Lab 3: GitHub Integration & Collaboration

Overview

In **Lab 2**, you created a simple calculator library and wrote unit tests for it. You also built an integration test around a command-line driven calculator.py. Now, in **Lab 3**, we will introduce **GitHub** as a remote repository and collaboration tool. You'll learn how to:

1. Create a **remote repository** on GitHub.
2. **Push** existing local code to the remote repository.
3. **Add a new feature** to your calculator code.
4. Create a **new branch**, **commit** your changes, and open a **Pull Request (PR)**.
5. **Review** and **merge** the PR.
6. (Optional) Introduce a **merge conflict** and resolve it.

By the end of this lab, you will have a basic understanding of GitHub flows, how to integrate them with the continuous integration (CI) pipeline you began in Lab 2, and how to handle code collaboration and conflicts.

Prerequisites

1. **Git installed** on your local machine.
2. A **GitHub account**.
3. Completed code from **Lab 2** (the utils.py, calculator.py, and the tests/ folder with unit tests).

Part 1: Set Up a Remote Repository

1. **Create a new repository** in your GitHub account.
   - Go to https://github.com/ and click on **New**.
   - Give it a name, e.g., calc-lab3, and make it public
   - **Do not** initialize with a README or .gitignore at this point (to avoid conflicts).
2. **Initialize your local repository** (if not done already).
   - If you already have a local Git repository from Lab 2, skip the git init step.
   - If you didn't turn your Lab 2 code into a local Git repo yet:

```
# Bash commands
```

```
cd your_lab2_folder
git init
git add .
git commit -m "Initial commit for Lab 3"
```

3. **Link your local repo to the new GitHub repo.**
   - Copy the **remote URL** from GitHub (e.g., https://github.com/your-username/calc-lab3.git).
   - In your terminal:

```
git remote add origin https://github.com/YOUR_USERNAME/calc-lab3.git
```

   - **Push** your existing code:

```
# If your main branch is master
git push -u origin master

# If your main branch is main
git push -u origin main
```

4. **Verify that your code is now visible on GitHub:**

Refresh your newly created repo page on GitHub and confirm all your files from Lab 2 appear.

Part 2: Add a New Feature in a Separate Branch

In Lab 2, you created four basic arithmetic functions (add, subtract, multiply, divide). Let's add a **new mathematical operation** (e.g., exponentiation or logarithm). You'll do this by creating a **feature branch**.

1. **Create and switch to a new branch**:

```
# Branch names should be in the format
# [feat/fix/improvement]/DESCRIPTIVE_NAME
git checkout -b feat/exponent_operation
```

This keeps your new changes organized and separate from the main code in master or main.

2. **Implement the new feature** in utils.py. For instance:

```
def exponent(base, power):
        return base ** power
```

3. **Add corresponding unit tests** in your test file:

```
def test_exponent():
        assert exponent(2, 3) == 8
# Add more test scenarios if needed
```

4. Run linting and tests (see commands in previous lab) to ensure everything passes.
5. **Commit** your changes:

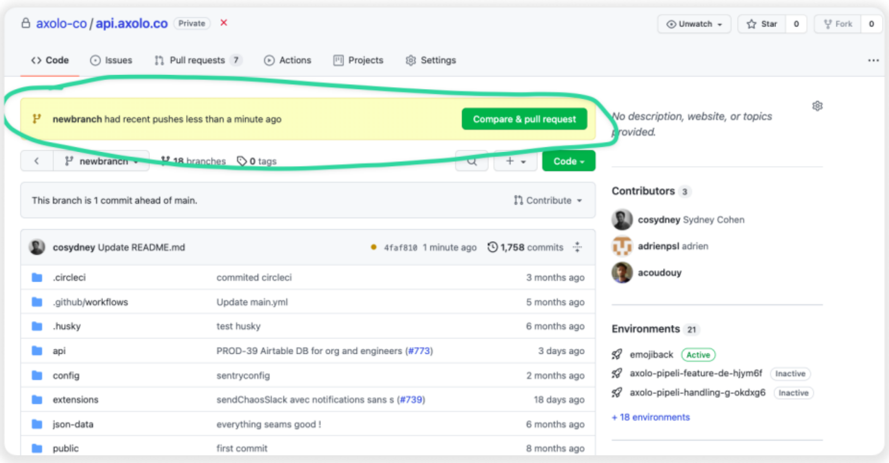
```
git add .
git commit -m "Add exponent function and tests"
```

Part 3: Push and Create a Pull Request

1. **Push your feature branch** to GitHub:

```
git push -u origin feature-exponent
```

2. Go to your repository on GitHub. You should see a prompt: **“Compare & pull request”** for your recently pushed branch.



newbranch had recent pushes less than a minute ago

This branch is 1 commit ahead of main.

cosydney Update README.md · 4faf810 1 minute ago · 1,758 commits

.circleci	commited circleci	3 months ago
.github/workflows	Update main.yml	5 months ago
.husky	test husky	6 months ago
api	PROD-39 Airtable DB for org and engineers (#773)	3 days ago
config	sentryconfig	2 months ago
extensions	sendChaosSlack avec notifications sans s (#739)	18 days ago
json-data	everything seams good !	6 months ago
public	first commit	8 months ago

No description, website, or topics provided.

Contributors 3

cosydney Sydney Cohen
adrienpsl adrien
acoudouy

Environments 21

emojiback Active
axolo-pipeli-feature-de-hjym6f Inactive
axolo-pipeli-handling-g-okdxg6 Inactive
+ 18 environments

3. Click the button to create a **Pull Request**:
   - Provide a **title** (e.g., “Add exponent functionality with tests”).
   - Fill in a **description** of what changes you made and why.

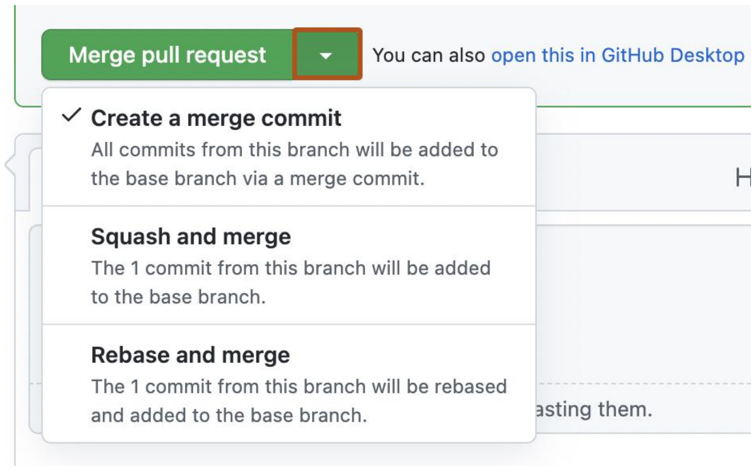
Part 4: Review the Pull Request

Even if you’re working solo, it’s good practice to simulate a review process:

1. **Open the Pull Request (PR)** on GitHub.
2. Look at the **“Files changed”** tab, and ensure everything is correct.
3. Optionally, add a **comment** to your own PR, or request a friend/classmate to review it.
4. Once you (or your reviewer) are satisfied, **approve** the PR.

Part 5: Merge the Pull Request

1. After approval (or after you decide to merge if you have no reviewers):
   - Click **“Merge pull request”** on GitHub.
   - Confirm the merge.



✓ Create a merge commit
All commits from this branch will be added to the base branch via a merge commit.

Squash and merge
The 1 commit from this branch will be added to the base branch.

Rebase and merge
The 1 commit from this branch will be rebased and added to the base branch.

asting them.

Now your **main** (or **master**) branch includes the new exponent function.

2. **Sync your local repository**:

```
git checkout master # Switches to the master/main branch
git pull origin master # Fetches and downloads content from a remote repo and immediately updates the local repo
```

(Use main if that’s the default branch in your repo.)

Part 6 (Optional): Practice Handling a Merge Conflict

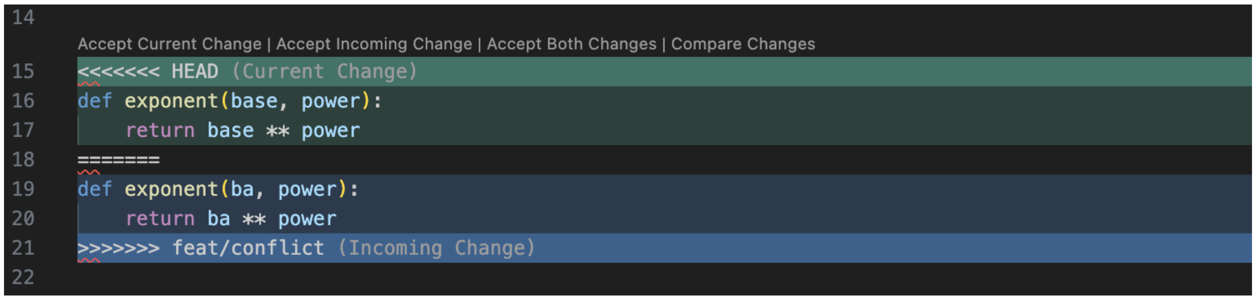
To deepen your Git knowledge, create a small merge conflict and resolve it.

1. **On master/main,** open utils.py and change the signature or the body of exponent (e.g., rename a parameter).
2. **On a new branch** (say, feat/creating_conflict), also change the same function in a different way.

3. Commit both sets of changes and attempt to merge the new branch into master.

```
git checkout master
git merge feat/creating_conflict
```

You'll see a **merge conflict** that must be resolved.


Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
<<<<<<< HEAD (Current Change)
def exponent(base, power):
    return base ** power
=======
def exponent(ba, power):
    return ba ** power
>>>>>>> feat/conflict (Incoming Change)
```

4. **Resolve** the conflict locally, usually by editing utils.py to reconcile the changes. Then:

```
git add utils.py
git commit -m "Resolve exponent function conflict"
```

5. Now that you've resolved the conflict, **push** to GitHub and finalize the merge.