GitHub Workflow

1. Initialise a Repository

Start with a new folder for the project.

```
bash
Copy code
# Step into a demo directory and create a new folder
mkdir mnist-demo
cd mnist-demo

# Initialize Git in this folder
git init
```

Explain: This creates a local Git repository.

2. Add Initial Files

Set up the project with a basic structure and some initial files.

```
bash
Copy code
# Create a README file
echo "# MNIST Library" > README.md

# Create a directory for source files
mkdir src

# Add a basic Python script
echo "print('Hello, MNIST!')" > src/main.py
```

3. Stage and Commit Files

Add the new files to the Git staging area and commit them.

```
bash
Copy code
# Stage all changes
git add .

# Commit the changes with a meaningful message
git commit -m "Initial commit: added README and main scrip
t"
```

> Explanation: Staging adds files to a "waiting area," and committing saves these changes to the repository history.

---

4. Connect to GitHub

Create a repository on GitHub (e.g., `mnist-demo`) and link it to your local project.

```
bash
Copy code
# Add the GitHub repository as a remote
git remote add origin https://github.com/your-username/mnis
t-demo.git

# Rename the default branch to 'main' (if necessary) and pu
sh the changes
git branch -M main
git push -u origin main
```

> Explanation: This step connects your local Git repository to a remote repository on GitHub, making it accessible online.

---

5. Add a New Feature

Simulate adding a feature to preprocess the MNIST dataset.

```
bash
Copy code
```

```
# Create a new Python script for preprocessing
echo "def preprocess(data):\n    return data / 255.0" > sr
c/preprocess.py

# Stage and commit the changes
git add src/preprocess.py
git commit -m "Added preprocess function for MNIST"
```

6. Create and Use a Branch

Simulate collaboration by creating a new branch for dataset visualization.

```
bash
Copy code
# Create and switch to a new branch
git checkout -b feature-dataset-visualization

# Add a visualization function in a new script
echo "def visualize_sample(sample):\n    print('Visualizin
g:', sample)" > src/visualize.py

# Stage and commit the new feature
git add src/visualize.py
git commit -m "Added dataset visualization feature"
```

> Explanation: Branching allows developers to work on features independently without affecting the main branch.

7. Merge the Branch

Integrate the visualization feature into the main branch.

```
bash
Copy code
# Switch back to the main branch
git checkout main
```

```
# Merge the feature branch into the main branch
git merge feature-dataset-visualization

# Push the changes to GitHub
git push
```

---

8. Simulate and Resolve a Conflict

Simulate a merge conflict by editing the same file in two branches.

1. **On the `main` branch:**

   ```
   bash
   Copy code
   # Modify the main script
   echo "print('Processing data...')" >> src/main.py

   # Stage and commit the change
   git add src/main.py
   git commit -m "Added processing message"
   ```

2. **On a new branch:**

   ```
   bash
   Copy code
   # Create and switch to a new branch
   git checkout -b feature-add-logging

   # Modify the same file
   echo "print('Logging initialized')" >> src/main.py

   # Stage and commit the change
   git add src/main.py
   ```

```
git commit -m "Added logging message"
```

3. **Attempt to merge back into `main`:**

```
bash
Copy code
# Switch to the main branch
git checkout main

# Attempt to merge
git merge feature-add-logging
```

> Explanation: This will create a merge conflict because the same lines in src/main.py were edited in both branches.

---

9. Resolve the Conflict

Manually resolve the conflict and complete the merge.

1. Open the conflicting file (`src/main.py`) in a text editor and merge the changes:

```
plaintext
Copy code
<<<<<<< HEAD
print('Processing data...')
=======
print('Logging initialized')
>>>>>>> feature-add-logging
```

   Resolve by combining both lines:

```
python
Copy code
print('Processing data...')
```

```
print('Logging initialized')
```

2. Stage and commit the resolved file:

```
bash
Copy code
git add src/main.py
git commit -m "Resolved merge conflict in main.py"
```

3. Push the updated branch:

```
bash
Copy code
git push
```

> Tip: This demonstrates how to resolve and finalize merge conflicts.

---

10. Wrap Up

Conclude the demo with the following:

- **Pull Requests**: Show how to create a pull request on GitHub.
- **Collaboration**: Explain how teammates pull changes using `git pull` and resolve conflicts.
- **Logs and History**: Demonstrate how to view the commit history using:

```
bash
Copy code
git log
```