

UTS

Testing dan QA Perangkat Lunak



DAFTAR ISI

1

Pengertian Unit Test

2

Teknik Unit Testing

3

Kelebihan Unit Testing

4

Kekurangan Unit Testing

5

Pengertian Whitebox Test

6

Teknik Whitebox Testing

7

Kelebihan Whitebox Testing

8

Kekurangan Whitebox Testing

9

Implementasi dalam Python

10

Pengertian Whitebox Test

DAFTAR ISI

1

Pengertian CI/CD

2

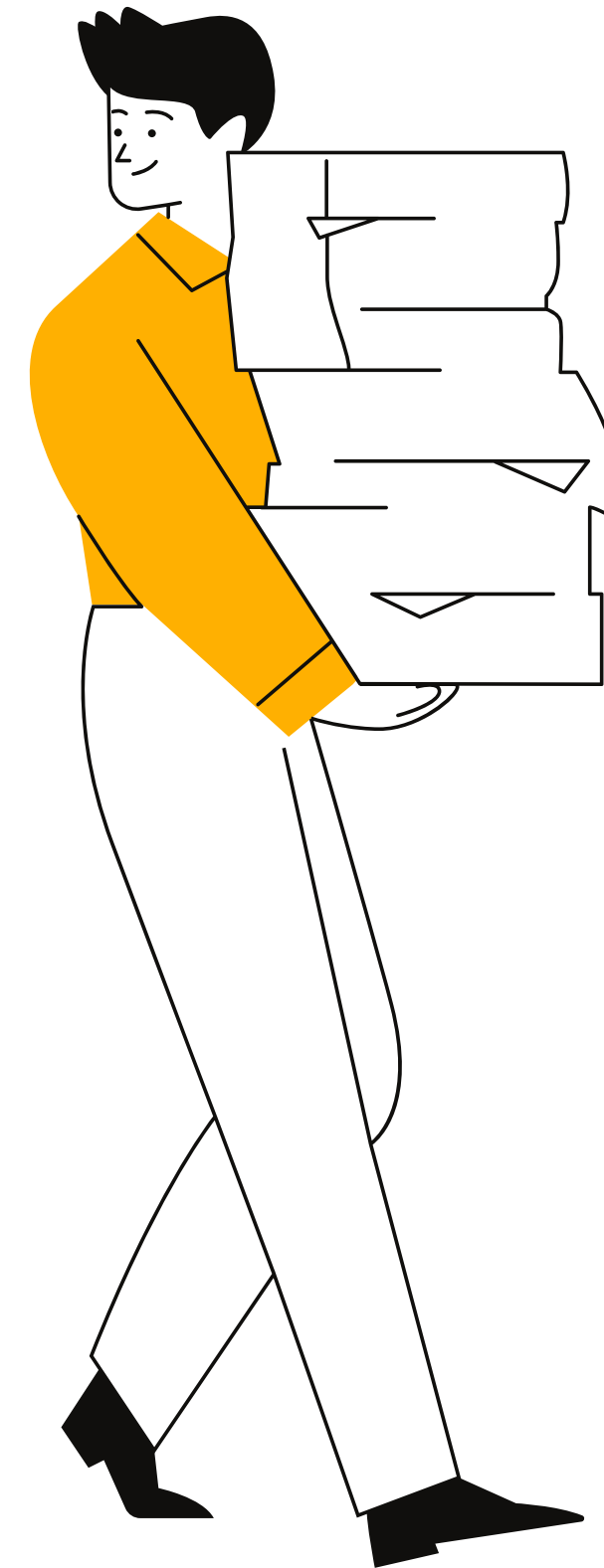
Keuntungan CI/CD

3

Tutorial CI/CD untuk Python

4

Daftar Pustaka



Unit Test

Pengujian Unit adalah jenis pengujian perangkat lunak di mana pengujian dilakukan pada unit atau komponen individu dari suatu perangkat lunak.



Teknik Unit Testing

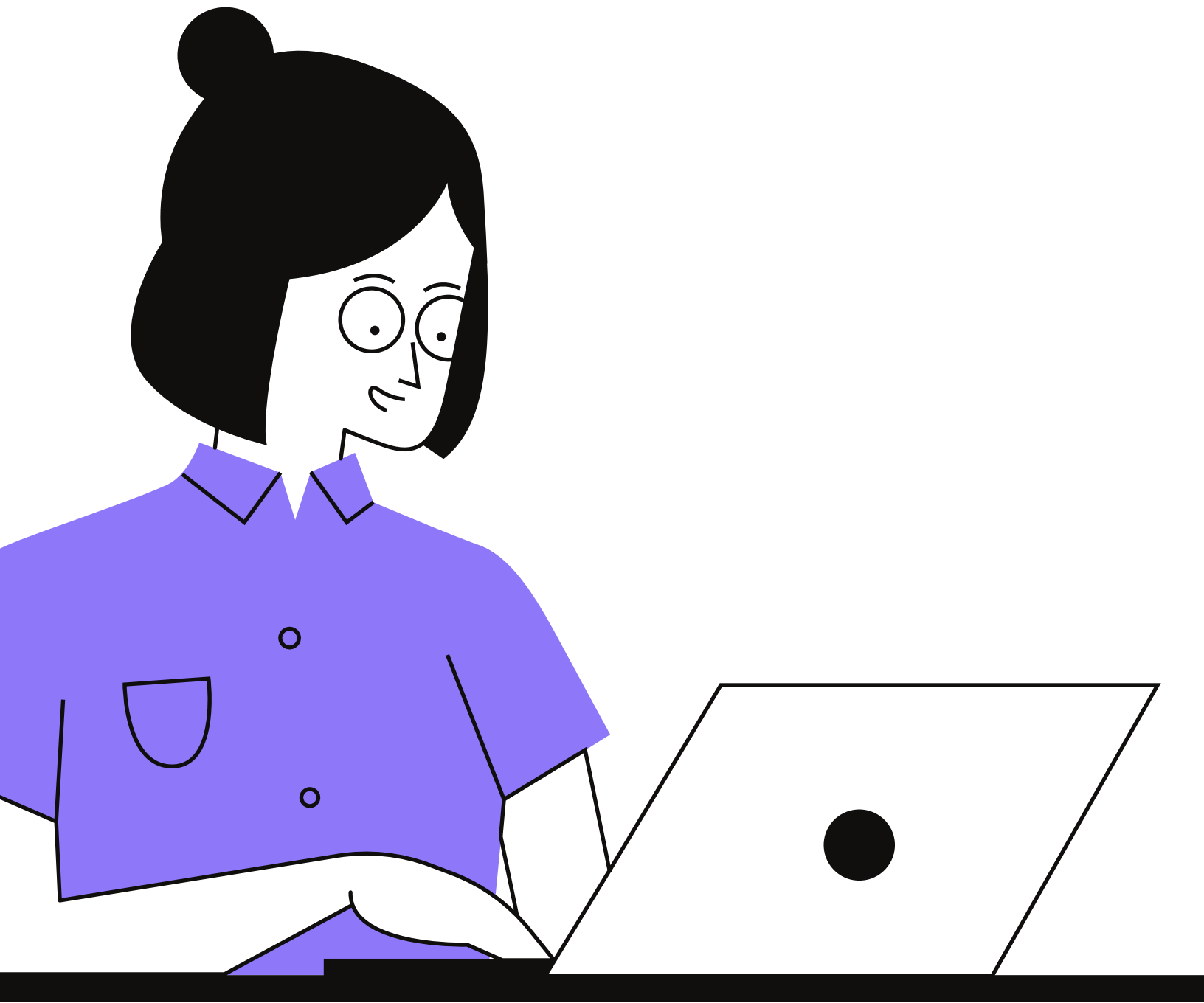
- **Black-box Testing**
Melibatkan pengujian antarmuka pengguna dan proses input – output
- **White-box Testing**
Melibatkan pengujian *functional behaviour* aplikasi perangkat lunak
- **Grey-box Testing**
Digunakan untuk menjalankan rangkaian pengujian, metode pengujian, kasus pengujian, dan melakukan analisis risiko.

Kelebihan Unit Testing

- *Developer* dapat memahami cara menggunakan dan memahami fungsionalitas suatu unit dengan melihat pengujian unit.
- Pengujian unit membantu *developer* memeriksa dan memastikan bahwa kode berfungsi dengan baik, bahkan setelah perubahan.
- Karena sifat modular dari pengujian unit, dapat dilakukan pengujian pada bagian *project* tanpa menunggu bagian lain selesai.

Kekurangan Unit Testing

- *Developer* dapat memahami cara menggunakan dan memahami fungsionalitas suatu unit dengan melihat pengujian unit.
- Pengujian unit membantu *developer* memeriksa dan memastikan bahwa kode berfungsi dengan baik, bahkan setelah perubahan.
- Karena sifat modular dari pengujian unit, dapat dilakukan pengujian pada bagian *project* tanpa menunggu bagian lain selesai.



Whitebox Test

White Box Testing adalah salah satu cara untuk menguji suatu aplikasi atau software dengan melihat modul untuk memeriksa dan menganalisis kode program

Teknik Whitebox Testing

- **Basis Path Testing**
Teknik bertujuan untuk mengukur kompleksitas kode program dan mendefinisikan alur yang dieksekusi.
- **Branch Coverage**
Pengujian ini dirancang agar setiap branch code diuji setidaknya satu kali.
- **Condition Coverage**
Tujuannya untuk menguji seluruh kode agar menghasilkan nilai TRUE atau FALSE. Dengan begitu, tester dapat memastikan perangkat lunak dapat bekerja dan mengeluarkan output sesuai dengan input dari pengguna.

Teknik Whitebox Testing

- **Loop Testing**
Pengujian ini yang wajib dilakukan untuk menguji berbagai perulangan/looping yang ada dalam program
- **Multiple Condition Coverage**
Teknik ini dilakukan untuk menguji seluruh kombinasi dari kode yang mungkin digunakan dalam berbagai kondisi.
- **Statement Coverage**
Dengan pengujian ini, dapat mengetahui kode-kode yang error sehingga dapat segera memperbaikinya.

Kelebihan Whitebox Testing

- Meningkatkan ketelitian dalam mengimplementasikan perangkat lunak.
- Memudahkan dalam menemukan kesalahan atau bug dalam perangkat lunak yang sebelumnya tidak terlihat.
- Memudahkan pengujian karena dilakukan secara menyeluruh sehingga memperkecil kemungkinan terjadinya error pada kode.
- Meminimalisir error atau bug karena pengujian dapat dilakukan sebelum perangkat lunak diluncurkan.

Kekurangan Whitebox Testing

- Menyusahkan karena pengujian ini cukup kompleks.
- Memerlukan waktu kembali ketika menambahkan atau mengganti kode, karena kamu perlu menguji keseluruhan kode kembali.
- Memakan sumber daya yang banyak karena White-box testing termasuk ke dalam pengujian yang cukup mahal.

Contoh Implementasi Dalam Python

```
root@LeppyKadall: /mnt/c/U: x + v
GNU nano 6.4                               dewa1.py
import unittest

class TestCaseString(unittest.TestCase):

    #test case string 'dewa' diubah menjadi kapital apakah sama dengan 'DEWA'
    #jika hasil success
    def test_uppersucc(self):
        self.assertEqual('dewa'.upper(), 'DEWA')
    #jika hasil fail
    def test_upperfail(self):
        self.assertEqual('dewa'.upper(), 'Dewa')

    #test case string input merupakan kapital semua
    def test_isupper(self):
        self.assertTrue('DEWA'.isupper())
        self.assertFalse('Dewa'.isupper())

    #test case jika string input di split, apakah sesuai dengan hasil yang ditentukan
    def test_split(self):
        s = 'Dewa Irtzadhany'
        self.assertEqual(s.split(), ['Dewa', 'Irtzadhany'])
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()

[ Read 26 lines (Converted from DOS format) ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy
```

Contoh Implementasi Dalam Python

```
root@LeppyKadall: /mnt/c/U: × + v

(root@LeppyKadall)-[/mnt/c/Users/Kadall/Pictures/smt7/Testing QA]
# nano dewa1.py

(root@LeppyKadall)-[/mnt/c/Users/Kadall/Pictures/smt7/Testing QA]
# python3 dewa1.py -v
test_isupper (__main__.TestCaseString.test_isupper) ... ok
test_split (__main__.TestCaseString.test_split) ... ok
test_upperfail (__main__.TestCaseString.test_upperfail) ... FAIL
test_uppersucc (__main__.TestCaseString.test_uppersucc) ... ok

=====
FAIL: test_upperfail (__main__.TestCaseString.test_upperfail)
-----
Traceback (most recent call last):
  File "/mnt/c/Users/Kadall/Pictures/smt7/Testing QA/dewa1.py", line 11, in test_upperfail
    self.assertEqual('dewa'.upper(), 'Dewa')
AssertionError: 'DEWA' != 'Dewa'
- DEWA
+ Dewa

-----

Ran 4 tests in 0.007s

FAILED (failures=1)
```

CI/CD

CI/CD termasuk dalam ranah DevOps dan menggabungkan praktik-praktik continuous integration dan continuous delivery. CI/CD mengotomatisasi sebagian besar atau seluruh intervensi manusia yang biasanya dilakukan secara manual.



Continuous Integration

Continuous integration adalah praktik menggabungkan perubahan kode secara teratur ke dalam repositori main branch source code dan dilakukan pengujian otomatis. Dengan ini, kesalahan dapat ditemukan dan diperbaiki lebih cepat. Dengan seringnya penggabungan dan pengujian otomatis, konflik kode berkurang, dan dapat memperbaiki masalah dengan cepat. Proses dimulai dengan analisis kode statis untuk memeriksa kualitasnya, lalu kode dikompilasi dan diuji otomatis, dengan penggunaan sistem kontrol versi.

Continuous Delivery

Continuous delivery adalah praktik pengembangan perangkat lunak yang bekerja dengan CI untuk otomatisasi penyediaan infrastruktur dan rilis aplikasi. Setelah kode diuji dan dibangun di CI, CD memastikan bahwa kode dikemas dengan semua yang diperlukan untuk diterapkan di lingkungan apa pun kapan saja, mulai dari penyediaan infrastruktur hingga penerapan aplikasi. CD memungkinkan penerapan perangkat lunak ke *production* kapan saja, baik secara manual atau otomatis.

Kelebihan CI/CD

- Lebih sedikit bug dan kesalahan yang masuk ke *production*
- Meningkatnya Efektivitas dan Efisiensi
- Lebih mudah menemukan dan memperbaiki bug
- CI/CD memudahkan perbaikan masalah dan pemulihan dari insiden, mengurangi *mean time to resolution* (MTTR).

Tutorial CI/CD dengan Github Actions


1. Membuat Repository pada Github dan Gitclone pada local env

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner *

 Kuliah-Dewa ▾

Repository name *

TestingQA

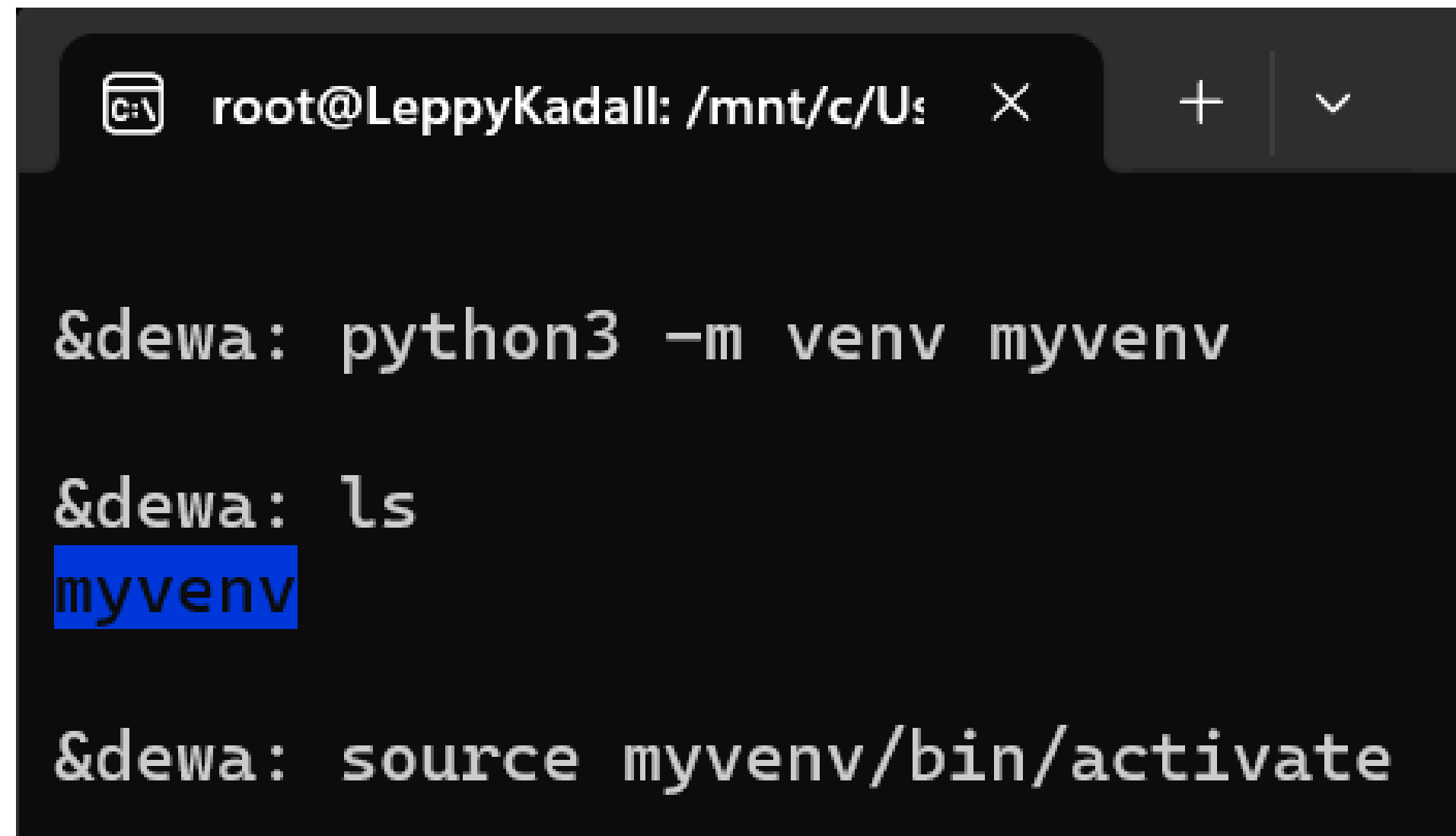
⚠ The repository TestingQA already exists on this account.

Great repository names are short and memorable. Need inspiration? How about [automatic-octo-carnival](#) ?

Description (optional)

Tutorial CI/CD dengan Github Actions

2. Buat dan Aktifkan Virtual Env dengan Python



```
root@LeppyKadall: /mnt/c/Us  X  +  v

&dewa: python3 -m venv myvenv

&dewa: ls
myvenv

&dewa: source myvenv/bin/activate
```

Tutorial CI/CD dengan Github Actions

3. Install Flask dan Pytest menggunakan pip

```
(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example]
# pip3 install flask pytest
Requirement already satisfied: flask in /usr/lib/python3/dist-packages (2.2.2)
Requirement already satisfied: pytest in /usr/lib/python3/dist-packages (7.2.1)
Requirement already satisfied: attrs>=19.2.0 in /usr/lib/python3/dist-packages (from pytest) (22.2.0)
Requirement already satisfied: iniconfig in /usr/lib/python3/dist-packages (from pytest) (1.1.1)
Requirement already satisfied: packaging in /usr/lib/python3/dist-packages (from pytest) (23.0)
Requirement already satisfied: pluggy<2.0, >=0.12 in /usr/lib/python3/dist-packages (from pytest) (1.0.0+repack)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead : https://pip.pypa.io/warnings/venv

(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example]
# pip freeze > requirements.txt
```

Tutorial CI/CD dengan Github Actions

4. Buat folder src dan file app.py di dalam folder src

```
(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example]
# mkdir src

(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example]
# cd src

(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example/src]
# nano app.py
```

```
GNU nano 7.2
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "Hello, world!"

if __name__ == "__main__":
    app.run()
```

```
(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example/src]
# python3 app.py
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a pr
tion WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Tutorial CI/CD dengan Github Actions

5. Buat folder tests dan file test_app.py di dalam folder tests. Jalankan pytest untuk software testing

```
(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example]
# mkdir tests

(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example]
# cd tests

(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example/tests]
# nano test_app.py
```

```
GNU nano 7.2
from app import index

def test_index():
    assert index() == "Hello, world!"
```

```
(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example]
# export PYTHONPATH=src

(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example]
# echo $PYTHONPATH
src

(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example]
# pytest

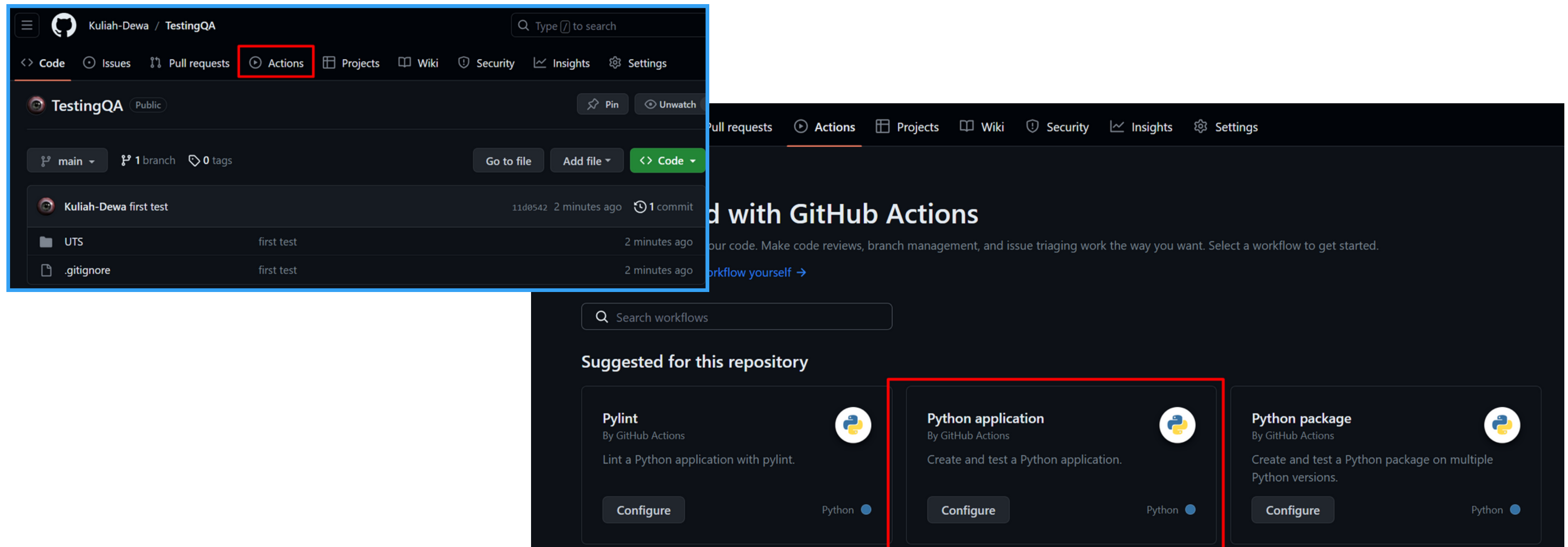
===== test session starts =====
platform linux -- Python 3.11.2, pytest-7.2.1, pluggy-1.0.0+repack
rootdir: /mnt/hgfs/UTSQA/cicd-example
plugins: anyio-3.6.2
collected 1 item

tests/test_app.py . [100%]

===== 1 passed in 0.11s =====
```

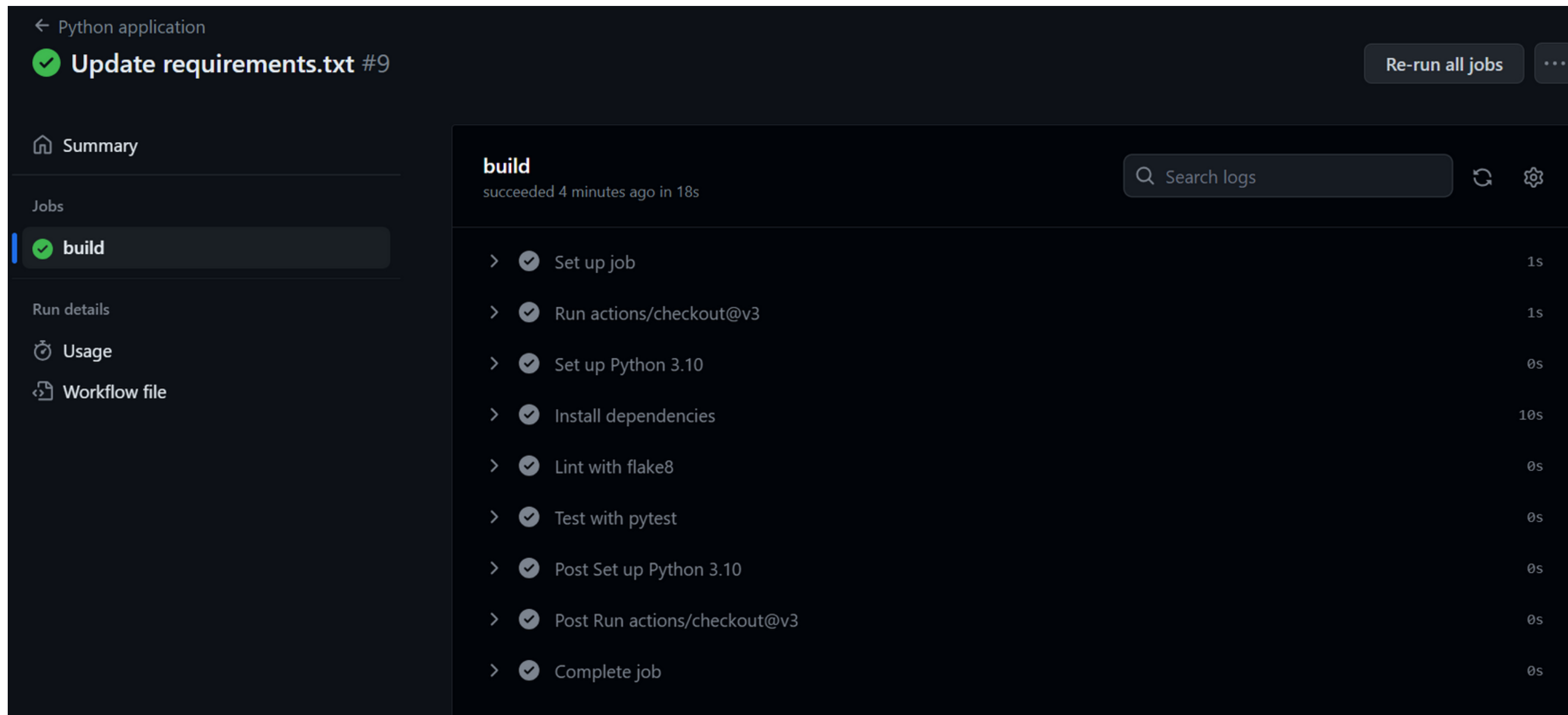
Tutorial CI/CD dengan Github Actions

6. Push file ke dalam github dan buka fitur Actions, pilih Python application. Kemudian commit file



Tutorial CI/CD dengan Github Actions

7. Continuous Integration akan otomatis berjalan setiap melakukan perubahan



The screenshot shows the GitHub Actions interface for a workflow named 'Python application'. The specific run is for the job 'Update requirements.txt #9', which has a green checkmark indicating success. The left sidebar contains navigation links: 'Summary', 'Jobs', 'Run details', 'Usage', and 'Workflow file'. The 'Jobs' section is expanded, showing a list of jobs with 'build' selected and marked as successful. The main area displays the details of the 'build' job, which succeeded 4 minutes ago in 18 seconds. A search bar for logs is present. The job steps are listed as follows:

Step	Duration
> ✓ Set up job	1s
> ✓ Run actions/checkout@v3	1s
> ✓ Set up Python 3.10	0s
> ✓ Install dependencies	10s
> ✓ Lint with flake8	0s
> ✓ Test with pytest	0s
> ✓ Post Set up Python 3.10	0s
> ✓ Post Run actions/checkout@v3	0s
> ✓ Complete job	0s

Tutorial CI/CD dengan Github Actions

8. Login ke aplikasi heroku untuk deployment

```
(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example]
# heroku login
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to [REDACTED]
[REDACTED]
Logging in... done
Logged in as dewa. [REDACTED]

(myvenv)-(root@kali)-[/mnt/hgfs/UTSQA/cicd-example]
# heroku create
Creating app... !
```

Tutorial CI/CD dengan Github Actions

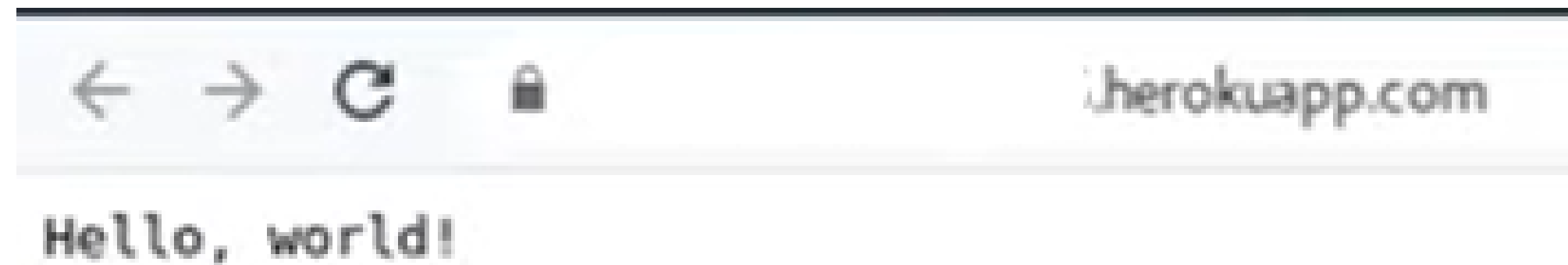
8. Deploy aplikasi ke heroku dengan menambahkan code ke pythonapp.yml

'Deploy to Heroku' Action

```
name: Deploy to Heroku
env:
  HEROKU_API_TOKEN: ${ secrets.HEROKU_API_TOKEN }
  HEROKU_APP_NAME: ${ secrets.HEROKU_APP_NAME }
if: github.ref == 'refs/heads/master' && job.status == 'success'
run: |
  git remote add heroku https://heroku:$HEROKU_API_TOKEN@git.heroku.com/$HEROKU_APP_NAME.git
  git push heroku HEAD:master -f
```

Tutorial CI/CD dengan Github Actions

9. Continuous Deployment akan otomatis berjalan setiap kali melakukan perubahan



DAFTAR PUSTAKA

- <https://socs.binus.ac.id/2020/07/02/teknik-dalam-white-box-dan-black-box-testing/>
- <https://www.dicoding.com/blog/white-box-testing/>
- <https://www.guru99.com/unit-testing-guide.html>
- <https://docs.python.org/3/library/unittest.html>
- <https://about.gitlab.com/topics/ci-cd/>
- https://www.youtube.com/watch?v=WTofttoD2xg&t=1883s&ab_channel=IndianPythonista

Link Github : <https://github.com/Kuliah-Dewa/TestingQA>