Version Control systems

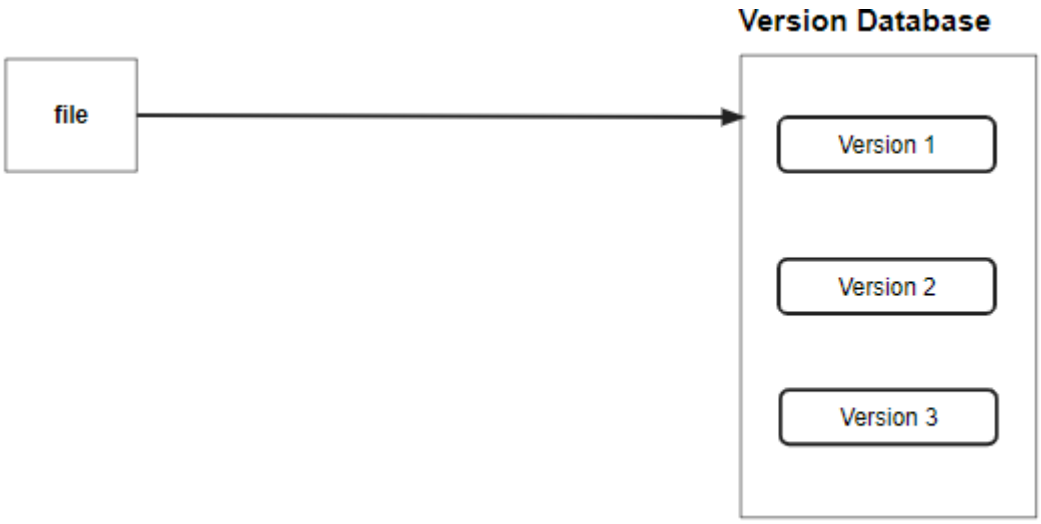
Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done to the code.

Types of Version Control Systems:

1. Local Version Control Systems
2. Centralized Version Control Systems
3. Distributed Version Control Systems

1. Local Version Control systems

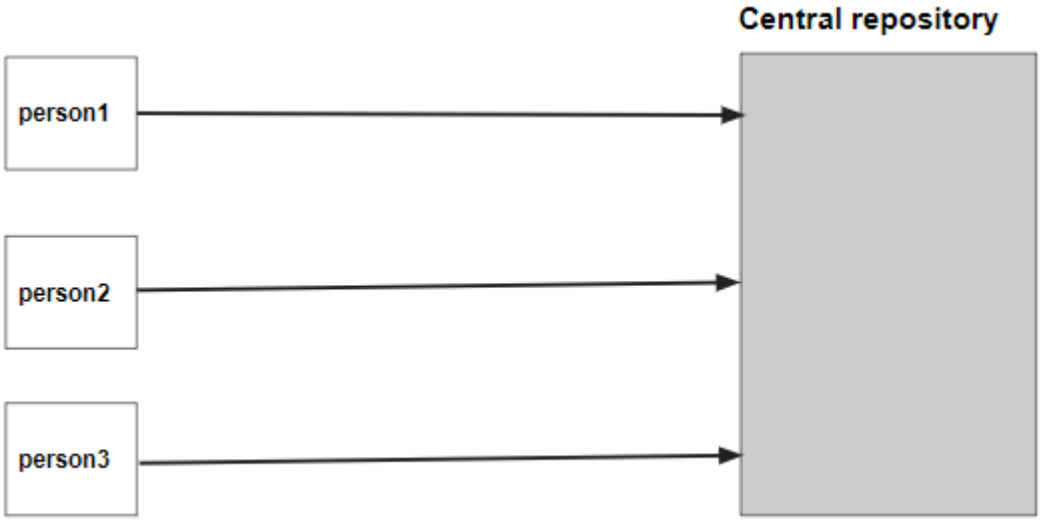
- A local version control system is a local database located on your local computer, in which every file change is stored as a patch.
- Every patch set contains only the changes made to the file since its last version



---

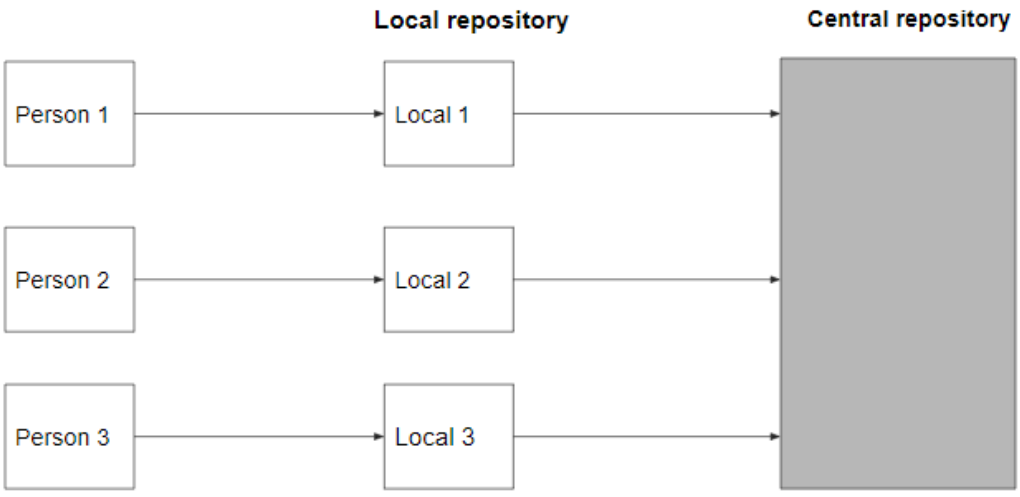
2. Centralized Version Control systems

- Centralized version control system has a single server that contains all the file versions.
- This enables multiple clients to simultaneously access files on the server, pull them to their local computer or push them onto the server from their local computer.
- This way, everyone usually knows what everyone else on the project is doing.
- Administrators have control over who can do what.
- This allows for easy collaboration with other developers or a team.



3. Distributed Version Control systems

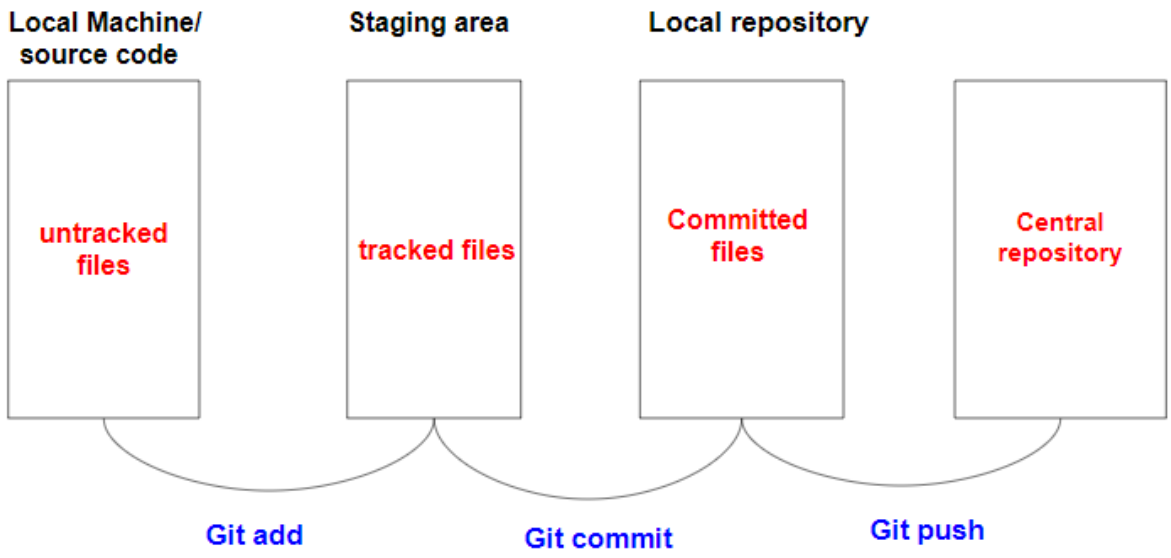
- Distributed version control systems contain multiple repositories.
- Each user has their own repository and working copy.
- Just committing your changes will not give others access to your changes.
- This is because commit will reflect those changes in your local repository and you need to push them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



GIT - Introduction

- Git is an open-source distributed version control system.
- It is designed to handle minor to major projects with high speed and efficiency.
- It is developed to coordinate the work among the developers.
- The version control allows us to track and work together with our team members at the same workspace.

Work flow of Git



Git commands

1. Git configuration

- **git config**: This command sets the author name and email address respectively to be used with your commits.
    - git config –global user.name “[name]”
    - git config –global user.email “[email address]”
- **git init:** This command is used to start a new repository.
    - Usage: git init [repository name]

2. Adding files to local repository

- **git add filename**: This command adds a file to the staging area.
- **git add *:** This command adds one or more to the staging area.
- **git commit -m "commit message"**: This command records or snapshots the file permanently in the version history.
- **git rm --cached filename**: This command deletes the file from your working directory and stages the deletion.
- **git rm --cached -r directory-name**: This command deletes the file from your working directory and stages the deletion.

3. Checking status

- **git status**: This command lists all the files that have to be committed.
- **git log**: This command is used to list the detailed version history for the current branch.
- **git log - -oneline**: This command is used to list the version history for the current branch in one line.

4. Pushing files to central repository

- **git remote add [variable name] [Remote Server Link]**: This command is used to connect your local repository to the remote server.
- **git push [variable name] master**: This command sends the committed changes of master branch to your remote repository.
- **git push [variable name] [branch]**: This command sends the branch commits to your remote repository.
- **git push --all [variable name]:** This command pushes all branches to your remote repository.

5. Git pull and clone

- **git pull [Repository Link]:** This command fetches and merges changes on the remote server to your working directory.
- **git clone URL:** This command is used to make a copy of a repository from an existing URL.
    - If you want a local copy of my repository from GitHub, this command allows creating a local copy of that repository on your local directory from the repository URL.

Additional commands: Branching in git

- **git branch**: This command lists all the local branches in the current repository.
- **git branch [branch name]**: This command creates a new branch.
- **git branch -d [branch name]:** This command deletes the feature branch.
- **git checkout [branch name]:** This command is used to switch from one branch to another.
- **git checkout -b [branch name]:** This command creates a new branch and also switches to it.
- **git merge BranchName:** This command is used to merge the specified branches history into the current branch.

Git revert v/s Git reset

- When you git revert a commit, only the changes associated with that commit are undone. Cumulative changes from subsequent commits aren't affected.

  Syntax: git revert commitID
- If you wish to undo every change since a given commit occurred, you'd want to issue a hard git reset, not revert.

  Syntax: git reset commitID

Git fork v/s Git clone

Fork	Clone
Forking is done on the GitHub Account	Cloning is done using Git
Forking a repository creates a copy of the original repository on our GitHub account	Cloning a repository creates a copy of the original repository on our local machine
Changes made to the forked repository can be merged with the original repository via a pull request	Changes made to the cloned repository cannot be merged with the original repository unless you are the collaborator or the owner of the repository
Forking is a concept	Cloning is a process
Forking is just containing a separate copy of the repository and there is no command involved	Cloning is done through the command '**git clone**' and it is a process of receiving all the code files to the local machine

Note: In git text editor window, you can press i to start entering text and save by pressing esc and :wq and enter, this will commit with the message you typed. In your current state, to just come out without committing, you can do :q