Lesson 3: Mastering Git & GitHub with Team Collaboration on the Expense Tracker Project

Objective: Learn the git commands and GitHub workflows necessary to effectively lead and collaborate on a team project.

The Fun Example: Building a Treehouse (Your Expense Tracker Project)

Imagine you're building a treehouse where each room serves a different purpose, much like the files in your expense tracker project. As the team lead, you're in charge of the blueprint, while your friends, Alice and Bob, are helping you build.

Team Members:

- **Team Lead**: You
- **Member 1**: Alice (working on the database connections)
- **Member 2**: Bob (adding new utility functions)

Project Files:

- `app.py`: The main application file, like the entrance to the treehouse.
- `utility.py`: Utility functions, like the toolbox in the treehouse.
- `src/db_connection.py`: Database connection setup, like the treehouse's supply room.
- `src/db_ops.py`: Database operations, like the control room for supplies.
- `expense_ops.py`: Expense operations, like the room where all the expense tracking happens.

1. Getting Started:

- **git init**: You've laid the foundation of the treehouse. In your project folder:

  ```
  git init
  ```

- **git clone**: Alice and Bob clone the repository to start contributing:

  ```
  git clone https://github.com/yourusername/expense-tracker.git
  ```

2. Organizing the Work (Branching and Staging):

- You create a new branch for the main application enhancements:

  ```
  git branch feature/main-layout
  git checkout feature/main-layout
  ```

- Alice creates a branch for the database connection improvements:

  ```
  git branch feature/db-connection
  git checkout feature/db-connection
  ```

- Bob creates a branch for adding utility functions:

  ```
  git branch feature/utility-enhancements
  git checkout feature/utility-enhancements
  ```

3. Building Together (Committing and Pushing):

- You make changes to `app.py` and commit:

  ```
  git add app.py
  git commit -m "Update main app layout"
  git push origin feature/main-layout
  ```

- Alice updates `src/db_connection.py` and commits her changes:

  ```
  git add src/db_connection.py
  git commit -m "Improve database connection setup"
  git push origin feature/db-connection
  ```

- Bob adds new functions to `utility.py` and commits:

  ```
  git add utility.py
  git commit -m "Add new currency conversion utilities"
  git push origin feature/utility-enhancements
  ```

4. Merging Work (Pull Requests):

- You review Alice's and Bob's pull requests on GitHub and merge them into the main branch after ensuring there are no conflicts.

5. Handling Overlaps (Merge Conflicts):

- Alice and Bob both made changes to `expense_ops.py` to add different features. A merge conflict arises when you try to merge their branches.
- You all come together to decide which changes to keep and update the file accordingly:

  ```
  # After pulling the latest main branch and switching to the conflicting branch
  git merge main
  # Manually resolve conflicts in expense_ops.py
  ```

```
git add expense_ops.py
git commit -m "Resolve merge conflict in expense_ops.py"
git push origin feature/conflicting-branch
```

6. Tracking Progress (Git Log and Status):

- To see the history of changes and who contributed what:

  ```
  git log
  ```

- To check the current status of the working directory and staging area:

  ```
  git status
  ```

7. Saving Work for Later (Stashing):

- Bob needs to switch tasks before completing his current work on `utility.py`. He uses stashing to save his work temporarily:

  ```
  git stash
  # Work on something else
  git stash apply
  ```

8. Leading the Team Project:

- As the team lead, you make sure everyone is aligned with the project goals. You hold daily stand-ups to discuss progress and plan.
- You encourage good commit messages, frequent commits, and pull requests for code review.
- You use GitHub's project boards to track tasks and issues to manage the team's workload.

Fun Example for Complex Topic - Merge Conflicts: Imagine the treehouse has a rope swing and a slide that both Alice and Bob built, but they accidentally put them in the same spot. You come together and decide to combine them into a super fun rope-slide. You take the thrill of the swing and the speed of the slide to make a new feature everyone loves.

Demo Code Example for the Expense Tracker Project:

- Alice needs to fix a bug in `src/db_ops.py` where expenses aren't saving correctly:

  ```
  git checkout -b fix/db-saving-bug
  # Alice fixes the bug in src/db_ops.py
  git add src/db_ops.py
  ```

```
git commit -m "Fix bug where expenses aren't saving"
git push origin fix/db-saving-bug
```

- You review her pull request, ensure it doesn't conflict with any other changes, and merge it into the main branch.

By mastering these Git and GitHub concepts and following this collaborative workflow, you and your team will be able to build an amazing expense tracker project, just like how a well-coordinated team builds a fantastic treehouse.