Version Control System

What is Version Control ?

Version Control is tracking and managing changes to software code.

Version Control System

A Version Control System (VCS) is a tool that helps track and manage changes to a project's codebase over time.

Why is the Version Control system so Important ?

- Track Changes Over Time

  VCS allow developers to track every modification made to the codebase. This means you can always go back to previous versions, ensuring no changes are lost.

- Collaboration

  VCS simplifies collaboration between team members. Everyone can work on different parts of the project without worrying about overwriting each other's work.

- Code History and Audit Trails

  With version control, you can see who made specific changes, when they were made, and why. This audit trail is invaluable for debugging, reviewing, or maintaining code.

- Backup and Recovery

  Version control systems offer a way to back up your project. If something goes wrong, you can always recover previous versions.

- Branching and Merging

  You can create branches for different features or bug fixes, allowing multiple developers to work simultaneously without interfering with each other's code. Once the work is done, branches can be merged back into the main project seamlessly.

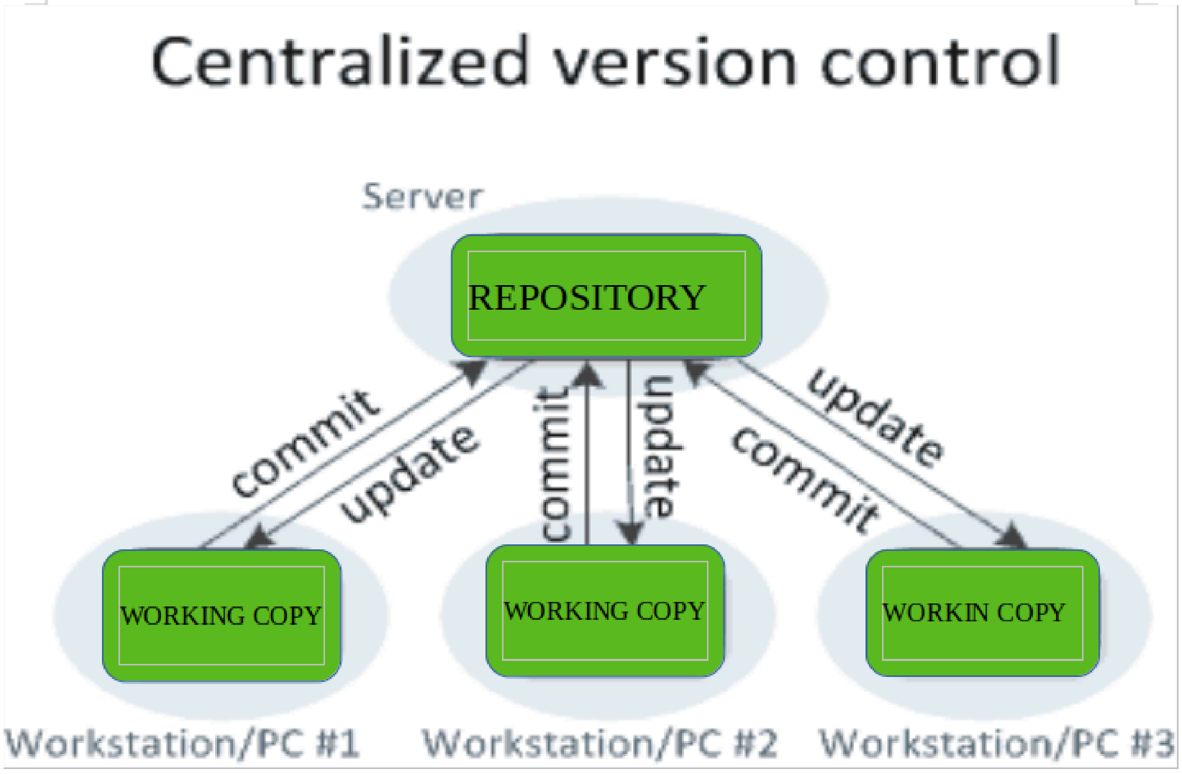
Types of Version Control Systems

01. Centralized Version Control Systems

Centralized version control systems contain just one repository globally, and every user needs to commit for reflecting one's changes in the repository. It is possible for others to see your changes by updating.

Two things are required to make your changes visible to others

- You commit
- They update



02. Distributed Version Control Systems

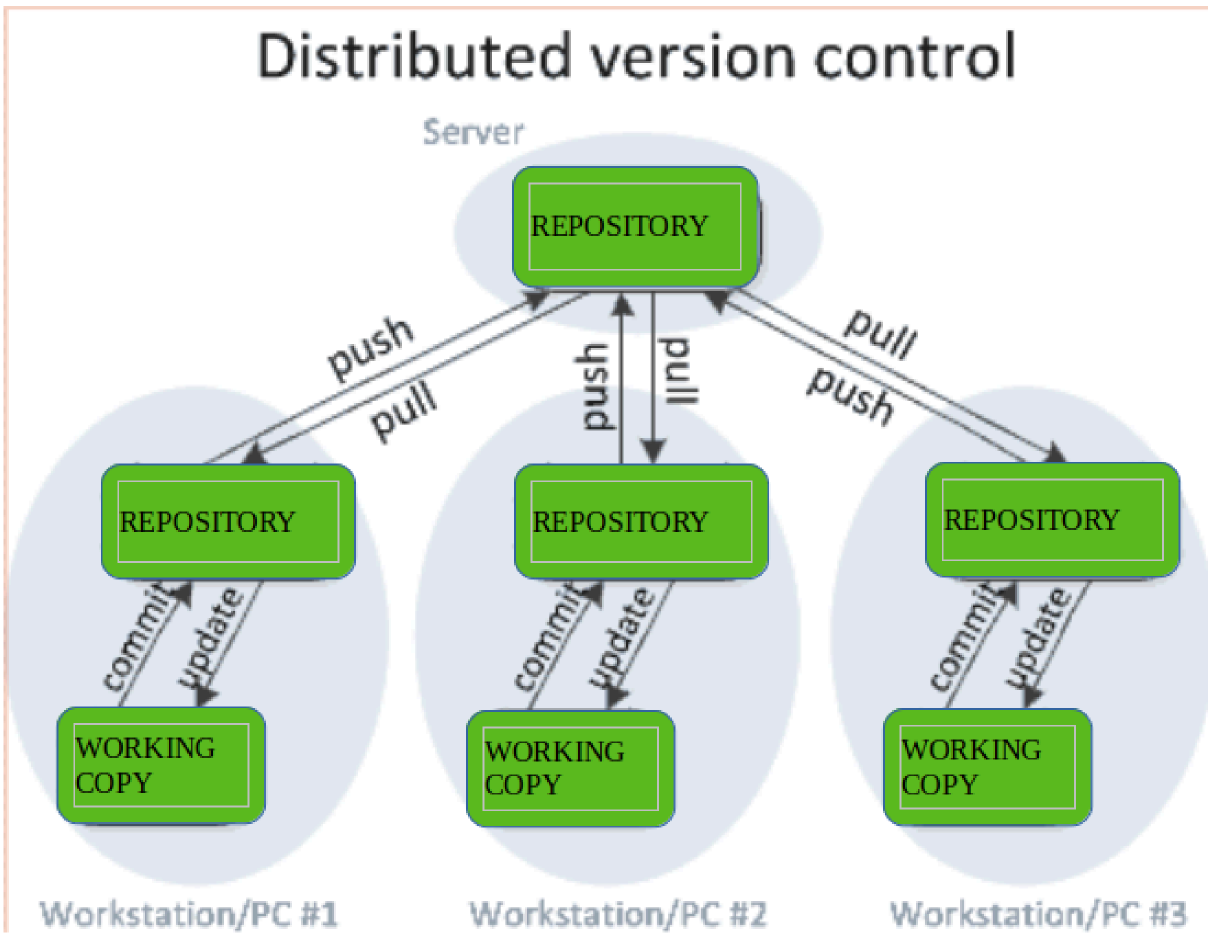
Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commits will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository.

Similarly, When you update, you do not get others’ changes unless you have first pulled those changes into your repository.

To make your changes visible to others, 4 things are required:

- You commit
- You push
- They pull
- They update

The most popular distributed version control systems are Git, and Mercurial



Git & Github

What is Git ?

Git is a popular version control system. It is highly efficient, supports branching and merging, and has a fast, decentralized workflow. Git is the backbone of services like GitHub, GitLab, and Bitbucket, making it a popular choice for developers worldwide.

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

What is GitHub ?

GitHub is a web-based platform built on top of the Git version control system, used for storing, tracking, and collaborating on software projects, making it easy for developers to share code and work together on open-source or private projects.

Git Push

Git push allows us to transfer files from the local repository to remote repository hosting services like GitHub, GitLab, etc. Other developers who want to work on the files can access them after being uploaded to a remote repository.

```
git push <remote> <branch>
```

- The <remote> option refers to the remote repository to which you want to push your files it will refer to its alias name where the name is mapped with the remote repository URL (like origin)

- The <branch> option represents the branch of the GitHub repository which you want to push.

Git Pull

Git pull is basically combination of git merge and git fetch which is used to update the local branch with the changes available in the remote repository branch

```
git pull [remote-name] [branch-name]
```

Git Clone

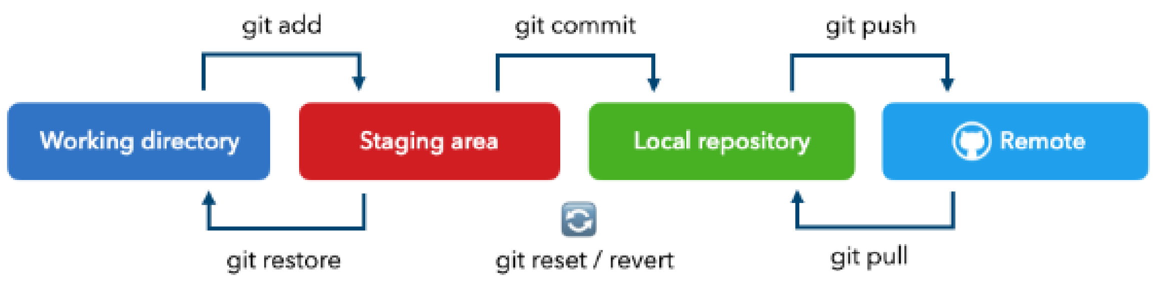
The git clone command creates a copy of an existing Git repository. This repository can be hosted on platforms like GitHub, GitLab, or Bitbucket, or it can reside on a local or remote server. Cloning a repository involves copying all the repository's data to your local machine, including the entire history and all branches.

```
git clone <repository-link>
```

Git Bash Codes

- Make directory
  ```
  mkdir <directoryName>
  ```
- Change directory
  ```
  cd <directoryName>
  ```
- Initialize Git
  ```
  git init
  ```
- Lists files and directories
  - lists files and directories in the current location.
    ```
    ls
    ```
  - lists all files and directories in the current location, including hidden files and directories
    ```
    ls -a
    ```

Git Staging Environment



One of the core functions of Git is the concepts of the Staging Environment, and the Commit.

As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.

Staged files are files that are ready to be committed to the repository you are working on.

- Create a new file inside the directory

  file.txt

- Check git status

  git status

- Stage file

  git add file.txt

  git add has 3 types

  - ➔ Git add . - Add all files
  - ➔ Git add file1.txt file2.txt - Add only chosen files
  - ➔ Git add folderName/ - Add all files in folder

- Unstage file

  git rm –cached file.txt

- Commit changes

  git commit -m <commit message>

- Delete files

  rm <fileName>

  (After removing the file the changes must be staged.)

Git ignore

When sharing your code with others, there are often files or parts of your project, you do not want to share.

like,

- ➔ log files
- ➔ temporary files
- ➔ hidden files
- ➔ personal files

Git can specify which files or parts of your project should be ignored by Git using a .gitignore file.

Git will not track files and folders specified in .gitignore. However, the .gitignore file itself IS tracked by Git.

- Ignore any files with the .log extension

  .log
- Ignore everything in any directory named temp

  temp/

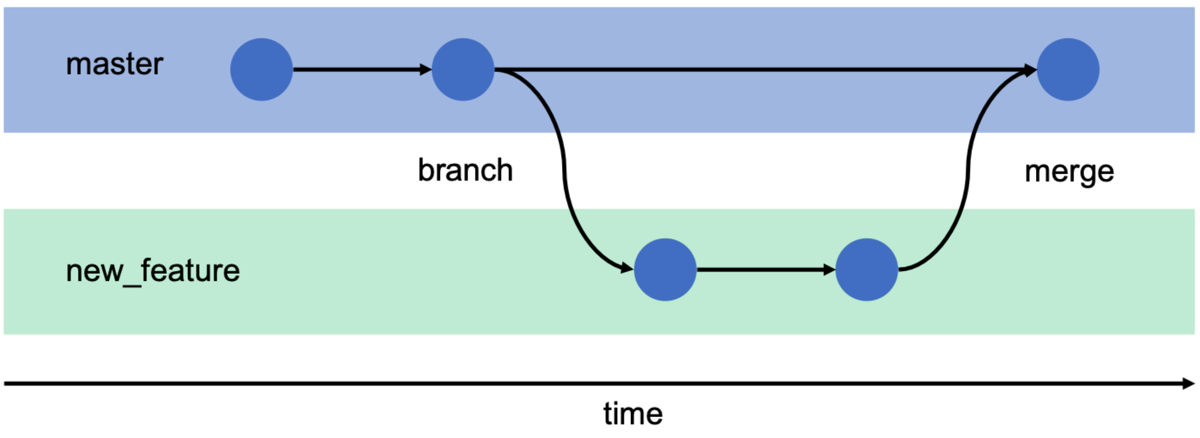
Branch Management

In Git, a branch is a new/separate version of the main repository.Branches allow you to work on different parts of a project without impacting the main branch.

When the work is complete, a branch can be merged with the main project.

You can even switch between branches and work on different projects without them interfering with each other.

Branching in Git is very lightweight and fast!



- Create Branch
  git branch <branchName>

- Get all branches
  git branch

- Switch branch
  git switch <branchName>

- Create & switch
  git switch -C <branchName>

- Rename branch
  git branch -m <currentBranchName> <newBranchName>

- Check log in all branches
  Shows the commits log

  - display each commit on a single line
    git log --oneline --all

  - display commit as graph
    git log --oneline --all --graph

- Delete branch
  - Only fully merged/empty branches are allowed to be deleted.
  - You need to switch to another branch before deleting.
    git branch -d <branchName>

- Force delete
  git branch -D <branchName>

Git stash

The Git stash command can be used to accomplish this if a developer is working on a project and wants to preserve the changes without committing them.
This will allow him to switch branches and work on other projects without affecting the existing modifications. You can roll back modifications whenever necessary, and it stores the current state and rolls back developers to a prior state.

- Push to Stash
  ```
  git stash push -m "<stash message>"
  ```

- View Stash files
  - View full list
    ```
    git stash list
    ```

  - Check with stash id
    ```
    git stash show <stash id>
    ```

  - Check with index no
    ```
    git stash show <index no>
    ```

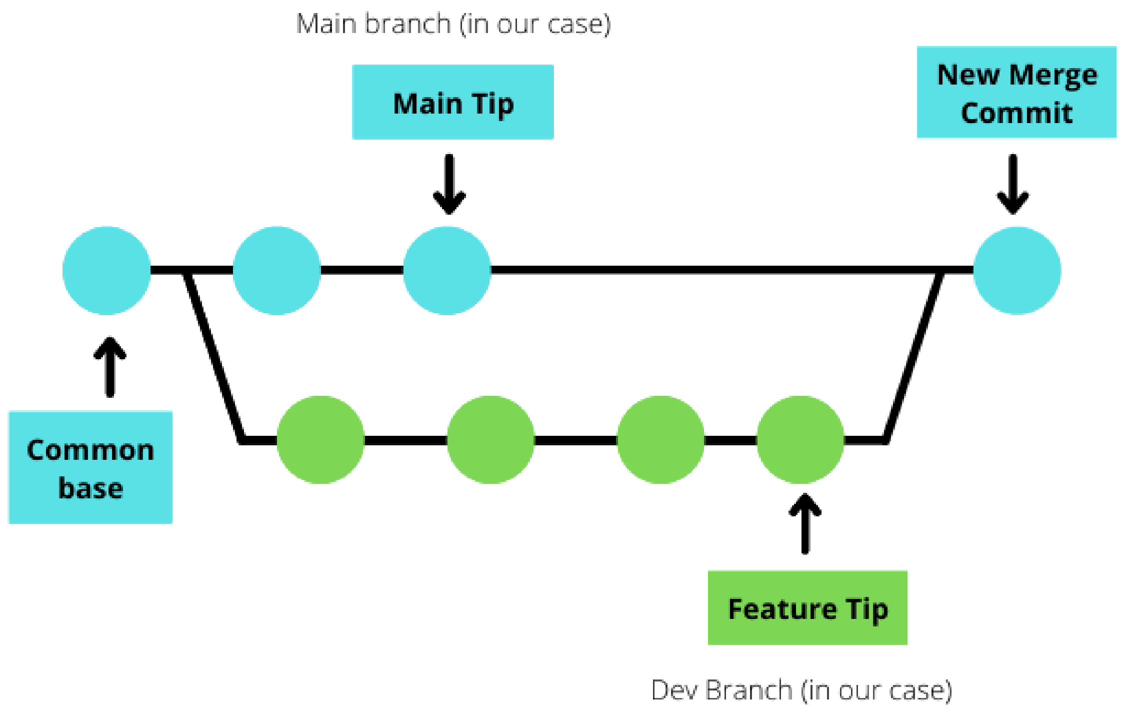
- Apply stash changes
  ```
  git stash apply
  ```

- Clear stash
  - Clear specific stash
    ```
    git stash drop <index no>
    ```

  - Clear all stash
    ```
    git stash clear
    ```

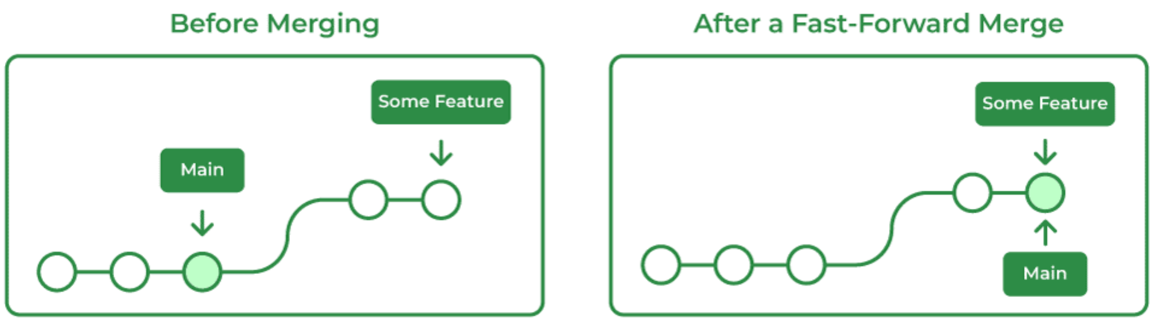
Merging

git merge is a command used to combine the changes from one or more branches into the current branch. It integrates the history of these branches, ensuring that all changes are included and conflicts are resolved.



01. Fast-forward merging.

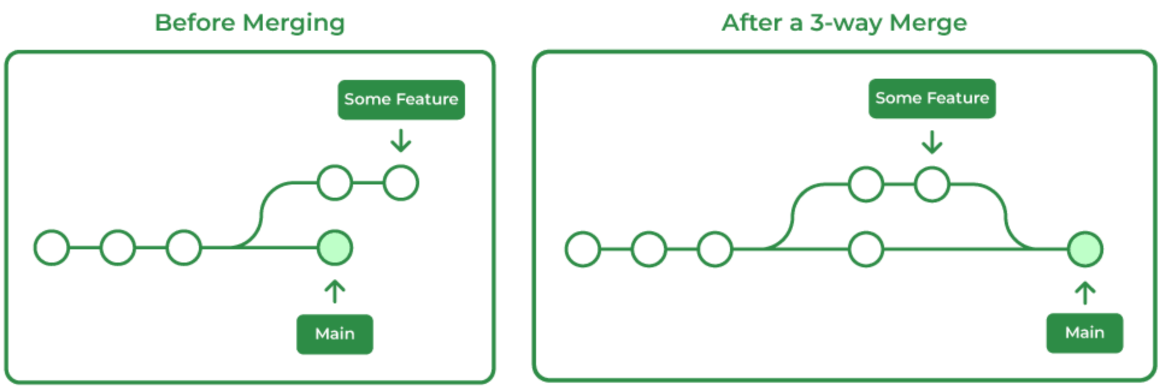
A fast forward in Git occurs during a merge operation when the base branch that's being merged into has no new commits since the feature branch (the branch being merged) was created or last updated. Instead of creating a new "merge commit," Git simply moves the pointer forward, hence the term "fast forward."



git merge <branchName>

02. Three-way merging.

A three-way merge in version control systems like Git combines changes from two branches by comparing them against their common ancestor, creating a new commit that integrates both histories.



```
git merge --no-ff <branchName>
```

Merge conflicts

A merge conflict happens when Git is unable to automatically reconcile differences in code between two commits. This typically occurs during a merge operation, where changes from different branches are combined.