# **Version control system[distributed]**

# [Git - Introduction](https://itdefined.org/course/training-material/2/23/764/)

Git is a distributed version control system used for tracking changes in source code during software development.

**Remote Repository:**

A remote repository is a Git repository hosted on a remote server or a code hosting platform like GitHub, GitLab, or Bitbucket.

It acts as a centralized location for collaborating changes from multiple developers, where multiple developers can push and pull changes to keep the repository in sync.

We can not make any changes directly to a remote repository and a valid user with proper access can push changes to remote repo from only local repo.

**Local Repository:**

A local repository is a copy of a Git repository that resides on your local machine.

It contains the complete copy of the repo repository, including all branches, commits, files etc.

The local repository allows you to make changes, commit them, switch between branches, and perform Git operations without needing a live network connection to remote repo.

You can create a local repository by initialising a new repository with git init or cloning an existing repository using git clone.  
   
 #To get a local repo / to download/clone a remote repo   
     git clone <url\_of\_the\_remote\_repository>

**git status**

The git status is used to display the current status of your Git repository.

Shows information about which files have been modified, which files are staged for the next commit, and which files are untracked.

**git log**

git log is used to view all commits history of a Git repository.

It displays a list of commits starting from the most recent commit on top.

The commit history provides essential information about the changes made to the repository such as,   
     commit messages, authors, dates, and unique identifiers (commit hash/id).

**git add**

Adds changes to the staging area.

This is where you can review your changes before committing them to your local repository.  
 # Add all changes to the staging area  
   git add .   (OR)   git add --all  
     
   git add -A is equivalent to git add --all  
   git add -u is equivalent to git add --update  
     
   NOTE: Extra information just in case you want to try but not required  
   Command            New-Files        Modified-Files    Deleted-Files                                         Description  
     git add --all            YES                          YES                      YES                       Stage all (new, modified, deleted) files  
     git add .                 YES                          YES                      YES                       Stage all (new, modified, deleted) files in current folder  
     git add -u                NO                          YES                      YES                       Stage modified and deleted files only  
         
**git commit**

Create a new commit containing the current changes of the stagging area (added files and directories)

We need to give a commit message which we can see in the git log message describing the changes.

 # Commit the changes with a message  
     git commit -m "Added new feature"  
       
**git push**

git push is a command used to push your local committed changes (after git add and git commit) to a remote repository.

When you make changes to your local repository and want to share them with others or synchronize them with a remote repository

   (such as on GitHub, GitLab, or Bitbucket), you use git push to send those changes to the remote repository.  
   
**git pull**

git pull is a command used to fetch and merge changes from a remote repository into your local repository.

When others push changes to the remote repository, you can use git pull to update your local repository to get new commits from the remote repository.

**Branch**   
branch is a parallel version of another branch repository.   
It allows multiple users to work on different versions of files & directories simultaneously, isolating the changes you make from one branch to another branch.  
we always create a branch from another branch.

To list all the branches in the local repository: **git branch**

To create a new branch named "development": **git branch development**

To change the current branch: **git checkout <new\_branch>**

To delete a branch named "development": **git branch -d development**

To create a branch and checkout simultaneously: **git checkout -b <new\_branch>**

**Origin (Not required for Interview)**

**Origin in Git:** In Git, "origin" refers to the default alias name for the remote repository (URL) from which a local repository was cloned.

**Cloning Repositories:**When you clone a repository using Git, the cloned repository is linked to the original repository from which it was copied, and this connection is labelled as "origin".

**Push and Pull:**Git uses "origin" as the default alias when pushing changes from a local repository to the remote repository, as well as when pulling changes from the remote repository to the local repository.

**Multiple Remotes:**While "origin" is the default remote name, Git allows users to add multiple remote repositories with custom names to enable collaboration with various remote repositories. (Usually, we never add multiple remotes)

**Checkout**   
used to switch between different branches, and commits.

**Switch to commit:**we can switch to any commit using git checkout   
 git checkout <commit\_it>

**Switching Branches:**It allows you to move to a different branch within a repository.  
git checkput <branch>

**git merge vs git rebase**

Both git merge and rebase are used to combines the changes from one branch into another branch but different approaches to integrating changes from one branch into another in Git

**merge**  
  - Merge creates a new commit which shows the changes combined from both branches.   
  - Merge preserves the history of both branches, so you can see which changes came from which branch.  
  - Merge can sometimes create merge conflicts, we need to discuss with the developers and solve the merge conflicts to complete the merge.  
    (usually merge conflicts occurs if 2 different users try the chagne the same lines in the same file and if we try to merge it)  
    
**rebase**  
  - Rebase do not creates a new commit that combines the changes from both branches.   
  - Rebase rewrites the history of the first branch, it will bring all the commits from source branch and place it to the tip of   
    the destination branch. (Commit history usually will become a linear and cleaner tree)  
  - Rebases do not create merge conflicts, but they can make it difficult to track the history of your repository.

**Revert vs Reset**

**git reset**

**History Rewriting:** git reset is used to erase commits from the project's history, effectively rewriting the commit history.

**Local Changes:** It is commonly used to undo local changes, discarding uncommitted modifications and resetting the staging area and working directory.

**Caution Required:**It is a powerful and potentially dangerous command that can permanently delete commits, so it should be used with caution, especially when working with shared repositories.

**git revert**

**Undo Commit:** git revert is used to create a new commit that undoes the changes made by a specific previous commit.

**Preservation of History:** Unlike git reset, it does not rewrite history. Instead, it creates a new commit that effectively undoes the changes from the specified commit while preserving the project's commit history.

**Safe Undoing:** It is a safer option for reverting changes, especially when working in a shared repository, as it does not alter the existing commit history.

**--hard**: **uncommit + unstage + delete** commits from local repo, nothing left. Need perform a git push --force to apply the same to remote repo

**--mixed** *(default)*: **uncommit + unstage** changes, changes are left in *working tree*.

**--soft**: **uncommit** changes, changes are left staged (*index*).

A diagram of a diagram

Description automatically generated

**git cherry-pink**

used to pick a single commit (or multiple commits) from one branch and apply it onto another branch. This is useful when you want to apply changes introduced by specific commits without merging an entire branch.

git checkout <branch-name>

Identify the commit you want to cherry-pick: git log

git cherry-pick <commit-hash>

**git fetch vs git pull**

git fetch and git pull are both used to update local repository changes from remote repository.

**git fetch**   
 - Brings the changes from a remote repository into your local repository.  
 - Does not automatically merge the changes into your current branch in local repository.  
 - Instead it keeps changes in a temporary branch called FETCH\_HEAD and we can inspect the changes then decide how you want to integrate       them into our local work in local repository.

**git pull**

Automatically git will fetch the changes from remote repo and merge it to the corresponding local branch

pull = fetch + auto merge

**Centralised Version Control System (CVCS):**

Uses a central server to store the entire codebase and version history.

Relies heavily on network connectivity for most operations.

Collaboration involves checking out and checking in files from the central server.

Offers centralised control for enforcing access controls and security measures.

Branching and merging can be more complex compared to DVCS.

**Distributed Version Control System (DVCS):**

Each user has a complete local repository with the full project history.

Allows users to work offline and synchronise changes later.

Facilitates flexible and independent collaboration among developers.

Provides better redundancy and backup capabilities with each user's local copy.

Excels in branching and merging, making it easier to experiment with new features.