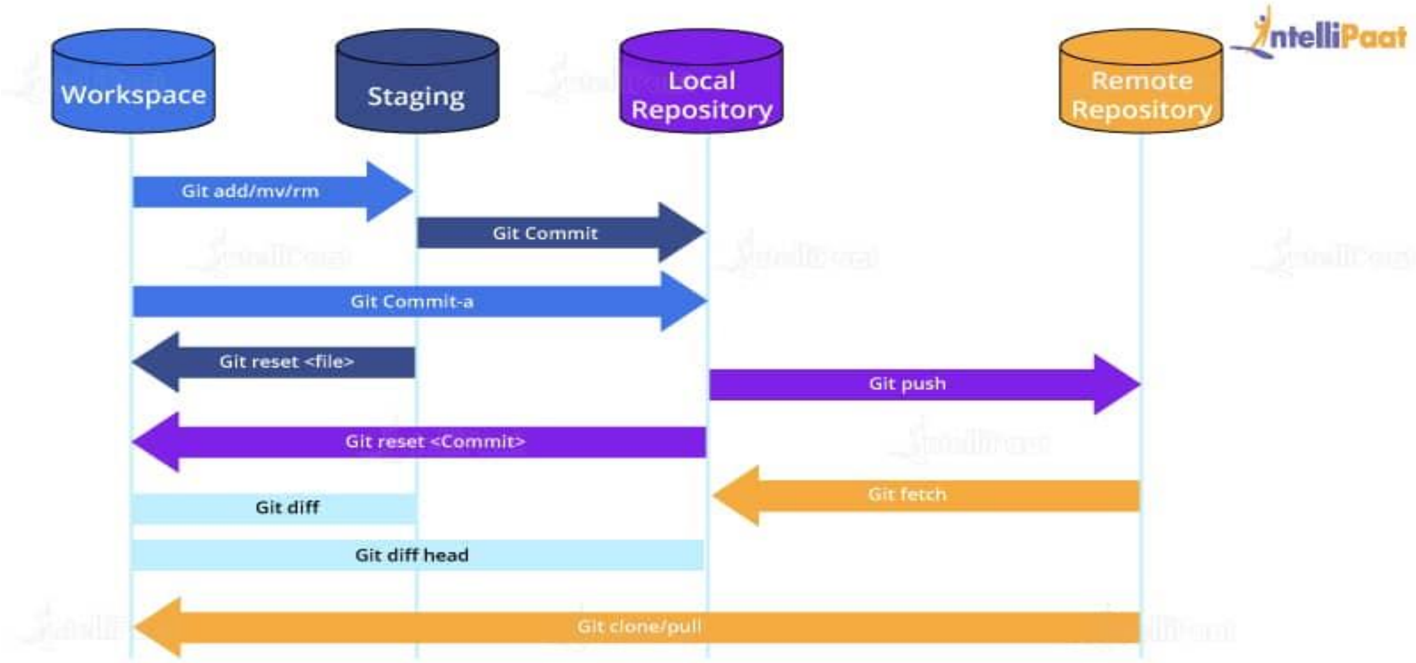
GIT

1. **Git Architecture**
2. **Source code.**
3. **Advantages**
4. **Available source code – Git, SVN, perforce, clear code**
5. **Git Advantages – Speed, Parallel branching, fully distributed**
6. **Stages – 4**
7. **Branching strategy**
8. **Type of repo in git**
    1) **Bare Repositories (Central Repo) -** Store and share only
    2) **Non-Bare Repositories (Local Repo) -** Where you can modify the files
9. **Elaborate commit**
10. **Snapshot**
    1) **Backup copy of each version**
    2) **Incremental backup**
    3) **Staging area**
11. **Github**
12. **Git merge**
13. **Git stash**
14. **Git reset**
    1) **Soft**
    2) **Mixed**
    3) **Hard**
15. **Git revert**
16. **Git pull**
17. **Git clone**
18. **Git fetch**
19. **Git merge**
20. **Git rebase**
21. **Bisect – Pick bad commit out of good commit**
22. **Squash**
23. **Git hook**
    1) **Pre commit**
    2) **Post commit**
24. **Chery-pick**
25. **Git/svn**
26. **Commit merge in git**
27. **Configuration management**
28. **Namespace**
29. **Trunk Based Development**
30. **Git Blame**

Git Architecture

The three layers are:

- **Working directory**: This is created when a Git project is initialized onto your local machine and allows you to edit the source code copied.
- **Staging area**: Post the edits, the code is staged in the staging area by applying the command, **git add**. This displays a preview for the next stage. In case further modifications are made in the working directory, the snapshots for these two layers will be different. However, these can be synced by using the same 'git add' command.
- **Local repository**: If no further edits are required to be done, then you can go ahead and apply the **git commit** command. This replicates the latest snapshots in all three stages, making them in sync with each other.



Version Control System/ Software Configuration Management

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

History of GIT

- Introduced by Linus Torvalds 2005.
- Designed for Linux.
- Founder of Linux **"LINUS TORVALDS"**
- Linux me 2005 se pahle versioning ke lie BITKEEPER s/w use kiya jata tha jo third party s/w tha. Then baad me ye s/w paid ho gya to LINUS TORVALDS ne khud ka Versioning Control System banaya, that is GIT.

Types of Version Control Syste

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

Drawback of Centralized Version Control System

- Not locally available, meaning you always need to be connected to a network to perform any action.
- Since everything is centralized, if central server gets failed, you will lose entire data.
- CVCS Tool name – SVN

CVCS	DVCS
- In CVCS a client need to get local copy of source code from server, do the changes and commit those changes to the server.	- In DVCS each client can have a local branch as well and have a complete history on it.
- There are no local repo.	- There are local repo.
- CVCS is slower as every command need to communicate with server.	- DVCS is faster as mostly user deals with local copy without hitting server everytime.
- Always require internet connectivity.	- Developers can work with a local repository without an internet connection.
- If CVCS server is down developers can't work.	- If DVCS server is down developer and work using their local copies.

WHAT IS GIT

- Git is a tool/Software used for source code management.
- GIT is a free and opensource distributed version control system.
- Used to handle small to very large project efficiently.
- Git is used to tracking changes in the source code.
- Enable multiple developers to work together.
- Git like tools AWS CodeCommit, Mercurial, Helix Core etc.

Advantages of GIT

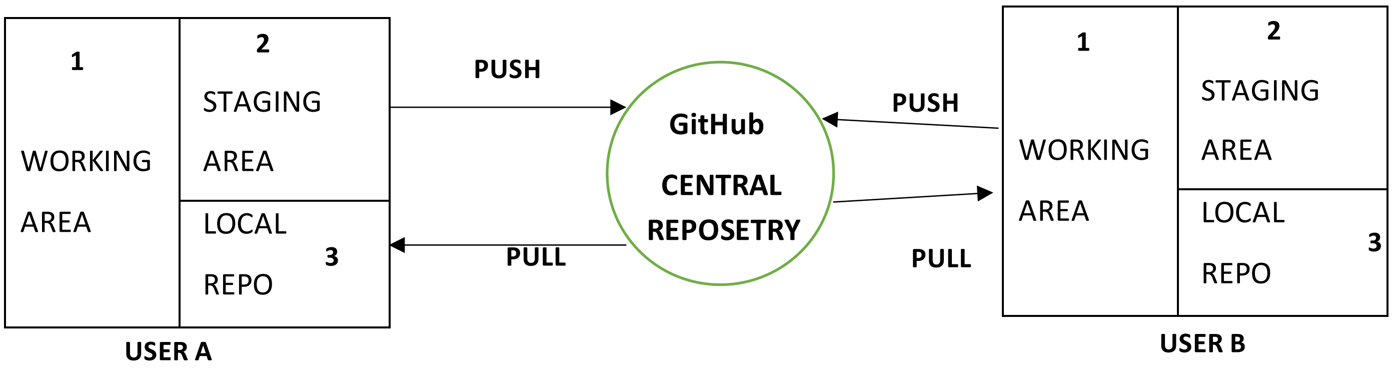
- Free and open source.
- Fast and small – as most of the operations are performed locally, therefore it is fast.
- Security – git uses a common cryptographic hash function called secure hash function (SHA-1) to name and identify objects within its database.
- No need of powerful H/W.
- Easier branching – if we create a new branch, it will copy all the codes to the new branch.

GIT BENEFITS

- Work on snapshots.
- Every operation is local.
- Git has integrity
- Git generally only adds data.

SOURCE CODE MANAGEMENT

1. CVCS = CENTRALIZED VERSION CONTROL SYSTEM (OLDER METHOD)
2. DVCS = DISTRIBUTED VERSION CONTROL SYSTEM (CURRENTLY USING EX. GIT)



IQ – DIFFERENCE B/W CENTRALIZED VERSION CONTROL SYSTEM AND DISTRIBUTED VERSION CONTROL SYSTEM?

ANS - The main difference between centralized and distributed version control is that, in centralized version control, the versions are saved in the remote repository, while in distributed version control, versions can be saved in the remote repository as well as in local repositories of the local.

4 STAGES OF GIT

1. **MODIFIED** (If file was changed since it was checked out but has not been staged, it is **modified)**
2. **STAGED** (If it has been modified and was added to the staging area, it is **staged)**
3. **COMMITED** (If a particular version of a file is in the Git directory, it's considered **committed)**
4. **Central Repository**

Repository

- Repo. is a place where you have all your codes or kind of folder on server.
- It is a kind of folder related to one product.
- Changes are personal to that particular repository.

Types of Repositories

Bare Repositories (Central Repo)

- Store and share only.
- All central repositories are bare repo.

Non-Bare Repositories (Local Repo)

- Where you can modify the files.
- All local repositories are non-bare repositories.

Server

It stores all repos.

It contains metadata also.

Working Directory

- Where you see files and do modification.
- At a time you can work on particular branch

Commit

- Store changes in repo. you will get one commit-ID/ Version-ID/ Version.
- It is 40 alpha-numeric charactors.
- It uses SHA-1 Checksum concept.
- Even if you change one dot, commit-ID will be change.
- It actually helps you to track the changes.
- Commit is also named as SHA-1 hash.

Tags

- Tag operation allows giving meaningful names to a specific version in the repository.
- git log --oneline ---> to see commit id's af all commits
- git tag -a <tag-name> -m <message> <commit-ID> --->to add tag to a commit.
- git tag ---> To see the list of tags
- git show <tagname> ---> To see particular commit content by using tag.
- git tag -d <tagname> ---> to delete a tag
- Tags assign a meaningful name with a specific version in the repository once a tag is created for a particular save, even if you create a new commit, it will not be updated.

Snapshots

- Represents some data of particular time. (Jese photo bhi 1 particular time ki hoti h na ki pure din ki)
- It is always incremental i.e., it stores the changes (appends data) only not entire copy.

Push

Push operations copies changes from a local repository server to a remote or central repo this is used to store the charges permanently into the git repository.

Pull

Pull operation copies the changes from a remote repo. to a local machine the pull operation is used for synchronization between two repo.

Branch

- Product is same so one repo. but different task.
- Each task has one separate branch.
- Finally merges (code) all branches.
- Useful when you want to work parallelly.
- Can create one branch on the basis of another branch.
- Changes are personal to that particular branch.
- Default branch is Master.
- File created in workspace will be visible in any of the branch workspace until yo u commit once you commit, then that file belongs to that particular branch.

GIT INSTALL

Yum install git -y

Git –version

mkdir /mywork

cd /mywork
git init (to initialize git workspace)

vim deepak

git status (to check code status)

git add deepak (to add file in staging are)

git commit -m “my first commit” deepak (m= message, msg aap apne hisab se kuch bhi de skte ho)

NOTE : Commit ID is a 40 Alphanumeric Characters

git log (to view all commits)

git log –decorate (makes git log display all of the references (e.g., branches, tags, etc) that point to each commit.)

git log –oneline (to display the output as one commit per line**)**

git log –oneline --decorate

GIT Config. For particular work space (Ye configuration dusri directory ke lie kaam nhi krega)

Git config user.name “Deepak Sharma”

Git config user.email deepakkrsharma29@grras.com

GIT Config. For Global Setting (Ye configuration globally kaam kregi)

Git config --global user.name “Deepak Sharma”

Git config --global user.email deepakkrsharma29@grras.com

GIT Config. For system Setting

Git config --system user.name "Deepak Sharma"

Git config --system user.email deepakkrsharma29@grras.com

Working on GITHUB

Github is a central repo.

Similar s/w like GitHub are GitLab.

Git remote add origin LINK (LINK = Link of github repository, ogigin name ke variable me hum link ki value store krvare h taki bar-bar link past nhi krna pde)

GIT Clone

git clone <url of github repo> (is command se GitHub (central repo.) ki all files local machine ke workspace me clone/copy ho jati h)

GIT Push

git push origin master (origin = aap origin ki jagha kuch bhi name de skte ho , master = master branch)

GIT Pull

git pull origin (github ka data local machine pe lane ke lie)

NOTE:- in this command it will ask username and password then you can give username and password (but nowdays it don't accept passwords that's why you have to generate token) otherwise you can go to GitHub website and ->profile->setting->developer setting->personal access tokens->generate new token And then generate token by selecting some checkboxes according your requirement.

Then copy that token number and past that token when it ask for password after giving Username.

BRANCHES

Branches **allow you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository**. You always create a branch from an existing branch. Typically, you might create a new branch from the default branch of your repository.

File created in workspace will be visible in any of the branch workspace until you commit. Once you commit then that files belongs to that particular branch.

NOTE:- Branches me 1 Master branch hoti h baki Feature branch hoti h.

BRANCH COMMANDS

git branch (to list all branch)

git branch kgf2 (to create a new feature branch)

git branch -D kgf2 (to delete a branch)

git checkout BRANCH_NAME (to switch one branch to another)

MERGE

git merge kgf2 (feature branch ka data master branch me add krne ke lie kiya jata h)

NOTE:- merge hamesa master branch se kiya jata h. Feature branch ka data master branch me add krne ke lie merge use kiya jata h.

You can’t merge branches of two different reposetories.

Git Commands to Resolve Conflicts

1. git log --merge

The git log --merge command helps to produce the list of commits that are causing the conflict

2. git diff

The git diff command helps to identify the differences between the states repositories or files

3. git checkout

The git checkout command is used to undo the changes made to the file, or for changing branches

4. git reset --mixed

The git reset --mixed command is used to undo changes to the working directory and staging area

5. git merge --abort

The git merge --abort command helps in exiting the merge process and returning back to the state before the merging began

6. git reset

The git reset command is used at the time of merge conflict to reset the conflicted files to their original state

MERGE CONFLICT

A merge conflict is **an event that takes place when Git is unable to automatically resolve differences in code between two commits**. Git can merge the changes automatically only if the commits are on different lines or branches.

When some file having different content in different branches, if you do merge, conflict occurs (Resolve conflict then add and commit)

Conflict occurs when you merge branches.

If you face **MERGE CONFLICT** then use this tool

- git mergetool

- git commit -m “final commit”

REBASE

git rebase master (feature branch me master branch ka data update krne ke lie kiya jata h)

NOTE:- rebase hamesa feature branch se kiya jata h. Agar master branch ne feature branch banane ke bad master branch me kuch update kya h to use feature branch me update krne ke liye feature branch ko rebase karna hoga.

IQ – DIFFERENCE BETWEEN MERGE AND REBASE?

ANS –

MARGE = merge hamesa master branch se kiya jata h. Feature branch ka data master branch me add krne ke lie merge use kiya jata h.

REBASE = rebase hamesa feature branch se kiya jata h. Agar master branch ne feature branch banane ke bad master branch me kuch update kya h to use feature branch me update krne ke liye feature branch ko rebase karna hoga.

PASSWORD CREDENTIAL SAVE PERMANENTALY

Agar aap bhi chahte h ki aapke system pe bhi bar-2 credentials na puche to aap credentials apne system pe store krva skte h is command ke through. Is command se kya hoga ki jab bhi aap is command ko run krane ke bad username or password denge to use system save kr lega fr dobara nhi puchega.

```
git config--global credential.helper store
```

file location cd /root/.git-credentials

if you delete this file then system will ask you username and password again.

GIT HOOKS

Git hooks are **scripts that run automatically every time a particular event occurs in a Git repository**. They let you customize Git's internal behaviour and trigger customizable actions at key points in the development life cycle.

By default system me kuch hooks hote h

Hooks location cd .git/hooks

(You can get hooks code from google.)

Here we will configure a **<u>Post Commit</u>** Hook

POST COMMIT: - The post-commit hook is called immediately after the commit-msg hook. It can’t change the outcome of the git commit operation, so it’s used primarily for notification purposes.

- Search on google “git hook for post commit”
- https://www.atlassian.com/git/tutorials/git-hooks
- vim .git/hooks/post-commit (is file ki command abhi nhi chlegi kyonki file ko abhi execute ki permission nhi h)

  ```
  #!/bin/bash
  git push origin master
  ```
- chmod a+x post-commit (is command se execute ki permission mil jayegi)
- ab hum master pe kuch bhi commit krenge to automatically github pe push ho jayegi.

<u>RESET / ROLE BACK</u>

- Git reset is a powerful command that is used to undo local changes to the state of a git repo.
- Reset current HEAD to the specified state.

To reset staging area

- git reset <filename>
- git reset .

GIT RESET 3 METHODS (by default reset method is MIXED)

1. SOFT
2. MIXED
3. HARD

1. SOFT (soft reset me file commit area(locak repo.) se staging area me aa jati h jise dobara commit kiya ja skta h)

- git reset --soft HEAD~
- git log --oneline --decorate

2. MIXED (mixed reset me file commit area or staging area dono jagha se working area me aa jati h jise dobara add or commit kiya ja skta h)

- git reset --mixed HEAD ~
- git log--oneline –decorate

3. HARD (hard reset me file sabhi jagha se delete ho jati h like working area, staging area or commit area(local repo.)

- git reset --hard HEAD ~
- git log --oneline --decorate

Reset staging area se specific commits ko delete kiya jata h.

REVERT

- The git revert command is essentially a reverse git cherry-pick. It creates a new commit that applies the exact opposite of the change introduced in the commit you're targeting, essentially undoing or reverting it.
- Revert some existing commits.
- The revert command helps you undo an existing commit.
- It does not delete any data in this process instead rather git creates a new commit with the included files reverted to their previous state so, your version control history moves forward while the state of your file moves backword.
- Revert command se automatically 1 new commit ho jata h with new commit ID.
- **Reset -> before commit**
- **Revert -> after commit**
- git revert <commit-id> ---> you can type commit as **"Please Ignore previous Commit"**
- git reflog ---> will show us our current commit history
- Revert se local repo ki specific commit delete ki jati h.

How to Remove Untracked files

Untracked files are **files that have been created within your repo's working directory but have not yet been added to the repository's tracking index using the git add command**.

- git clean -n ---> ask before delete
- git clean -f ---> forcefully

SQUASH

To "squash" in Git means **to combine multiple commits into one**. You can do this at any point in time (by using Git's "Interactive Rebase" feature), though it is most often done when merging branches. Please note that there is no such thing as a stand-alone git squash command.

You can merge several commits into a single commit.

Git squash krne se feature branch ke jitne bhi commit h vo master branch me show nhi honge jab aap "git status" command chalaoge tab

- git merge –squash FEATURE_BRANCH_NAME
- git merge – squash deepak

STASH (छिपाना)

git stash **temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on**.

- git stash (instead of commit command)
- git stash list (to show all stashed data)

- git show stash (to show what work you have done)
- git stash apply (to remove stash from file)

IQ – Difference between clone and pull?

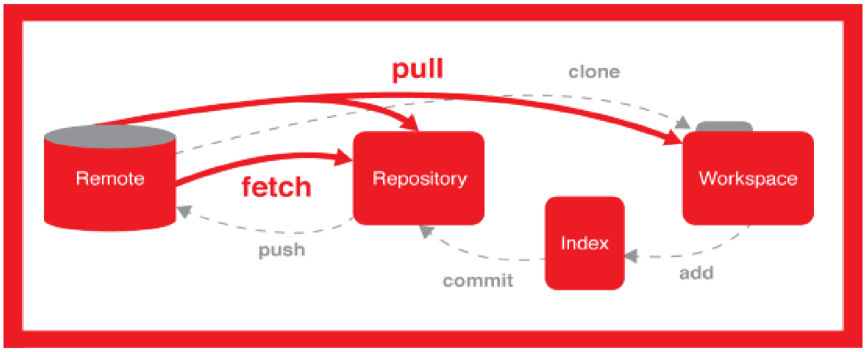
A -- Clone-: It will create exactly duplicate copy of your remote repository project in your local machine.

Pull-: Suppose two or more than two people are sharing the same repository. (Suppose another person name is Syam) (A Repository is a place where your project exist in Github) So if Syam does some changes in the same project in his local and pushes it to the remote repository So whatever the changes Syam did those changes will not reflect in your local. So to reflect those new changes in your local you have to use git pull. Overall we use git pull to update the project.

So basically we use git clone only once whereas we use git pull many times.

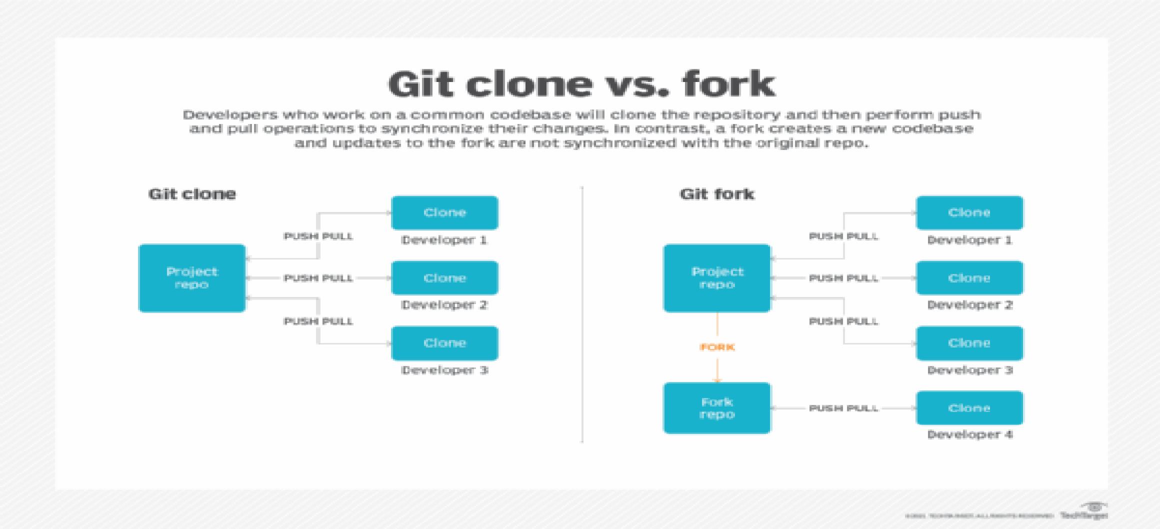
IQ – Difference b/w fetch and pull?

A -- git fetch is the command that tells your local git to retrieve the latest meta-data info from the original (yet doesn't do any file transferring. It's more like just checking to see if there are any changes available). git pull on the other hand does that AND brings (copy) those changes from the remote repository.



Fork

A developer who wants to set up a new, separate, and isolated project that is based on a publicly accessible Git repo should perform a fork.



Git clone vs. fork

Developers who work on a common codebase will clone the repository and then perform push and pull operations to synchronize their changes. In contrast, a fork creates a new codebase and updates to the fork are not synchronized with the original repo.

IQ – difference b/w reset and revert?

A -- **Reset -> before commit** (reset private file pe kiya jata h)

Revert -> after commit (Revert Public file pe kiya jata h)

Git Bisect

- **Pick bad commit out of good commit.**
- The git bisect tool is an incredibly helpful debugging tool used to find which specific commit was the first one to introduce a bug or problem by doing an automatic binary search.
- Use binary search to find the commit that introduced a bug.
- The git bisect command is **used to discover the commit that has introduced a bug in the code**. It helps track down the commit where the code works and the commit where it does not, hence, tracking down the commit that introduced the bug into the code.
- Find a bad commit in your app

How to use Git Bisect

- git bisect start
- git bisect bad
- git bisect good <commit-ID>
- git bisect reset reset ---> to finish bisect

How to view the changes

- git show <commit-ID>

How to revert a bad commit

- git revert <commit-ID>

Git Blame

The git blame command annotates the lines of any file with which commit was the last one to introduce a change to each line of the file and what person authored that commit. This is helpful in order to find the person to ask for more information about a specific section of your code.

Cherry-Pick (Synonyms = Choose, Elect, Handpick)

- Cherry-pick is used **to select the best or most desirable commit.**
- The git cherry-pick command is used to take the change introduced in a single Git commit and try to re-introduce it as a new commit on the branch you're currently on. This can be useful to only take one or two commits from a branch individually rather than merging in the branch which takes all the changes.
- Apply the changes introduced by some existing commits.

Namespace

Git supports dividing the refs of a single repository into multiple namespaces, each of which has its own branches, tags, and HEAD.

Trunk Based Development

Trunk Based Development is a version control management practice. Where developers create more feature branches and develop on these small branches and once it's ready they merge into trunk or the master, this is called as trunk-based development.

Benefits of Trunk-Based Development

Reduced Complexity

The main benefit of trunk-based development is the reduced complexity of merging different branches into one. This approach aims to avoid merge hell, a situation when different pieces need to be combined for the first time which leads to unexpected bugs, integration issues and blocks the team from deploying.

Increasing Speed of Delivery

Implementing and using trunk-based development in the long term could increase team discipline and a feeling of teamwork by establishing clear processes and giving more opportunities for collaboration.

Shorting the Feedback Loop

Building in short feedback cycles could also help verify initial design assumptions. Because the trunk should be always stable, the code is potentially releasable to customers for getting early feedback.

Automation

Automating the building, testing and deployment software are key enablers of trunk-based development. It allows for making quick iterations while reducing the chance of breaking the main branch.

Feature Flags

Using feature flags you can decouple deploying the new code to the production environment from releasing the feature to the users.