Day 1 Git Session
21 September 2023
09:55

Git Introduction

Git is a distributed version control system (DVCS) that is widely used in software development to track changes in source code and coordinate collaboration among multiple developers or teams. It was created by Linus Torvalds in 2005 and has since become the de facto standard for version control in the software development industry.
Certainly, here are some short key points about Git:

- **Version Control System (VCS):** Git is a VCS that tracks changes in source code and project files.
- **Distributed:** Each developer has a complete copy of the project's history on their local machine.
- **Branching:** Developers can create branches for separate lines of development.
- **Merging:** Git allows merging changes from one branch into another.
- **Collaboration:** Multiple developers can work on the same project, sharing changes via remote repositories.
- **History:** Git maintains a detailed history of all changes made to a project.
- **Remote Repositories:** Git supports hosting repositories on remote servers or platforms.
- **Security:** Git provides authentication and authorization mechanisms.
- **Open Source:** Git is open-source and widely used in software development.
- **Extensible:** Git can be customized with scripts and tools to enhance functionality.

Git is essential for code management, collaboration, and maintaining software project history.

Git Commands

1.Initialize a New Git Repository: To start tracking changes in a project, you can initialize a Git repository in the project's directory:
~Use the Git Command **-git init**
2.**Clone a Remote Repository:** To create a local copy of a remote repository, use the git clone command:
~git clone<repository_url>
3.**Check Status:** To see the status of your working directory and staged changes, use: **git status**
4.**Add Changes to Staging Area:** Use git add to stage changes for the next commit: **git add <file_name>**
5.**Commit Changes:** Commit staged changes with a descriptive message:
git commit -m "Your commit message here"
6.Create a New Branch: Create a new branch for a new feature or bug fix:
git branch <branch_name>

7.Switch Branches: Move to a different branch:
git checkout <branch_name>
8.Merge Branches: Merge changes from one branch into another:
git merge <branch_name>
9.View Commit History: See the commit history with:
git log
10.Add a Remote Repository: Add a remote repository to collaborate with others:
git remote add <remote_name> <repository_url>
11.Fetch Remote Changes: Fetch changes from a remote repository:
git fetch <remote_name>
12.Pull Remote Changes: Fetch and merge remote changes into your current branch:
git pull <remote_name> <branch_name>
13.Push Changes to Remote: Share your local changes with the remote repository:
git push <remote_name> <branch_name>
14.Discard Local Changes: Discard changes in your working directory:
git checkout -- <file_name>
15.Remove Untracked Files: Remove untracked files from your working directory:
git clean -f
Stashing:
In Git, **"stashing"** refers to the process of temporarily saving changes in your working directory that you're not ready to commit yet. It's a way to set aside your current changes so that you can switch to a different branch, address an urgent issue, or perform other tasks without committing incomplete work. Stashing allows you to work on different aspects of your project without cluttering your commit history with unfinished changes.

Here's how the stashing process works in Git:

1. **Stash Changes:**
To stash your uncommitted changes, you can use the following command:**git stash**
~ This command takes all the changes in your working directory that are not yet committed and saves them in a special area known as the "stash." **You can also provide a descriptive message along with the stash, like this:**
Commnd~ git stash save "Your stash message"

2. **Switch to a Different Branch or Perform Tasks:**
Once you've stashed your changes, you can switch to a different branch or perform other tasks without your changes interfering. For example, you can create a new branch, switch to an existing one, or address an urgent bug or task.
~ **git checkout <branch_name>**

3. ****Retrieve Stashed Changes:****
When you're ready to work on the changes you stashed, you can apply them back to your working directory using the following command: **~ git stash apply**
- This command will apply the most recent stash. If you have multiple stashes and want to apply a specific one, you can specify it by name: **git stash apply stash@{1}**

4. **Delete a Stash (Optional):**
If you no longer need a particular stash, you can delete it with the following command:

git stash drop stash@{1}

This will remove the specified stash from your stash list.

5. **List Stashes:**
You can list all your stashes to see what you've stashed and their respective names using: **git stash list**

Stashing is a valuable feature in Git that allows you to work on different aspects of your project without losing your work or polluting your commit history with incomplete changes. It's particularly useful when you need to switch contexts quickly or address urgent tasks while maintaining a clean and organized Git history.

Command Name	Command	Use
List File and Directories	ls	This command lists the files and directories in the current directory. >
Change Directory	cd <directory_name>	Use cd to change the current directory.
Create a Directory:	mkdir <directory_name>	This command creates a new directory with the specified name.
Remove a File:	rm -rf<file_name>	To delete file
Display File Contents:	cat <file_name>	Use cat to display the contents of a file.

Show inside hidden folder	ls -a	Show inside hidden folder
Create new file	touch filename.extension	To add new file
Add file in branch	Git add <filename>	To add file in branch
Git init	Initilize github repo	
Vi file name	Open file in vim editor	
Unstage changes	Git rm --cached <filename>	
Remote access any repo	Git remote add origin <url>	
for commit	Git commit -m "message"	