UNIT I INTRODUCTION TO DEVOPS

Devops Essentials - Introduction To AWS, GCP, Azure - Version control systems: Git and Github.

VERSION CONTROL SYSTEM

Version Control Systems are software tools used in collaborative projects, particularly in software development, to manage and track changes to files over time. It keeps a record of every single change made to the code. It also allows us to turn back to the previous version of the code if any mistake is made in the current version. Without a VCS in place, it would not be possible to monitor the development of the project.

Functionalities of VCS

- **Track changes:** Every modification made to a file is recorded, along with who made it, when, and why.
- **Revert to previous versions:** Easily undo changes or revert a file or the entire project to an earlier state if needed, saving time and effort spent manually undoing mistakes.
- **Compare changes:** Review the differences between versions to identify exactly what was changed.
- **Collaborate effectively:** Facilitate teamwork by allowing multiple developers to work on the same codebase simultaneously without overwriting each other's work.
- **Branch and merge:** Allow developers to create independent lines of development ("branches") to work on new features or bug fixes in isolation, then merge those changes back into the main project.
- **Provide backup and recovery:** Act as a backup system, ensuring that project history can be restored even if a local copy is lost.

Why are VCS important?

VCS are essential for:

- **Ensuring code integrity and quality:** By tracking every change and facilitating reviews, VCS help maintain a consistent and functional codebase.

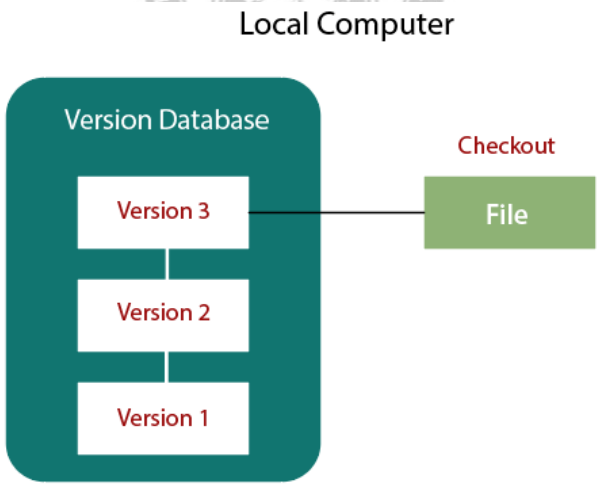
- **Speeding up development:** Automating tasks like testing and deployment, and enabling faster iteration and delivery of new features.
- **Promoting collaboration and communication:** Giving developers a single source of truth and enabling effective communication around changes.
- **Reducing risks:** Preventing loss of work, mitigating errors, and simplifying debugging by allowing developers to easily revert to previous versions.

Types of VCS

There are two main types of version control systems:

1. Local Version Control System

The Local Version Control System is located in your local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost. If anything happens to a single version, all the versions made after that will be lost. Also, with the Local Version Control System, it is not possible to collaborate with other collaborators.



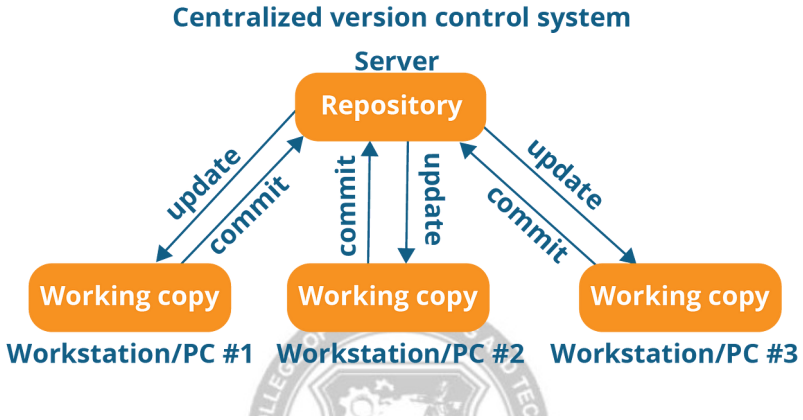
2. Centralized Version Control Systems (CVCS):

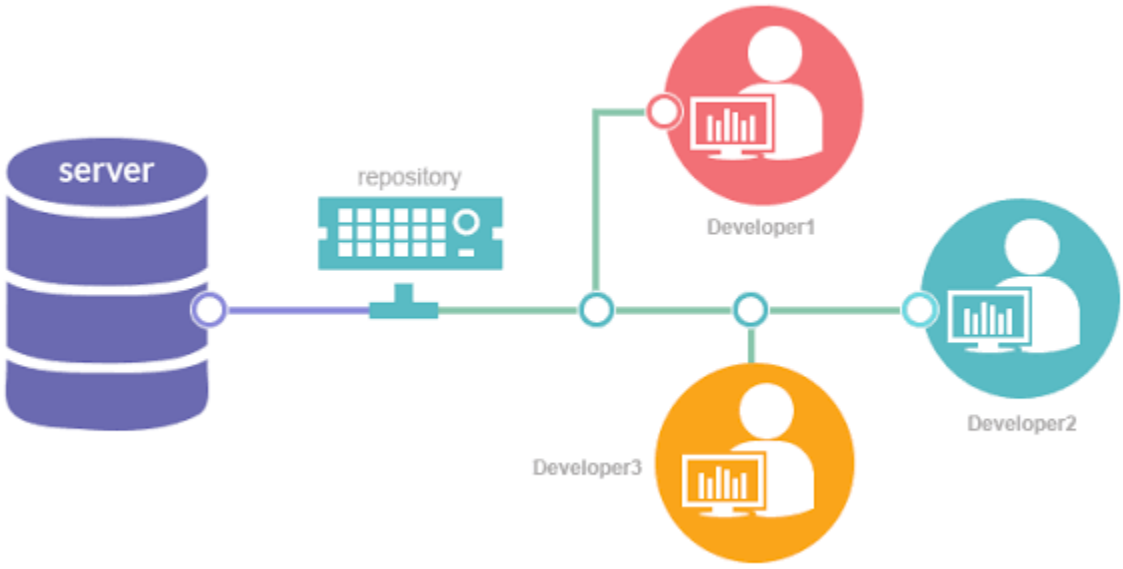
All file versions are stored on a single central server. It contains all the files related to the project, and many collaborators checkout files from this single server (only have a working copy). Developers check out files from the server, make changes, and then check them back in. Administrators manage access and merge the work of collaborators.

Advantages: Simple to set up and manage, suitable for smaller teams or projects with limited collaboration needs.

Disadvantages: Single point of failure (if the server goes down, work halts), slower performance due to constant communication with the server, less flexible in customizing workflows.

Examples: Subversion (SVN)





3. Distributed Version Control Systems (DVCS):

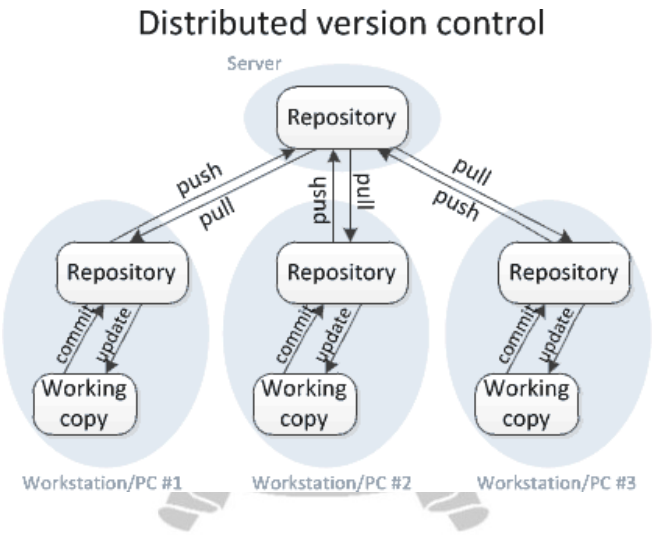
Each collaborator will have an exact copy (mirroring) of the main repository(including its entire history) on their local machines. Developers can work offline, make commits locally, and then push/pull changes to/from a central repository or other users. There will be one or more servers and many collaborators similar to the

centralized system.

Advantages: Allows developers to work offline, better handling of branches and merges, faster performance due to local operations, more resilient as each clone acts as a backup. Hence even if the server crashes we would not lose everything as several copies are residing in several other computers.

Disadvantages: More complex to set up and understand initially, requires more storage space as each developer has a full copy, potentially higher bandwidth usage when syncing with remote servers.

Examples: Git, Mercurial



Popular VCS tools

- **Git:** The most widely used distributed version control system, known for its speed, flexibility, and robust features. It is the backbone of services like GitHub(Microsoft), GitLab (GitLab Inc), and Bitbucket (Atlassians).
- **Subversion (SVN)**: **Apache Subversion (SVN)** is a **centralized version control system (VCS)** used to manage changes to source code and documents over time. It is simpler to learn than Git and favored by some for its centralized control.
- **Mercurial:** Another distributed version control system, similar to Git but often considered to have a simpler interface. Bitbucket used to support Mercurial, but dropped support in 2020, fully switching to Git.

The choice between a centralized and distributed system depends on the project's

specific needs, team size, workflow preferences, and the scale of the project. While DVCSs like Git are widely considered the standard for modern software development due to their flexibility and scalability, CVCSs like SVN remain relevant in some scenarios, particularly for smaller teams or specific enterprise requirements.

Git and GitHub

Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

It is used for:

- Tracking code changes
- Tracking who made changes
- Coding collaboration

What does Git do?

- Manage projects with Repositories
- Clone a project to work on a local copy
- Control and track changes with Staging and Committing
- Branch and Merge to allow for work on different parts and versions of a project
- Pull the latest version of the project to a local copy
- Push local updates to the main project Working with Git
- Initialize Git on a folder, making it a Repository
- Git now creates a hidden folder to keep track of changes in that folder
- When a file is changed, added or deleted, it is considered modified
- You select the modified files you want to Stage
- The Staged files are Committed, which prompts Git to store a permanent snapshot of the files
- Git allows you to see the full history of every commit.
- You can revert back to any previous commit.
- Git does not store a separate copy of every file in every commit, but keeps track of changes made in each

What is GitHub?

- Git is not the same as GitHub.
- GitHub makes tools that use Git.
- GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.

DEFINE repository:

A repository is a directory (aka folder) where you are saving all the code files for any given project. We can make a folder and then use git init to make it into a git repository.

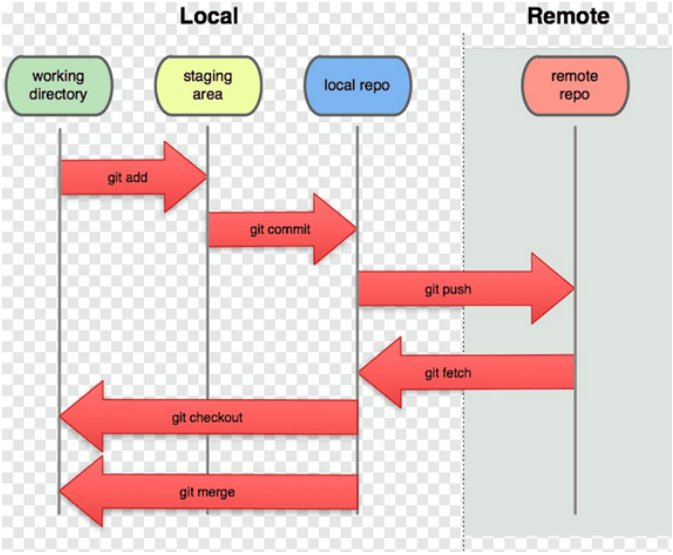
Branching

"On branch master" (means on the main project, not a side branch) to add to main branch use -m to add to side branch use –t

Git Repository Structure

It consists of 4 parts:

1. **Working directory:** This is your local directory where you make the project (write code) and make changes to it.
2. **Staging Area (or index):** this is an area where you first need to put your project before committing. This is used for code review by other team members.
3. **Local Repository:** this is your local repository where you commit changes to the project before pushing them to the central repository on Github. This is what is provided by the distributed version control system. This corresponds to the .git folder in our directory.
4. **Central Repository:** This is the main project on the central server, a copy of which is with every team member as a local repository.



Difference between Git and GitHub

Git is a version control tool (software) to track the changes in the source code. GitHub is a web-based cloud service to host your source code(Git repositories). It is a centralized system. Git doesn't require GitHub but GitHub requires Git.

Installation of Git

There are two ways of installing Git.

1. Install Git for using WSL (Windows Subsystem for Linux) Install Ubuntu – https://www.youtube.com/watch?v=X-DHaQLrBi8

a. Install Git in Ubuntu – https://www.digitalocean.com/community/tutorials/how-to-install-git-on-ubuntu-20-04

2. Install Git software for windows – https://git-scm.com/download/win

a. Go ahead with whichever method is comfortable for you. Continue further once you are done with the installation.

Git operations and commands

Before deep-diving into Git operations and commands, create an account for yourself on GitHub if you don't have it already.

Git Commands: Working With Local Repositories

1. **git init** The command git init is used to create an empty Git repository. After the git init command is used, a .git folder is created in the directory with some subdirectories.

Once the repository is initialized, the process of creating other files begins

Syntax: $git init

2. **git add** Add command is used after checking the status of the files, to add those files to the staging area. Before running the commit command, "git add" is used to add any new or modified files.

   Syntax: $git add

3. **git commit** The commit command makes sure that the changes are saved to the local repository. The command "git commit –m <message>" allows you to describe everyone and help them understand what has happened.

   Syntax: git commit -m "commit message"

4. **git status** The git status command tells the current state of the repository. The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status. Also, if there are no changes, it will show the message no changes to commit, working directory clean.

   Syntax: $git status

5. **git config** The git config command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository. When git config is used with --global flag, it writes the settings to all repositories on the computer.

   Syntax: git config --global user.name "any user name" git config --global user.email <email id>

6. **git branch** The git branch command is used to determine what branch the local repository is on. The command enables adding and deleting a branch.

   Syntax: # Create a new branch

   git branch <branch_name>

   # List all remote or local branches

   git branch –a

   # Delete a branch

git branch -d <branch_name>

7. **git checkout** The git checkout command is used to switch branches, whenever the work is to be started on a different branch. The command works on three separate entities: files, commits, and branches.

   Syntax:

   # Checkout an existing branch

   git checkout <branch_name>

   # Checkout and create a new branch with that name

   git checkout -b <new_branch>

8. **git merge** The git merge command is used to integrate the branches together. The command combines the changes from one branch to another branch. It is used to merge the changes in the staging branch to the stable branch.

   Syntax: git merge <branch_name>

Git Commands: Working With Remote Repositories

1. **git remote** The git remote command is used to create, view, and delete connections to other repositories.

   Syntax: $ git remote add origin <address>

2. **git clone** The git clone command is used to create a local working copy of an existing remote repository. The command downloads the remote repository to the computer. It is equivalent to the Git init command when working with a remote repository.

   Syntax: $ git clone <remote_URL>

3. **git pull** The git pull command is used to fetch and merge changes from the remote repository to the local repository. The command "git pull origin master" copies all the files from the master branch of the remote repository to the local repository.

   Syntax: $git pull <branch_name> <remote URL>

4. **git push** The command git push is used to transfer the commits or pushing the content from the local repository to the remote repository. The command is used after a local repository has been modified, and the modifications are to be shared with the remote

team members.

Syntax: $ git push -u origin master

5. **git branch** is a new/separate version of the main repository.

Accessing Github central repository via HTTPS or SSH

Here, transfer project means transfer changes as git is very lightweight and works on changes in a project. It internally does the transfer by using Lossless Compression Techniques and transferring compressed files. Https is the default way to access Github central repository.

- By git remote add origin http_url: remote means the remote central repository. Origin corresponds to your central repository which you need to define (hereby giving HTTPS URL) in order to push changes to Github.
- Via SSH: connect to Linux or other servers remotely.