Experiment No. 4:

Setting up GitHub Actions for CI/CD and Contributing to Open-Source Projects via Forks and Pull Requests

(CI/CD stands for: **CI → Continuous Integration and CD → Continuous Delivery** or **Continuous Deployment,** These are **DevOps practices** used to **automate** software development, testing, and delivery processes.)

Aim:

To explore advanced Git and GitHub functionalities including GitHub Actions for automation, working with forks and pull requests, contributing to open-source projects, and understanding GitHub security and best practices.

Expected Outcome:

By the end of this experiment, students will be able to:

- ✓ Configure GitHub Actions for automated testing or deployment.
- ✓ Fork open-source repositories and submit pull requests.
- ✓ Understand and use Git submodules.
- ✓ Practice open-source contribution workflows.
- ✓ Apply security best practices in GitHub.

Practical Conduction Steps with Examples

Part A: Set Up GitHub Actions for Automated Testing

Step-by-Step:

Create a GitHub repository (or use existing one):

Name: python-ci-demo

Add Python project files, e.g., app.py and test_app.py:

app.py

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

test_app.py

```
from app import add
def test_add():
   assert add(2, 3) == 5
```

Create GitHub Actions Workflow:

Create folder .github/workflows

Inside it, create a file python-app.yml

.github/workflows/python-app.yml

```
name: Python CI
on: [push, pull_request]
jobs:
  build:
```

```
  runs-on: ubuntu-latest
  steps:
  - name: Checkout code
    uses: actions/checkout@v3

  - name: Set up Python
    uses: actions/setup-python@v4
    with:
      python-version: 3.10

  - name: Install dependencies
    run: pip install pytest

  - name: Run tests
    run: pytest
```

Push to GitHub:

```
git add .
git commit -m "Set up GitHub Actions for Python testing"
git push origin main
```

GitHub will now automatically run tests on every push or pull request!

Part B: Fork a Repository and Submit a Pull Request

Step-by-Step:

Fork a public open-source repository
(Example: https://github.com/octocat/Spoon-Knife)

Clone the forked repo locally:

```
git clone https://github.com/your-username/Spoon-Knife.gitcd Spoon-Knife
```

Create a new branch:

```
git checkout -b feature-update
```

Make changes (e.g., edit index.html or README):

```
<!-- Add this line --><p>Contributed by Your Name 🚀</p>
```

Commit and push:

```
git add .
git commit -m "Added contribution message"
git push origin feature-update
```

Create Pull Request:
Go to your fork → Compare & pull request
Add title, description
Click **Create pull request**

Part C: Use of Git Submodules (Advanced Topic)

Example:

Add another repository as a submodule:

```
git submodule add https://github.com/username/library-repo.git libs/library-repo
```

Commit submodule entry:

```
it add .gitmodules libs/library-repo
git commit -m "Added submodule for library-repo"
```

Part D: Security and Best Practices

Best Practice	**Description**
✅ Use .gitignore	Avoid committing unnecessary files (e.g., .env, *.pyc)
✅ Branch Protection	Require PR reviews before merging to main
✅ Secrets Management	Use **GitHub Secrets** to store tokens/passwords securely
✅ License File	Include an open-source license (MIT, Apache 2.0, etc.)
✅ Dependabot	Enable automatic dependency updates
✅ Review Permissions	Set repository collaborators and team access properly

---

Summary of Commands Used:

Command	**Purpose**
git clone	Clone repo
git checkout -b	Create and switch to a new branch
git add .	Stage changes
git commit -m	Commit changes
git push	Push changes
git submodule add	Add another repo as a submodule

Let's practically perform a CI/CD pipeline setup using GitHub Actions, step-by-step with actual commands, file contents, and expected outputs.

We will use a Python project with automated testing using **pytest**, and GitHub Actions to implement the CI part of CI/CD.

Practical CI/CD Setup Using GitHub Actions

Example Project: simple-ci-app

A Python app with a basic calculator function and automated testing.

Step-by-Step with Inputs and Outputs

🔷 Step 1: Create a New GitHub Repository

Visit: https://github.com
Click **New Repository**
Fill details:
Name: simple-ci-app
Initialize with README
Click **Create Repository**

✅ **Output:** Repo created at https://github.com/your-username/simple-ci-app

Step 2: Clone the Repository Locally

```
git clone https://github.com/your-username/simple-ci-app.gitcd simple-ci-app
```

✅ **Output:** Cloned repo into simple-ci-app folder

🔷 Step 3: Add Project Files

📄 **app.py**

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

📄 **test_app.py**

```
from app import add
def test_add():
   assert add(3, 4) == 7
```

📄 **.gitignore**

```
pycache__/
*.pyc
```

✅ **Input:**

```
git add .
git commit -m "Added calculator function and test"
```

Step 4: Create GitHub Actions Workflow

Create folder and file:

```
mkdir -p .github/workflowstouch .github/workflows/python-app.yml
```

.github/workflows/python-app.yml

```
name: Python CI
```

```
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.10'

    - name: Install dependencies
      run: pip install pytest

    - name: Run tests
      run: pytest
```

Input:

```
git add .github/
git commit -m "Added GitHub Actions workflow for CI"
git push origin main
```

Step 5: Trigger GitHub Actions (Push or PR)

When you **push this code to GitHub**, GitHub Actions is automatically triggered.

✅ **Expected Output (on GitHub):**

- Go to the **Actions** tab in your repo.
- You'll see a workflow named **"Python CI"** running.
- After ~20 seconds:

```
Mathematica
✅ build job succeeded
✅ Run pytest - 1 passed, 0 failed
```

Result:

You have now successfully set up CI – every code change will automatically:

- Build
- Install dependencies
- Run tests
- Show pass/fail status

To Test it Fails:

Change test file to intentionally fail:

```
def test_add():
    assert add(3, 4) == 8  # Wrong expected value
```

✅ **Input:**

```
git commit -am "Intentional fail test"
git push
```

Expected Output (on GitHub):

```
❌ test_add failed: AssertionError
```

Summary Table

Step	Input	Output
Create Repo	GitHub UI	GitHub repo created
Clone Repo	git clone <url>	Local folder created
Add Python files	app.py, test_app.py	Files ready for test
Create Workflow	.github/workflows/python-app.yml	CI config set
Push Code	git push	Triggers GitHub Actions
View Actions	GitHub → Actions Tab	Test results shown