A continuación, te proporciono un ejemplo de flujo de trabajo de GitHub Actions que ejecuta pruebas unitarias para un proyecto de Python:

```
name: Python Unit Tests

on: [push, pull_request]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: '3.9'
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
        pip install pytest
    - name: Run tests
```

```
      run: |
        pytest tests/
```

Este flujo de trabajo se desencadena en cada push y pull_request en la rama predeterminada. Se ejecuta en una máquina virtual con Ubuntu 20.04 y utiliza Python 3.9. El flujo de trabajo realiza las siguientes acciones:

1. Clona el repositorio actual.
2. Configura Python y establece la versión de Python en 3.9.
3. Instala las dependencias necesarias para ejecutar el proyecto, incluyendo pytest para ejecutar las pruebas unitarias.
4. Ejecuta las pruebas unitarias utilizando pytest.

La estructura de un repositorio de Python con algunas pruebas unitarias y un archivo requirements.txt que especifica las dependencias necesarias es:

```
my-python-project/
├── my_python_code.py
├── requirements.txt
└── tests/
    ├── test_sumar.py
    ├── test_restar.py
    └── test_multiplicar.py
```

En este ejemplo, el archivo my_python_code.py contiene el código Python que se quiere probar. El archivo requirements.txt especifica las dependencias necesarias para el proyecto. En la carpeta tests/ se encuentran varios archivos de prueba que cubren diferentes partes del código, cada uno con una función test_ que realiza pruebas unitarias.

Donde el código que tendría cada archivo en el repositorio es:

Archivo my_python_code.py:

```
def sumar(a, b):
    return a + b

def restar(a, b):
    return a - b

def multiplicar(a, b):
    return a * b
```

En este ejemplo, el archivo my_python_code.py contiene tres funciones que realizan operaciones matemáticas básicas.

Archivo requirements.txt:

```
pytest==6.2.2
```

Este ejemplo es muy básico y solo especifica una dependencia, pytest, que se necesita para ejecutar las pruebas unitarias.

Archivo tests/test_sumar.py:

```
from my_python_code import sumar
def test_sumar():
    assert sumar(2, 3) == 5
    assert sumar(0, 0) == 0
    assert sumar(-1, 1) == 0
```

En este ejemplo, el archivo test_sumar.py importa la función sumar del archivo my_python_code.py y realiza tres pruebas unitarias para verificar si la función funciona correctamente.

Archivo tests/test_restar.py:

```
from my_python_code import restar
def test_restar():
    assert restar(5, 2) == 3
    assert restar(0, 0) == 0
    assert restar(-1, 1) == -2
```

En este ejemplo, el archivo test_restar.py importa la función restar del archivo my_python_code.py y realiza tres pruebas unitarias para verificar si la función funciona correctamente.

Archivo tests/test_multiplicar.py:

```
from my_python_code import multiplicar

def test_multiplicar():
    assert multiplicar(2, 3) == 6
    assert multiplicar(0, 0) == 0
    assert multiplicar(-1, 1) == -1
```

En este ejemplo, el archivo test_multiplicar.py importa la función multiplicar del archivo my_python_code.py y realiza tres pruebas unitarias para verificar si la función funciona correctamente.