GIT:

Git is a distributed, open-source version control system. It enables developers and data scientists to track code, merge changes and revert to older versions-AWS

Created by Linus Torvalds, creator of Linux, in 2005

**Properties:** flexibility and popularity.

**Version control:**

A version control system (short: "VCS") can be thought of as a type of "database." It allows you to save a screenshot of your entire project at any time. Your VCS will show you exactly how an older snapshot (let's call it "version") differed from the previous one when you look at it later.

**Commit:**

It stores your files and their development history in a local repository. Whenever you save changes you have made, Git creates a commit. A commit is a snapshot of current files. These commits are linked with each other, forming a development history graph, as shown below. It allows us to revert back to the previous commit, compare changes, and view the progress of the development project [**Azure DevOps**](https://docs.microsoft.com/en-us/devops/develop/git/what-is-git).

To undo changes on a file before you have committed it: –git reset HEAD --filename (unstagesthe file) –git checkout --filename (undoes your changes) .

To view status of files in working directory and staging area: –git statusor git status –s (short version)

▪To see what is modified but unstaged: –git diff ▪To see a list of staged changes: –git diff --cached ▪To see a log of all changes in your local repo: –git log or git log --oneline(shorter version)

Branches:

The branches are copies of the source code that works parallel to the main version. To save the changes made, merge the branch into the main version. This feature promotes conflict-free teamwork. Each developer has his/her task, and by using branches, they can work on the new feature without the interference of other teammates. Once the task is finished, you can merge new features with the main version (master branch).

Branch Creation**,** Branch Switching, Branch Merging, Branch Deletion, Remote Branches, Branch Strategies.

Note:

To create a commit, you need to first stage changed files. You can add or remove changes in the staging area and then package these changes as a commit with a message describing the changes.

Benefits of git:

* Track changes: It allows developers to view historical changes. Development history makes it easy to identify and fix bugs.
* IDE Integration: Due to its popularity, Git integration is available in all development environments, for example VS Code and Jupyter Lab.
* Team collaboration: A developer team can view their progress, and by using branches, they can work individually on a task and merge changes with the main version. Pull requests, resolving merge conflicts, and code review promote team collaboration.
* Distributed VSC: In a distributed system, there is no centralized file storage. There are multiple backups for the same project. This approach allows developers to work offline and commit changes.

Difference between commit and branch:

Commits are the individual changes that are recorded in the version control system, while branches are a way to organize and isolate different lines of development.

Commits exist within branches, and branches provide a way to manage the flow of changes in a collaborative development environment.

Commits are the changes themselves, and branches are a way to organize and manage the development process by providing a mechanism for isolating and merging changes.

Staging area:

the "staging area" is an intermediate area between your working directory and the repository. It plays a crucial role in managing and organizing changes before they are committed to the repository. The staging area is also commonly referred to as the "index."

Git commands:

* **git init** create a Git repository in a local directory.
* **git clone** <remote-repo-address>: copy the entire repository from a remote server to remote directory. You can also use it to copy local repositories.
* **git add** <file.txt>: add a single file or multiple files and folders to the staging area.
* **git commit** –m “Message”: create a snapshot of changes and save it in the repository.
* **git config** use to set user-specific configurations like email, username, and file format.
* **git status** shows the list of changed files or files that have yet to be staged and committed.
* **git push** <remote-name> <branch-name>: send local commits to remote branch of repository.
* **git checkout** -b <branch-name>: creates a new branch and switches to a new branch.
* **git remote** –v: view all remote repositories.
* **git remote add** <remote-name> <host-or-remoteURL>: add remote server to local repository.
* **git branch** –d <branch-name>: delete the branch.
* **git pull** merge commits to a local directory from a remote repository.
* **git merge** <branch-name>: after resolving merge conflicts the command blends selected branch into the current branch.
* **git log** show a detailed list of commits for the current branch.
* **git stash:**

Stashes changes in the working directory that are not yet committed. This is useful when you want to switch branches or save your changes temporarily.

* **git ignore:**

The correct command is actually gitignore, and it's not a Git command by itself. Instead, it's a file named .gitignore that you create in your Git repository. This file lists files or patterns that Git should ignore.

* **git log:**

Shows the commit logs. It displays information about the commit history, including commit messages, authors, dates, and commit hashes.

* **git index or git staging area:**

The index, also known as the staging area, is where changes are added before committing. It acts as a buffer between your working directory and the committed changes.

* **git cherry-pick:**

Applies changes from a specific commit or range of commits to the current branch.

* The term "HEAD" is a reference to the latest commit in the currently checked-out branch.
* Git pull and git fetch deal with fetching changes from remote repositories, with git pull additionally merging them into the current branch.
* **git fork:**

git fork is not a native Git command. Forking is typically done on Git hosting platforms like GitHub, GitLab, or Bitbucket. It's a process of creating a personal copy of someone else's repository to make changes without affecting the original project.

* **git diff:**

Shows the differences between two sets of changes. It can be used to compare changes between commits, branches, or between the working directory and the staging area.

Difference between centralized and distributed version control systems:

Repository Structure:

In CVC, there is a single central repository.

In DVC, each developer has their own complete repository, and changes can be synchronized between repositories.

Offline Work:

CVC systems often require a network connection to access the central repository for most operations.

DVC systems allow developers to work offline, as they have a local copy of the entire repository.

Branching and Merging:

CVC systems may have limitations and be less flexible in terms of branching and merging.

DVC systems excel in branching and merging, making it easier for developers to work on separate features and merge changes seamlessly.

History Tracking:

In CVC, the entire version history is stored centrally.

In DVC, each repository contains the full version history, providing redundancy and flexibility.

Dependency on Central Server:

CVC systems are dependent on the central server for various operations.

DVC systems allow users to perform most operations locally without relying on a central server.