

# **ITAS 276**

# **DevSecOps**

## **Lab #2**

### Using GitHub Actions for CI

Aiden Adzich  
2026/02/08

Github Repo:

<https://github.com/aidenadzich/calculator-ci>

## Figure 1

*Local Development Environment*

The screenshot shows a local development environment using the Visual Studio Code (VS Code) IDE. The left sidebar displays a file tree for a project named 'AADZICH\_276L01'. The tree includes standard directory entries like '.pycache', '.github', '.pytest\_cache', '.venv', and 'docs' (which contains three image files: 'fail1.png', 'success1.png', and 'success2.png'), along with configuration files like '.coverage', '.gitignore', '.python-version', 'calculator.py' (the active file), 'pyproject.toml', 'README.md', 'requirements.txt', 'test\_calculator.py', and 'uv.lock'. The right pane is a code editor showing the contents of 'calculator.py'. The code defines a 'Calculator' class with methods for addition, subtraction, multiplication, division, and power. It also contains a main block that prints the results of these operations. The code editor has syntax highlighting and line numbers. At the bottom, there's a navigation bar with links to 'PROBLEMS', 'OUTPUT', 'TERMINAL', 'GITLENS', 'DEBUG CONSOLE', and 'PORTS'. The terminal tab is currently selected, showing the command line prompt '(aadzich\_276L01) PS C:\Users\Aiden\Documents\School\ITAS276\aadzich\_276L01>'.

```
calculator.py X
calculator.py > ...
You, 3 minutes ago | 1 author (You)
1  class Calculator:
2      def add(self, a, b):
3          return a + b
4
5      def subtract(self, a, b):
6          return a - b
7
8      def multiply(self, a, b):
9          return a * b
10
11     def divide(self, a, b):
12         if b == 0:
13             raise ValueError("Cannot divide by zero")
14         return a / b
15
16     def power(self, a, b):
17         return a ** b
18
19     def main(): # pragma: no cover
20         calc = Calculator()
21         print("=" * 60)
22         print("~~ Simple Calculator ~~")
23         print("=" * 60)
24         print("Addition:", calc.add(5, 3))
25         print("Subtraction:", calc.subtract(10, 4))
26         print("Multiplication:", calc.multiply(6, 7))
27         print("Division:", calc.divide(20, 4))
28         print("Power:", calc.power(2, 3))
29         print("=" * 60)
30
31     if __name__ == "__main__": # pragma: no cover
32         main() You, 45 minutes ago • Initial Commit ...
```

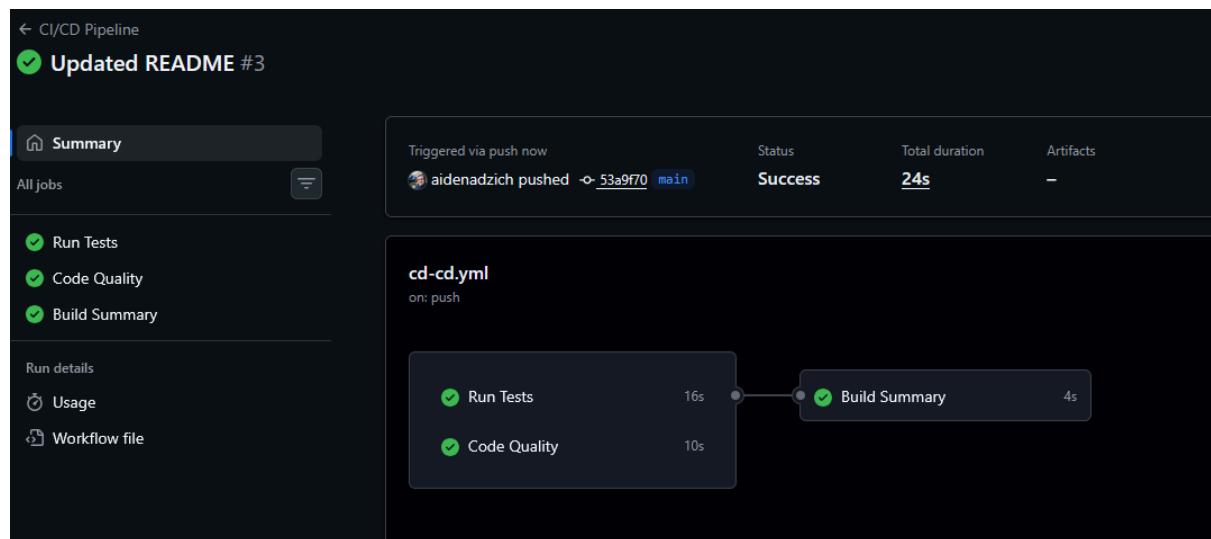
**Figure 2**

*Git ls Results*

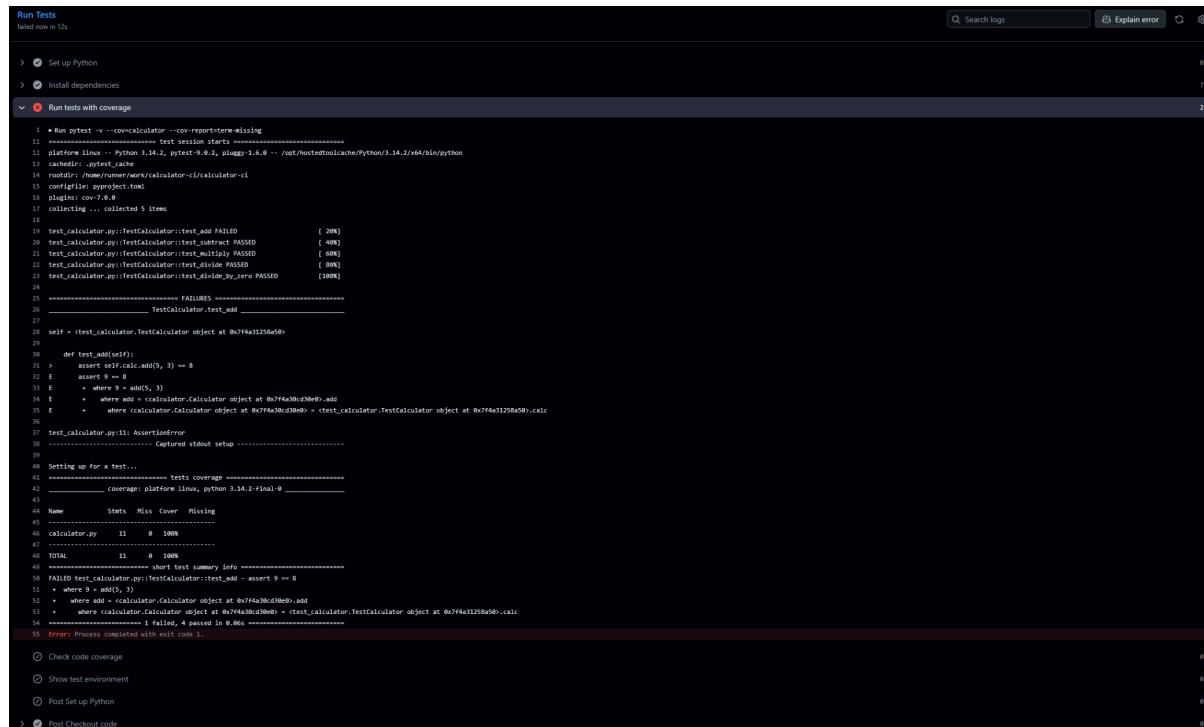
```
(aadzich_276L01) PS C:\Users\Aiden\Documents\School\ITAS276\aadzich_276L01> git ls-files
.coverage
.github/workflows/cd-cd.yml
.gitignore
.python-version
README.md
calculator.py
docs/fail1.png
docs/success1.png
docs/success2.png
pyproject.toml
requirements.txt
test_calculator.py
uv.lock
```

**Figure 3**

*GitHub action success*



**Figure 4**  
*GitHub action fail*



```

Run Tests
failed now in 12s

> ⚡ Set up Python
> ⚡ Install dependencies
✖ Run tests with coverage
  1 | Run pytest -v --cov=calculator --cov-report=term-missing
  1 | ===== test session starts =====
  1 | platform linux -- Python 3.14.2, pytest-9.0.2, pluggy-1.6.0 -- /opt/hostedtoolcache/Python/3.14.2/x64/bin/python
  1 | cache_dir: .pytest_cache
  1 | configfile: setup.cfg
  1 | configfile: pyproject.toml
  1 | configfile: pyproduct.toml
  1 | plugins: cov-7.0.0
  1 | collecting ... collected 5 items
  1 |
  1 | test_calculator.py::TestCalculator::test_add FAILED
  1 | test_calculator.py::TestCalculator::test_subtract PASSED [ 20%]
  1 | test_calculator.py::TestCalculator::test_multiply PASSED [ 40%]
  1 | test_calculator.py::TestCalculator::test_divide PASSED [ 60%]
  1 | test_calculator.py::TestCalculator::test_divide_by_zero PASSED [100%]
  1 |
  1 | ===== FAILURES =====
  1 | testCalculator.test_add
  1 |
  1 | self = ctest_calculator.TestCalculator object at 0x7f4a31258a50
  1 |
  1 | def test_add(self):
  1 |     assert self.calc.add(5, 3) == 8
  1 | E     assert 9 == 8
  1 | E     + where 9 is <calculator.Calculator object at 0x7f4a31258a50>.add
  1 | E     + where calculator.Calculator object at 0x7f4a31258a50> = ctest_calculator.TestCalculator object at 0x7f4a31258a50>.calc
  1 |
  1 | test_calculator.py::AssertionError
  1 | -----
  1 | ----- Captured stdout setup -----
  1 |
  1 | Setting up for a test...
  1 | ===== tests coverage =====
  1 | coverage: platform linux, python 3.14.2-final-0
  1 |
  1 | Name      Stats Miss  Cover  Missing
  1 | calculator.py    11      0   100%
  1 | -----
  1 | TOTAL       11      0   100%
  1 | ===== short test summary info =====
  1 | FAILED test_calculator.py::TestCalculator::test_add - assert 9 == 8
  1 | + where 9 is add(5, 3)
  1 | + where add = calculator.Calculator object at 0x7f4a31258a50>.add
  1 | + where calculator.Calculator object at 0x7f4a31258a50> = ctest_calculator.TestCalculator object at 0x7f4a31258a50>.calc
  1 | =====
  1 | Error: Process completed with exit code 1.

```

The screenshot shows a GitHub Actions pipeline step titled "Run Tests". It lists several steps: "Set up Python", "Install dependencies", and "Run tests with coverage". The "Run tests with coverage" step is highlighted with a red circle and has a status of "failed now in 12s". The log output shows the execution of a Python test suite using pytest. It includes test cases for addition, subtraction, multiplication, division, and division by zero. One test case for addition fails with an assertion error: `assert 9 == 8`. Coverage analysis is also shown, with 100% coverage for all files except for the test itself. The final message indicates the process completed with an exit code of 1.

## Reflection

The CI/CD pipeline acts as a safety net by automatically catching bugs on every push before they reach the production environment. Without it, we would need to rely on manual testing, which is both slower and prone to human error. We ran tests and linting in parallel to save time. By having both tasks execute simultaneously, we were able to check the code as it was being tested, cutting down the wait time for results. 100% code coverage ensures that every bit of logic in our code is being verified and tested during the build. This is enforced using the `--cov-fail-under` flag, which will automatically fail the test if the code coverage drops below the given threshold. For a team of developers, this setup would ensure that one person's broken or messy code won't interfere with the work of the rest of the team.