2-Hour Git Collaboration Tutorial

Objective:

By the end of this tutorial, you will be able to:

- Initialize a Git repository and stage/commit changes.
- Create and manage branches for collaborative development.
- Collaborate using GitHub, including pull requests and resolving merge conflicts.
- Apply best practices for version control in a team setting.

Materials Needed:

- Git installed on all machines.
- GitHub account.
- A simple Python script (e.g., a calculator or a to-do list) that can be incrementally improved.

Tutorial Outline

1. Introduction (10 mins)

Objective: Set the stage for the importance of Git and collaboration.

Activity:

- Briefly explain Git and GitHub.
- Show a real-world example of collaboration (e.g., open-source projects like Django or Flask).
- Explain the Python project they will work on (e.g., a simple calculator that will be incrementally improved).

2. Setting Up Git and GitHub (15 mins)

Objective: Ensure everyone has Git set up and can connect to GitHub.

Activity:

```Git
# Step 1: Verify Git installation
git --version

# Step 2: Configure Git with username and email
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

```GitHub
# Step 3: Create a new repository on GitHub
1. Go to GitHub.com and log in.
2. Click the "+" icon in the top-right corner and select "New repository."
3. Name your repository (e.g., "calculator-project").
4. Do NOT initialize the repository with a README or .gitignore.
5. Click "Create repository."
```

```Git
# Step 4: Clone the repository locally
git clone https://github.com/your-username/your-repo-name.git
```

3. Initializing the Python Project (15 mins)

Objective: Create a simple Python project and commit it to Git.

Activity:

```Python
# Step 1: Create a simple Python script (e.g., calculator.py) with basic functionality
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

if __name__ == "__main__":
    print("Add:", add(5, 3))
    print("Subtract:", subtract(5, 3))
```

```Git
# Step 2: Initialize the Git repository
git init

# Step 3: Stage and commit the initial code
git add calculator.py
git commit -m "Initial commit with basic calculator functions"
```

```GitHub
# Step 4: Push the code to GitHub
1. Go to your GitHub repository.
2. Copy the remote repository URL (e.g., https://github.com/your-username/your-repo-name.git).
3. Run the following commands in your terminal:
   git remote add origin https://github.com/your-username/your-repo-name.git
   git push -u origin main
```

4. Branching and Collaborative Development (30 mins)

Objective: Introduce branching and simulate collaboration.

Activity:

```Git
# Step 1: Create a new branch for a feature (e.g., multiplication)
git checkout -b feature-multiplication
```

```Python
# Step 2: Add a new function to the Python script
def multiply(x, y):
    return x * y

if __name__ == "__main__":
    print("Multiply:", multiply(5, 3))
```

```Git
# Step 3: Commit the changes
git add calculator.py
git commit -m "Added multiplication function"

# Step 4: Push the branch to GitHub
git push -u origin feature-multiplication
```

```Git
# Step 5: Merge the feature branch into main
1. Switch back to the main branch:
   git checkout main

2. Pull the latest changes from the main branch:
   git pull origin main

3. Merge the feature branch into main:
   git merge feature-multiplication

4. Push the merged changes to GitHub:
   git push origin main
```

5. Handling Merge Conflicts (20 mins)

Objective: Simulate and resolve a merge conflict.

Activity:

```Git
# Step 1: Have two classmates work on different branches (e.g., one adds division, another adds exponentiation).

# Step 2: Both classmates modify the same line in calculator.py (e.g., the if __name__ == "__main__": block).

# Step 3: Attempt to merge the branches and encounter a conflict.

# Step 4: Resolve the conflict manually by editing the file, then commit the resolved changes:
git add calculator.py
git commit -m "Resolved merge conflict"

# Step 5: Push the resolved changes to GitHub.
git push origin main
```

6. Best Practices and Wrap-Up (20 mins)

Objective: Recap key concepts and discuss best practices.

Activity:

- Discuss best practices for Git collaboration:
  - Write meaningful commit messages.
  - Use `.gitignore` to exclude unnecessary files.
  - Regularly pull changes from the main branch to avoid conflicts.
- Recap the key commands:
  - `git init`, `git add`, `git commit`, `git push`, `git pull`
  - `git branch`, `git checkout`, `git merge`
- Q&A session to address any questions or concerns.

Homework/Follow-Up (Optional)

Objective: Reinforce learning through practice.

Activity:

- Clone a public repository on GitHub, make a small change, and submit a PR.
- Explore GitHub's collaboration features (e.g., issues, code reviews).

Resources

- **Git Documentation:** https://git-scm.com/doc
- **GitHub Guides:** https://guides.github.com/
- **Pro Git Book:** https://git-scm.com/book/en/v2