==

Purpose

This document provides a comprehensive overview of Git collaboration workflows, with examples and step-by-step instructions for using Git commands in a team setting. It is based on DevOps Session 03 notes and aims to help beginners understand the practical aspects of working with Git repositories, both local and remote.

Git Collaboration and Commands - DevOps Session 03

1. Git Collaboration Model

The diagram illustrates the collaboration flow between two developers (Tejaswini and Rehnuma) working on a shared project using Git. The central element is the remote repository, which acts as a collaboration hub. Each developer clones the remote repo, makes changes locally, and pushes those changes back to the remote repository.

Key Concepts:

- - Remote Repo: Central repository for collaboration.
- - Tejaswini and Rehnuma: Developers who clone the remote repo.
- - Commands: git clone, git pull, git push, git add, git commit.

2. Initial Setup and Token Usage

Steps to save token and push to repository:

- - Create a repository.
- - Create and name folder as per user (e.g., username_model).
- - First day: Only run git init.
- - Second onwards: Use git clone.

3. Git Commands for Collaboration

Example workflow between two developers:

Developer Rehnuma:

- - touch javafile
- - git add -A

- - git commit -m "Added javafile"
- - git push

Developer Tejaswini:

- - git pull
- - git add -A
- - git commit -m "1st commit by Tej"
- - git push

Subsequent pushes alternate between developers with pull, add, commit, push steps.

4. Core Git Commands Summary

Basic workflow for version control with Git:

1. Add changes to staging: git add -A

2. Save changes locally: git commit -m "your message"

3. Upload changes to remote: git push

To get latest changes from remote: git pull

Git Add Command Variations Explained

====================================

1. git add <filename>

- Purpose: Adds a specific file to the staging area.

- Use Case: When you only want to commit changes to one particular file and ignore the rest.

- Example:

```
  git add index.html
```

- Effect: Only 'index.html' will be staged for the next commit.

2. git add <foldername>/

- Purpose: Adds all the files inside the specified folder (including subfolders) to the staging area.

- Use Case: When you made changes in a specific directory and want to stage everything inside it.

- Example:

```
git add src/
```

- Effect: All files under 'src/' will be staged.

3. git add .

- Purpose: Adds all new and modified files in the current directory and subdirectories to the staging area.

- Use Case: When you want to stage all changes from your current working location downward.

- Example:

```
git add .
```

- Effect: Will not stage deleted files.

4. git add -A

- Purpose: Stages all changes in the entire working directory, including:

    - New files

    - Modified files

    - Deleted files

- Use Case: When you want to stage everything—no matter where you are in the repo.

- Example:

```
git add -A
```

- Effect: Stages everything across the repo, including deletions.

Summary Table:

Command	Adds New Files	Adds Modified Files	Adds Deleted Files	Scope
git add <file>	Yes	Yes (if that file)	No	Specific file
git add <folder>	Yes	Yes	No	Specific folder
git add .	Yes	Yes	No	Current dir and below
git add -A	Yes	Yes	Yes	Whole working directory