request`, `push` sur les branches `main` et `develop`, et `workflow_dispatch`. Le premier déclenche le workflow lorsque des pull-requests seront créées ou mises à jour. Le deuxième déclenche le workflow lorsque des commits seront effectués sur les branches `main` ou `develop`. Enfin, le dernier permet de déclencher le workflow manuellement via l'interface utilisateur GitHub.

## `env`

- Cette section définit les variables d'environnement pour le workflow. Nous avons défini `fail-fast` sur `true`, ce qui signifie que le workflow s'arrêtera immédiatement si une étape échoue.

## `permissions`

- Cette section définit les autorisations de l'action. Nous avons défini `contents` sur `read`, ce qui signifie que l'action ne peut lire que les contenus du référentiel.

## `jobs`

- Cette section définit les tâches que le workflow doit exécuter. Nous avons défini une seule tâche appelée `"test"`, qui exécutera les tests de notre projet.

## `name (job)`

- Ceci définit le nom de la tâche. Nous avons nommé la tâche `"Run Python Tests"`.

## `runs-on`

- Ceci définit l'environnement d'exécution de la tâche sur ubuntu-latest, ce qui signifie que les tests seront exécutés sur un système d'exploitation Ubuntu.

### steps

- Cette section définit les étapes que la tâche doit effectuer. Nous avons défini six étapes dans notre fichier YAML :

- *Checkout Repository*

Cette étape permet de récupérer le code source à partir du référentiel. Nous avons utilisé l'action actions/checkout@v2 pour cela. Nous avons également défini la valeur de ref sur `${{ github.head_ref }}`, ce qui signifie que l'action récupérera la branche ou la PR sur laquelle l'événement a été déclenché.

- *Set up Python*

Cette étape permet de configurer Python pour l'exécution des tests. Nous avons utilisé l'action actions/setup-python@v2 pour cela. Nous avons également défini la version de Python à utiliser sur 3.11.

- *Upgrade pip*

Cette étape permet de mettre à jour la version de pip. Nous avons exécuté la commande `python -m pip install --upgrade pip` pour cela. Nous avons également défini `continue-on-error` sur `true`, ce qui signifie que le workflow continuera à s'exécuter même si cette étape échoue.

- *Install dependencies*

Cette étape permet d'installer les dépendances requises pour exécuter les tests. Nous avons exécuté la commande `pip install -r requirements.txt` pour cela.

- *Run BDD tests*

Cette étape permet d'exécuter les tests de comportement dirigés par les tests (BDD). Nous avons exécuté la commande `python tests/bdd-unittests.py` pour cela.

- *Run TDD tests*

Cette étape permet d'exécuter les tests de développement dirigés par les tests (TDD). Nous avons exécuté la commande `python tests/tdd-unittests.py` pour cela.

**Voici un exemple de scénario, pour lequel tous les tests sont passés avec succès :**

Les 11 tests du fichier BDD, ainsi que les 3 tests du TDD sont passés.

← Tests

✓ feature/tests-automation wip #10

Summary

Jobs

Run Python Tests

Run details

Usage

Workflow file

Run Python Tests

succeeded 54 minutes ago in 9s

> ✓ Set up job

> ✓ Checkout Repository

> ✓ Set up Python

> ✓ Upgrade pip

> ✓ Install dependencies

▼ ✓ Run BDD tests

1 ▶ Run python tests/bdd-unittests.py

8 .....

9 -----

10 Ran 11 tests in 0.001s

11

12 OK

▼ ✓ Run TDD tests

1 ▶ Run python tests/tdd-unittests.py

8 ...

9 -----

10 Ran 3 tests in 0.001s

11

12 OK

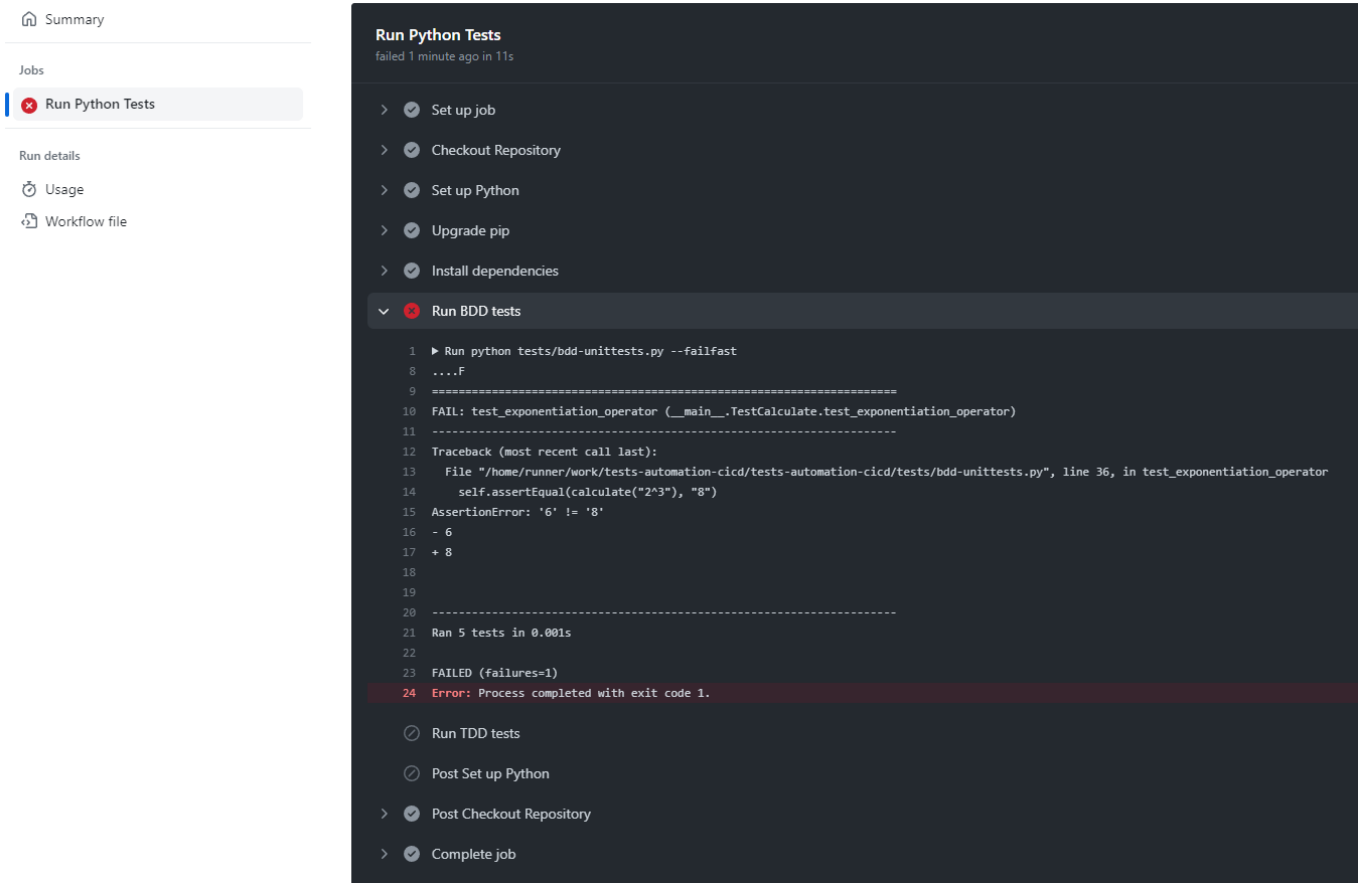
> ✓ Post Set up Python

> ✓ Post Checkout Repository

> ✓ Complete job

## Voici un exemple de scénario, pour lequel nous avons déclenché une erreur :

Pour insérer une erreur dans le code de sorte qu'un des tests unitaires échoue, nous avons remplacé l'opérateur `^` par `*` au lieu de `**` pour provoquer une erreur dans le calcul des expressions avec l'opérateur d'exponentiation.



The screenshot displays a GitHub Actions workflow run titled "Run Python Tests". The left sidebar shows the workflow's structure: Summary, Jobs (with "Run Python Tests" selected), Run details, Usage, and Workflow file. The main panel shows the execution steps for the "Run Python Tests" job, which failed 1 minute ago in 11s. The steps listed are: Set up job, Checkout Repository, Set up Python, Upgrade pip, Install dependencies, Run BDD tests (failed), Run TDD tests, Post Set up Python, Post Checkout Repository, and Complete job. The "Run BDD tests" step is expanded, showing a terminal output where a test fails. The failure message is: "FAIL: test\_exponentiation\_operator (\_\_main\_\_.TestCalculate.test\_exponentiation\_operator)". The traceback indicates the error occurred in the file "/home/runner/work/tests-automation-cicd/tests-automation-cicd/tests/bdd-unittests.py" at line 36, in the function test\_exponentiation\_operator. The specific error is an AssertionError: "'6' != '8'", which is preceded by a calculation showing "- 6" and "+ 8". The terminal output concludes with "Ran 5 tests in 0.001s", "FAILED (failures=1)", and "Error: Process completed with exit code 1."

Voici comment nous avons mis en place l'intégration continue de notre projet en utilisant GitHub Actions. En utilisant ce fichier YAML, les tests sont exécutés automatiquement à chaque fois qu'un commit est effectué sur les branches main ou develop, ou lorsqu'une pull request est créée ou mise à jour. Cela permet de s'assurer que le code est toujours testé et fonctionnel avant d'être fusionné dans la branche principale du projet.