Detailed Git Workflow Explanation

Context: What Are You Doing?

You're practicing branching strategies, file/folder handling, and version tagging with Git in a local project and pushing it to GitHub. These actions simulate real-world release workflows that DevOps engineers and developers follow.

Step 1: Working With Folders and Files

Git doesn't track empty folders — it only tracks files inside them. After creating folders and adding a file to only one, Git recognized only the folder with the file.

Step 2: Committing File Additions

You staged and committed the new file using 'git add .' and 'git commit -m'. This created a snapshot including only the folder with a file.

Step 3: Deleting Files and Committing the Deletion

You deleted a file and committed the deletion. This updates the Git history, ensuring the action is recorded and traceable.

Step 4: Viewing Commit History

Using 'git log' shows SHA, author, message, and tag info. Essential for traceability and audit.

Step 5: Creating Tags for Releases

Tags mark important points in history like release versions. Created using 'git tag Release_V.1.2.0'. These are local until pushed.

Step 6: Pushing Tags to Remote

Tags are pushed using 'git push origin <tag>'. This shares them with your remote (e.g., GitHub), making them available for collaboration and CI/CD.

Step 7: Checking Out a Tag

Switching to a tag places you in a detached HEAD state. You can browse or create a branch from here.

Step 8: Committing Feature Changes and Pushing

New files are added, committed, and pushed. Each commit is logged and traceable.

Step 9: Tagged Commits in Log

Tagged commits show in 'git log' decorated with tag names. Helps identify which release includes which commit.

Best Practices

- Use semantic versioning (v1.2.0)

- Write clear commit messages

- Include .gitkeep in empty folders

- Use detached HEAD only for browsing or hotfixing

- Tag after UAT is passed