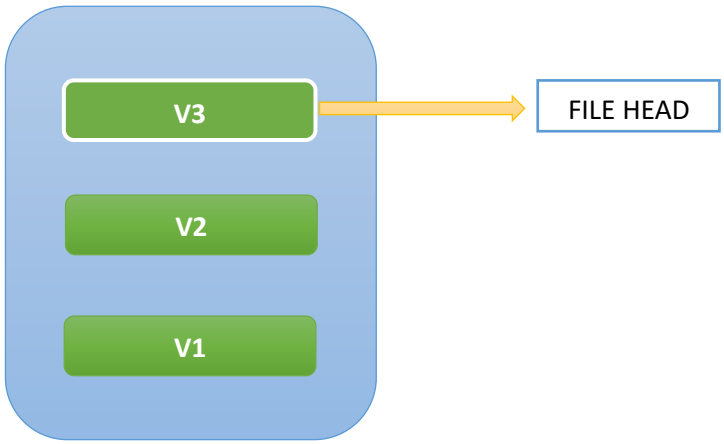
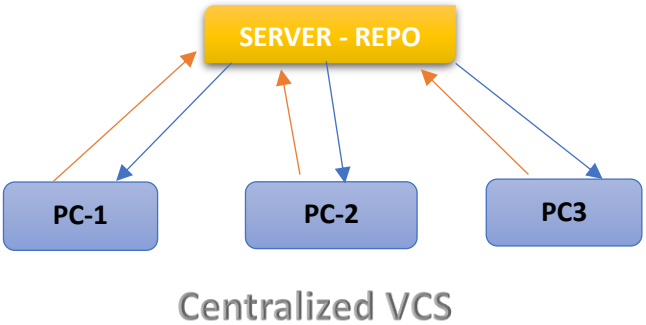
Git

Version Control System

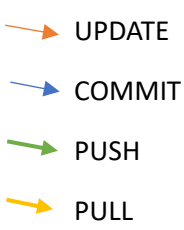
- It is a repository of files every changes made to the files are tracked with revisions, along with who made the change, why they made it and reference to problems fixed or enhancements introduced by the change.
- It is also know as revision control system [RCS] or Source code management [SCM]
- It help the team to ship the products faster improves visibility, traceability for every change ever made helps teams collaborate around the world.
- Types of version control systems
  - Local VCS
  - Centralized VCS
  - Distributed VCS
- Local VCS
  Only my Pc has a version database where I would make changes to a particular file

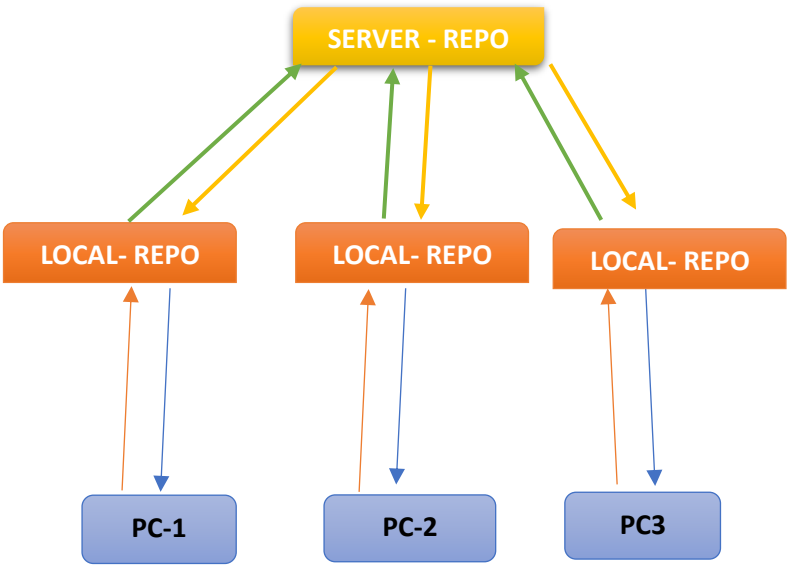


- Centralized VCS and Distributed VCS
  Over the network if I want to make any changes or update a file or collaborate with others the we would opt to centralized VCS or Distributed VCS



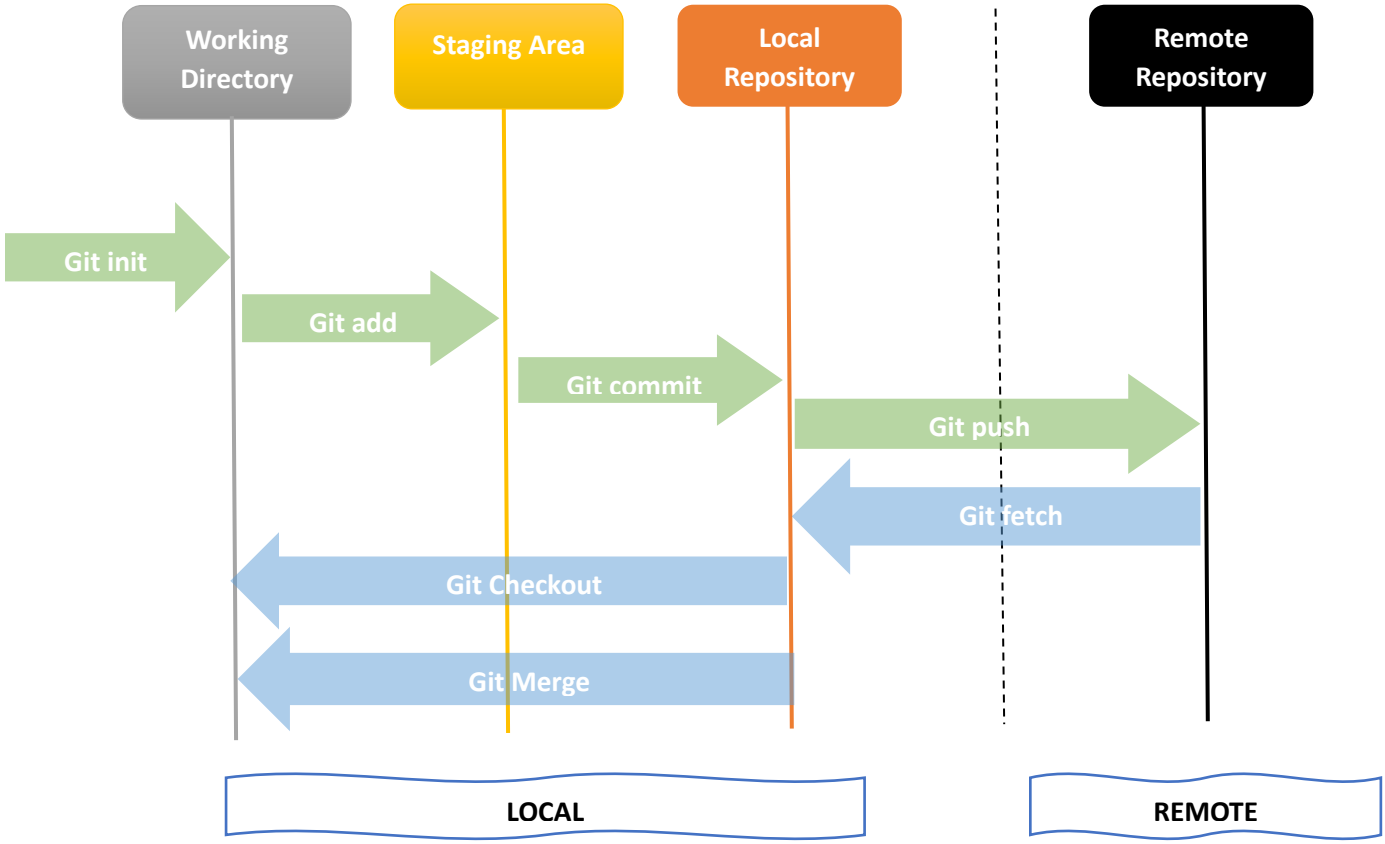
Centralized VCS





Distributed VCS

- List of version control system tools
  - There are several types of version control systems that teams use today, some are centralized some are distributed
  - There are few of the most popular types of VCS
    1. Git
    2. SVN
    3. Clear Case
    4. Mecurial
    5. TFS
    6. Helix core [Preforce]
- **GIT**
  - It is a open source tool and distributed version control system, infact git version is one of the most popular version control system
  - It is the most widely used modern version control system today
  - It is originally developed in 2005 by **Linus Torvalds,** the famous creator of **linux OS** kernel.
- **Git Features**
  - works on a distributed system
  - compatible with all OS
  - Allows for non-linear development
  - Branching
  - Fast as Flash
  - Reliable
- **GIT Workflow**



- Git branches

  branches are used to create independent lines of development. They allow you to work on different features, fixes, or experiments in isolation without affecting the main project
- Git stashing
  - In Git, **stashing** is a way to temporarily save your uncommitted changes (both tracked and untracked) without committing them, allowing you to switch to another branch or perform other tasks. Afterward, you can reapply these changes whenever you need them.
  - If you're in the middle of working on something and need to switch branches or pull changes from the remote, but your working directory has uncommitted changes, you can't switch without committing or discarding them. Stashing allows you to save those changes without needing to commit incomplete work.
  - **Save incomplete work**: Temporarily save your changes without needing to commit incomplete code.
  - **Clean working directory**: Stashing gives you the flexibility to switch branches without committing.
  - **Apply changes later**: You can always come back to your stashed work and apply it later.
- Git orphan branch
  - An **orphan branch** in Git is a branch that does not have any commit history from the parent branch it was created from. This means it starts with no commits and no reference to the commits of the branch you're on when you create it. This is useful when you want to create a completely new line of development without inheriting the history of the current branch.
  - **No common history**: It doesn't share any commits with the branch it was created from.
  - **Starts fresh**: The first commit on this branch will be the root commit for this branch, as if you're starting a new project.
  - **Useful for separate projects**: Often used to maintain documentation, GitHub Pages, or completely independent work in the same repository.
- Merge Conflicts
  - Sometimes Git cannot automatically merge changes, especially if the same part of the code has been modified in both branches. This results in a **merge conflict**. Git will pause the merge and ask you to resolve the conflicts manually.
- Git fetch

  In Git, the git fetch command is used to download commits, files, and references from a remote repository into your local repository, **without merging them** into your current working branch. It updates your local copy of the remote branches but doesn't modify your working directory or create any new commits on your local branches.
- Git pull

  In Git, **git pull** is a command that updates your local branch with changes from a remote repository. It's essentially a combination of two Git commands: **git fetch**: Downloads changes from the remote repository.
  git merge: Automatically merges those changes into your current branch.
- Git rebase
  - In Git, **git rebase** is a powerful command that allows you to move or replay commits from one branch onto another. It is used to create a cleaner, linear commit history by applying your changes on top of a different base (or branch) rather than merging them together with a merge commit.
  - **Cleaner History**: It eliminates unnecessary merge commits, making the commit history linear and easier to understand.
  - **Replaying Changes**: Rebasing moves or replays your local commits on top of another branch, integrating changes in a chronological order.
  - **Avoid Merge Commits**: It prevents creating merge commits, which can clutter the history when working with feature branches.
- Git ignore
  - In Git, **.gitignore** is a file used to specify which files or directories Git should **ignore** (i.e., not track or include in version control). This is helpful to prevent accidentally committing files that aren't necessary, such as temporary files, build artifacts, or sensitive information (e.g., API keys or environment files).
  - **Ignore unnecessary files**: Files that aren't part of the project's source code (e.g., log files, temporary files, binaries) should be excluded.

- **Prevent clutter**: Helps keep your repository clean by avoiding the inclusion of irrelevant files.
- **Avoid sensitive data**: Keeps sensitive files (like configuration files with passwords or tokens) out of version control.

- Git Commands
  - **touch** – to create a file
  - **mkdir** – to create a directory/ folder
  - **git init** – initialize git repository in current directory
  - **ll -lrt** – to list all the files in the current directory
  - **ll .git** - .git has all the files all those files are metadata and we are listing it using this command
  - **vim** – to open and enter the content/ insert or replace in that particular file [text editor]
  - **git add** – add the file from the working directory to the staging area, ' . ' represents all files
  - **git status** – to check the status of the file and suggest to move forward or backward in the staging area
  - **git rm –cached** <**file name**> - to unstage the file in the staging area
  - **git commit -m** '**filled added'** – adds the file in the local repository / saves the file with the commit message ' -m'
  - **git restore** <**file name**> - restores the modified file
  - **cat** <**file name**> - show the content in the file
  - **echo "This is a file content" > file.txt**- a text is stored in a file .txt and created
  - **git log** – helps you to track the author
  - **git log –oneline** – helps you to track the author in single line
  - **git reset <commit id>** - you can shift the head to the desired modification – 'default soft'
  - **git reset –hard <commit id>** - not only shifts the head to the desired modification but also deletes along with the data
  - **git reset --soft <commit id>** - it will not show you any changes and it will remove only the data in VCS but not in your directory
  - **git revert <commit id>** --no-commit – it will forcing remove without the editor and changes will not view in the log
  - **git stash list -** You can view a list of all stashed changes
  - **git stash apply** - To apply the most recent stash
  - **git stash apply stash@{2} -** To apply a specific stash
  - **git stash pop** - This applies the most recent stash and removes it from the stash list
  - **git stash drop stash@{0}** -To delete a specific stash from the list
  - **git stash -u** - By default, git stash only stashes changes in tracked files. If you want to stash untracked files too, use
  - **git stash clear** - To remove all stashes
  - **git checkout --orphan <branch_name>** - To create an orphan branch in Git, you use the --orphan option with the git switch or git checkout command.
  - **git rm -rf .** - **Remove all files from the working directory**: Since the orphan branch will bring all the files from the previous branch, but without the commit history, you can remove these files to start fresh:
  - **git merge <branch>** - merging two branches
  - **git merge –abort** – abort the merge
  - **git merge origin/main** - **Merging Fetched Changes**: Once you've fetched the changes, you can merge them manually
  - **git fetch** - Fetch Changes from the Remote Repository
  - **git fetch origin** - **Fetch from a Specific Remote**: If you want to fetch from a specific remote, you can specify the remote's name:
  - **git fetch origin main** - **Fetch Specific Branch**: You can fetch a specific branch or tag by specifying it:
  - **git log origin/main - View the Fetched Changes**: After fetching, you can inspect the updates by using
  - **git pull origin main** - To pull the changes from the main branch of the default remote (usually origin)

- **git pull --rebase origin main** - Instead of using the default merge strategy, you can use git pull --rebase to apply your changes **on top of** the fetched changes, avoiding a merge commit:
- **git rebase main** - Rebase the current branch onto main
- **git rebase –continue** - Continue after resolving conflicts
- **git rebase –abort -** Abort a rebase
- **git rebase -i HEAD~3 -** Interactive rebase for the last 3 commits
- **git clean -fdx** - The git clean -fdx command is a powerful Git tool used to remove untracked files and directories from your working directory. It essentially resets your working directory to a clean state by deleting any files that Git isn't tracking. Here's a breakdown of the command: