Git Lab (not GitLab!)

Overview

This lab will guide you through using Git and GitHub to manage a collaborative software project using GitHub Desktop and Visual Studio Code (VS Code). You will focus on making changes, working with branches, handling merge conflicts, and managing pull request approvals.

Working in Pairs

Select a partner and assign the roles of Repository Owner (RO) and Editing Teammate (ET).

Lab Tasks

1. Forking the Repository

Forking creates a personal copy of a repository under your GitHub account. This allows you to experiment and make changes without affecting the original project.

RO:

- Open a web browser and navigate to https://github.com/EmmettMyers/Git-Lab.
- Click Fork in the upper-right corner to create a copy of the repository under your GitHub account.
- Once the fork is created, click the Code button.
- Under the Clone section, click Open with GitHub Desktop.
- GitHub Desktop will open, prompting you to choose a local folder to clone the repository.
- Click Clone to download the repository locally.
- Navigate to the forked repository on GitHub and go to Settings > Collaborators. Add your partner using their GitHub username or email.

2. Clone the Repository Locally

Cloning downloads a repository from GitHub to your local machine, allowing you to work on the code offline.

RO & ET:

- Open GitHub Desktop.
- Click Clone a repository from the Internet.

- Select the forked repository and choose a local folder.
- Click Clone.

3. Understanding the Repository

- Look through the repository to gain an understanding of its main functionalities.
- The repository consists of the following key components:
    - `src/git_lab.py`: The main source code file where functions are implemented. You will modify this file to add new functions and fix errors.
    - `tests/test_git_lab.py`: The test file containing unit tests for the functions in `git_lab.py`. You will add test cases here to verify correctness.
    - `.gitignore`: A text file that tells Git which files or directories to ignore when tracking changes in your repository. This is useful for excluding files that are generated automatically (like `__pycache__`) or environment-specific files.

4. Creating a README File

A *README* file provides an overview of the project, often including instructions, descriptions, or usage guidelines.

RO & ET:

- In GitHub Desktop, click Repository > View on GitHub.
- Click Add a README.
- Write a description of this lab and format the text using Markdown syntax.
- Commit the changes directly to the `main` branch.

5. Adding a New Function and Test

RO:

- Open Visual Studio Code and open the cloned repository folder.
- Navigate to the `src` folder and open `git_lab.py`.
- Add the following function to `git_lab.py`:

```
def add_numbers(a, b):
    """Returns the sum of two numbers."""
    return a + b
```

- Navigate to `tests/test_git_lab.py` and add a test for the new function:

```
import unittest
```

```python
from src.git_lab import add_numbers

class TestGitLab(unittest.TestCase):
    def test_add_numbers(self):
        self.assertEqual(add_numbers(2, 3), 5)
        self.assertEqual(add_numbers(-1, 1), 0)

if __name__ == "__main__":
    unittest.main()
```

- Save the files.
- In GitHub Desktop, select Changes, enter a commit message like "Add add_numbers function and test."
- Click Commit to main and then Push origin to upload the changes.

6. Working Collaboratively (Introducing an Error)

ET:

- Open GitHub Desktop and click Fetch origin.
- Click Pull origin to get the latest changes.
- Open VS Code, navigate to `src/git_lab.py`, and modify `add_numbers` incorrectly:

```python
def add_numbers(a, b):
    """Returns the incorrect sum of two numbers."""
    return a - b  # Introduces an error
```

- Save the file.
- Run the tests by executing: `python test_git_lab.py`
- The test should fail.
- In GitHub Desktop, commit and push the changes with a message like "Modify add_numbers incorrectly (intentional error)."

Good Commit Message Examples:

- `feat: Add user authentication` (Uses a type prefix and is concise)
- `fix: Resolve issue with incorrect date formatting` (Uses a type prefix and is specific)
- `feat(api): Implement endpoint for retrieving user data` (Scope added for more detail)
- `Fix #123: Correct calculation of total price in cart` (References an issue number)

Bad Commit Message Examples:

- `Fixed bug` (Too vague - what bug?)
- `Updated code` (No description of what was updated)
- `Added some stuff` (Vague and unhelpful)
- `Minor changes` (Doesn't convey the specific changes)

7. Handling Merge Conflicts and Fixing the Error

A *merge conflict* occurs when two people make conflicting changes to the same file.

RO:

- Pull the latest changes from the repository.
- Open `src/git_lab.py` and modify `add_numbers` to fix the issue:

```python
def add_numbers(a, b):
    """Returns the correct sum of two numbers."""
    return a + b
```

- Commit and push the changes.

ET:

- Modify `add_numbers` in a different way (e.g., multiplying instead of adding).
- Commit and push the changes.
- A merge conflict will occur.
- Pull the latest changes and resolve the conflict in VS Code:
    - Look for `<<<<<<<`, `=======`, and `>>>>>>>` markers in `git_lab.py`.
    - Edit the file to keep the correct version and remove the conflict markers.
    - Save the file.
- Go back to GitHub Desktop, enter a commit message like "Resolve merge conflict in git_lab.py."
- Click Commit to main and then Push origin.

8. Creating and Merging Branches

Branches allow you to work on new features or fixes without affecting the main codebase.

ET:

- In GitHub Desktop, click Branch > New Branch... and name it `new-feature`.
- Modify `git_lab.py` to add another function of your choice.
- Add a corresponding test in `tests/test_git_lab.py`.
- Commit changes to `new-feature`.

- Push the branch to GitHub.
- In GitHub, navigate to the repository and create a pull request for `new-feature`.

RO:

- Review the pull request changes and provide feedback.
  - Add a few comments in the pull request before approving.
- Approve and merge the pull request into `main`.
- Delete the `new-feature` branch after merging.

9. Advanced Merge Conflicts

RO:

- Open the repository in VS Code.
- Navigate to the src folder and delete the `git_lab.py` file.
- Commit the changes with a message like "Delete git_lab.py to refactor code."
- Push the changes to the `main` branch.

ET:

- Open the repository in VS Code.
- Navigate to the `src` folder and modify the `git_lab.py` file by adding a new function:

```
def multiply_numbers(a, b):
    """Returns the product of two numbers."""
    return a * b
```

- Add a corresponding test in `tests/test_git_lab.py`:

```
def test_multiply_numbers(self):
    self.assertEqual(multiply_numbers(2, 3), 6)
    self.assertEqual(multiply_numbers(-1, 1), -1)
```

- Commit the changes with a message like "Add multiply_numbers function and test."
- Attempt to push the changes to the `main` branch.
  - You should encounter a merge conflict because the `git_lab.py` file was deleted by RO but modified by ET.
- In GitHub Desktop, pull the latest changes to see the conflict.
- Resolve the conflict by deciding whether to keep the deleted file or restore it:
  - If you want to keep the changes made by ET, restore the `git_lab.py` file.
  - If you want to proceed with the deletion, confirm the deletion.

- Commit the resolved changes with a message like "Resolve file deletion conflict in git_lab.py."
- Push the resolved changes to the `main` branch.

10. Reverting Commits

Reverting undoes a specific commit, allowing you to roll back changes without losing history.

RO:

- Open the repository in GitHub Desktop.
- Go to the History tab and locate the commit where `git_lab.py` was deleted.
- Right-click the commit and select Revert this Commit.
    - This will create a new commit that undoes the changes made in the selected commit.
- Commit the revert with a message like "Revert deletion of git_lab.py."
- Push the changes to the `main` branch.

Submission

Submit the link to your forked GitHub repository on Canvas. Each partner should submit individually.