GIT

Git is Global Information Tracker and also called VCS and SCM

Version control system (VSC) helps to manage the multiple versions of single file and It allows users to revert to previous versions, compare changes and collaborate with others.

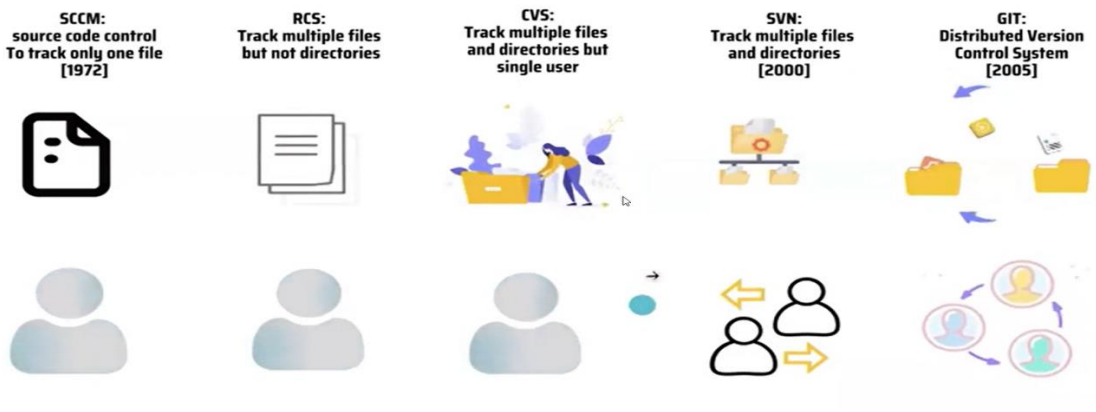
i.e., Version1 <<<->>> Version2 <<<->>> Version3

Source code management (SCM) helps to managing and tracking changes to source code files within a project. Every app has a background code i.e., Source code

- Global Tracking : Git tracks changes in multiple files across projects.
- Version Maintenance : It keeps different versions of the same file.
- Platform Flexibility : Git works on any Platform.
- Free and Open-Source : It's free to use and open for anyone to modify.
- Efficient Handling : Git handles large projects well.
- Time-Saving : Developers can fetch updates and create pull requests seamlessly.
- Modern Version Control : Git is a 3rd Generation Version Control System (VCS).

HISTORY OF VCS/GIT

Linus Torvalds, the creator of Linux, developed Git in 2005 to manage Linux kernel Development.



1. SCCM (Source Code Control Management) [1972]
   - It tracks only single file and single user.
2. RCS (Revision Control System)
   - It tracks multiple files but not directories and Tracks changes by single user
3. CVS (Concurrent Version System)
   - It tracks multiple files and directories but it tracks changes by single user
4. SVN (Sub-Version) [2000]
   - It tracks multiple files and directories as well as multi user.
5. GIT (Global Information Tracker) [2005]
   - It tracks multiple files and directories as well as multi user.
   - Because of Distributed Version Control System, we use git.

DISTRIBUTED VERSION CONTROL SYSTEM [DVCS]

- DVCS is a type of version control system, helps multiple users collaborate on the same code by allowing them to make changes independently and then merge those changes into a particular repository. (or)
- GIT tracks the changes you made (add, delete, create), So you have a record of what has been done and you can revert specific versions.
- It makes collaboration easy, allowing changes by multiple people to all merged into one source [one repo]
- Advantages Of DVCS
    - Backup copies in every team member's local machine.
    - Fast merging and flexible branching.
    - Rapid feedback and fewer merge conflicts.
    - Flexible to work offline (doesn't require internet connection, except push & pull)

WHY GIT

Let's take a scenario,

- If the client asks you to develop application.
- You can develop according to client requirements and you released version-1 of the application.
- One year later, the client comes to you and asks to change the options in the application.
- But unexpectedly the app got failed, In this case you can rollback to specific previous version.
- In This case, by using git we can comes back to previous version.

Git Stages

Git Stages are 3 types:

1.Working Directory:

- This is where your project files are stored.
- Git doesn't track these files by default (Untrack files).
- To start tracking changes, you need to add files to Git using $ git add <fileName>
- Once added, files move to the staging area.

2.Staging Area:

- Staging area is like a rough draft space.
- In this stage, Every file and folder being tracked by Git. (Tracking files).

- You can untack the tracking file by using command $ git rm --cached <filename>
- Here, You can prepare changes for next version of your project.
- Whenever we change data or version, need to commit the changes by using command $ git commit -m "message" <filename/*/.>
- Once we commit, the files will direct go to repository.

3.Repository:

- Repository in Git is considered as your project folder (Commit files)
- Repository has all project related data.
- It consists of the collection of the files and also history of changes made to those files.
- Types Of Repositories:

1.Local Repo

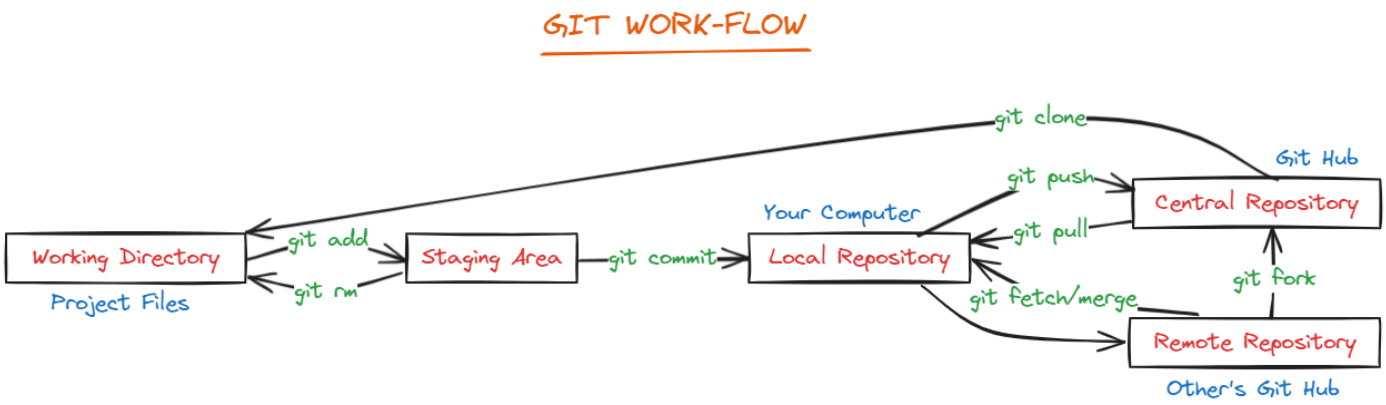
- The Local Repo is your computer.
- Everything stored in .git directory, It contains all the commits. (so don't delete).
- In local repo, Mainly we can see here all the commits or checkpoints.

2.Remote Repo

- This stored on another remote computers (Others GitHub repo)

3.Central Repo

- This will be present in our GITHUB
- GitHub is web-based platform used to store the source code.



GIT ALTERNATIVES

- Git-Lab
- Sub-Version (SVN)
- Bit-Bucket
- Perforce (P4)
- Stash
- Helix
- Fossil
- Bazaar

MANDATORY STEPS TO INSTALL GIT

1. Update Packages List:
   - To get latest software versions $ sudo yum update -y
2. Install Git:
   - Use package manager to install Git $ sudo yum install git -y
   - To check Git version $ git --version
3. Initialise Git Repository:
   - Initialise an empty Git Repository using $ git init .
   - Empty .git folder created inside root. i.e., root/.git
   - To get list of files, including hidden files using $ ls -al

Note: Without .git folder impossible track files. So, we need to create .git folder.

GIT COMMANDS

touch filename	To create a file.
git add filename ▪ git add * ▪ git add .	To track a single file. ▪ To track all files or multiple files. ▪ To track all files including hidden files.
git status	To check the file tracking or not.
git commit -m "commit message" <fname/*/.> ▪ m - message	To commit the file with message – saved
git rm --cached filename	To untrack (That means staging area to working directory)
git log	To check commit history → Tap 'q' for quit
git log -2	To see top 2 commit → Tap 'q' for quit
git log --oneline	To see only commit ID's and message
git show commit_Id filename git show commit_id --name-only	To see details of commit filename & details.
git log --follow --all filename	To see all commits of a single file
git revert <commit Id>	To delete particular commit id
git reset <commit id>	To get recent deleted commit id details

Git Log

Used to show all the commit history Details

$ git log

GIT-CONFIG

Git Config is used to Configure your Email and Name in Git CLI with Commands

git config --global user.name "naveen_silver"	To config user name/author
git config --global user.email "naveensilver136@gmail.com"	To config user email ID

Note: Now give the git log command to see changes, it won't work because after config we haven't done anything.

Now create a file and commit that file. Enter $ git log you will see changes as you configure.

- You can change Name and Emails configurations by following steps:

Go to .git folder > vim config > change username and email or replace →: %s/silver/naveen

GIT – IGNORE

When you have multiple files to track at a time but you don't want to track some specific files then we use a file called .gitignore

- Create some txt, pdf and jpg files
- vi .gitignore
    - *.txt → here we are ignoring .txt files

Ex : In maven project 'target' folder will be available which is not required to commit to central repo. This we can configure in .gitignore file.

GIT-DELETE and GIT-RESTORE

Tracking and Untracking files, we can delete by using rm -rf command. But by using git restore, we can get back those tracked deleted files.

- git restore <filename/*>

But you want to delete the tracked files permanently. First delete and commit those files

- rm -rf <filename/*>
- git commit -m "delete all the files" <filename/*>

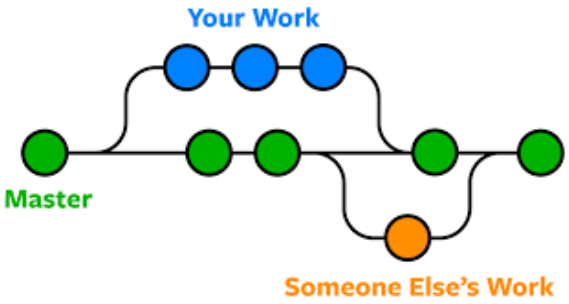
GIT – BRANCH

Git Branch is like separate path of work in your project. It lets you try out new ideas, fix issues, or test changes without affecting the main code.

- The default branch in Git is 'Master' branch.
- We use branch strategy mainly for two purposes:

1. Collaboration: Imagine several people working on the same project. Each person can take their own branch from Main(Master) branch to work their part separately. When they finish, their work can be merged back into 'Main' Branch. This keeps everything organised and prevent conflicts.

2. Versioning: You can also use branches to keep different versions of your project. For Ex, if you're making changes to your app, you can create a branch for each version like version-1, version-2. This allows you to experiment with new features or fixes without affecting the main version until you're ready to merge your changes in. So, We can maintain the multiple versions of the application.



- Git branch out from master branch.
- Easily work with other developers.
- Lot of flexibility in your workflow

Real Time Scenario,

Branches are used to maintain multiple code bases for the parallel work in the project.

One project will have only one GitHub repository which is created by Git Admin.

In one repo, we can create multiple branches like Main, Develop, QA, Release and Research..

- Development team will integrate the code in 'develop' Branch.
- Bug-fix team will integrate the code in 'QA' Branch.
- R&D team will integrate the code in 'Research' Branch.
- Production Deployment will happen from 'Release' Branch.

Using branches team members can do the work parallelly.

BRANCH Commands

git branch	To get list of all branches / default branch default branch : master
git branch branch-name	To create a branch Ex: git branch devops
git branch -m <old-b.name> <new-b.name>	To Rename the branch name Ex: git branch -m master silver
git checkout <shifting branch-name>	To shift one branch to another branch Ex: git checkout devops {master - devops}
git checkout -b <new branch-name>	To create a branch as well as shift to new branch Ex: git checkout -b aws
git branch -d/D <deleting branch-name> D- delete forcefully, branch has not been pushed/merged yet. d-branch has already pushed/merged with remote branch.	To delete a branch Ex: git branch -d devops [when you're deleting a branch, you shouldn't present in deleting branch]
git clone -b <branchName> <Repo_URL>	To get particular branch from Git-Hub
git merge <Source_branch-name>	To merge one branch files to another branch Ex: git merge aws {master ← aws }
git cherry-pick <paste commit_Id>	To merge a particular branch file Ex: first copy the particular merging file commit_ID{copy aws branch file commit id and shift to master branch}

Q) How To Merge the changes from One branch to Another Branch?

Branch Merging:

The process of merging changes from one branch to another branch is called Branch merging.

We will use pull request for Branch merging in GitHub

Steps to do Branch Merging:

- Create feature branch from Main/Master branch
- Clone feature branch
- Create new file in Feature branch then commit and push to central repo
- Go to GitHub repo then create pull request to merge feature branch changes to main branch

Note: Once feature branch changes are merged to main branch then we can delete feature branch (if required)

---------- IN This Scenario Merge-Conflicts will Happen ----------

MERGE CONFLICT

Merge conflicts happens, when multiple developers merging/commit the data from different branches in a single file then Git needs your help to decide which changes to include in the final merge. $ git merge <branch-name>

Steps solve merge conflicts by using VIM Editor:

- Open merge conflict file in Vim editor $ vim filename
- Change/modify the data and save the file
- Add file to git $ git add <filename>
- Commit the file $ git commit -m "committing merge file" <filename/.>
- Now u can see merged data.

GIT-STASH

Stashing is used to save the data temporarily for switching b/w the branches.

- Stash Used, when u are switching from one branch to another branch without committing the tracking files.
- Stash applied, only on tracking files and it will not applied for committed/saved files.

Scenario:

Manager assigned task id : 101

I am working on that task (I am middle of the task)

Manager told that stop the work for 101 and complete 102 on priority

Once 102 is completed then resume your work on 101

- When manager is asked us to start 102 task, we have already done few changes for 101 (partially completed)
- We can't push partial changes to repo because with our partial changes existing functionality may break
- We can't delete our changes because we have few hours of time to implement those changes.

---------- IN This Scenario we will go for 'git stash' option ----------

$ git stash is used to save working tree changes to temporary location and make working tree clean.

After priority work completed, we can get stashed changes back using 'git stash apply'

STASH Commands

git stash	To Create stash / Creating temporary files
git stash apply	To get back stash files/Returning temp files
git stash apply stash@{n}	To get specific stash from list
git stash save "message"	To save stash with a message
git stash list	To get list of stash with index
git stash pop	To delete recent one-stash
git stash clear	To clear all stashes

Git-Tag

By using tags, we can tag the commits for further reference.

Git Tag Commands

git tag	To see current tags
git tag <tag-name>	To add new tag
git show <tag-name>	To show tag details
git push origin <tag2>	To push a specific tag to git hub
git tag -d <tag>	To delete a tag

After tagging the commits, we can check the tags in GitHub – open repo – click on tag – you can see tag files – if u open the tag files we get Source code [zip & tar.gz]

You can release from tags and delete the tags.

GIT REVERT

Git revert is a git command that creates a new commit that undoes the changes made by a previous commit.

Git revert is used, when you want undo the changes made by the commit, but you want to keep a record of the fact that you reverted those changes. Simply we can say delete the commit id.

- To use git revert, first you need to identify the commit id that you want to undo.
- You can get the commit id history by using command $ git log
- Once you identified the commit you can use $ git revert <commit_ID>

Git revert is powerful for undoing changes in git.

There are few alternatives

- git reset
- git checkout
- git cherry-pick
- git rebase

GIT-HUB

- Git-Hub is a Centralised Web-Based Platform used for version control.
- Multiple users can used to Store the code and share the code.
- In Git-Hub, The Source code is stored in Repository [Contains project data].
- We have 2 types of repositories i.e., Private and Public
  - Public Repo means everybody can access but we can choose who can modify our repo.
  - Private Repo means we will choose who can access and modify our repo.
- In Git-Hub we store the code in repository.
- In Git-Hub we can create branches. The default branch in GitHub is "main branch"

Create Git-Hub Account

- Search – Github.com
- Enter Email and Create Password – login and continue for free.

Create New Repository

Path – click on profile – your repositories – tap on new – repo name - create new repository.

- Repository name – Any name like PUBG, Paytm…etc
- Description (optional)
- In GitHub we have 2 repositories – Public and Private
- Read me file (optional) – If u choose (Default Main branch)
- Create Repository
- Unique code will be generated with Repo Name (i.e., Repo URL)
- All Developers will connect to Repository Using Repo URL.

To Establish Link b/w Git and Git-Hub

In Git – First select a branch → git remote add origin <GitHub URL>

- GitHub URL → Open Repository – tap on CODE and Copy link.

Push File from Git to Git-Hub $ git push -u origin <selected branch>

- Username – naveensilver
- Password – token [we can generate token in Git-Hub]
- Now automatically every file will Pushed to selected repo in GitHub [only committed files pushed]
- If u want to push another branch files $ git checkout <branch> → git push -u origin <branchName>

To check repository connected/not and which repo connected $ git remote -v

Q) How to get Data/Repo from Git-Hub to Git (Central repo to Local Repo)

- To get Git hub entire repo to our Git server $ git clone <Repo URL>
- To get only changed data in Git Hub $ git fetch && git merge (or) git pull

Git Clone:

Git Clone is Used to Download Git repository from GitHub.

$ git clone <Repo URL>

Note: To get particular branch from Git-Hub $ git clone -b <branchName> <Repo_URL>

Git Pull:

Pull command is used to take latest changes from central repository to local.

When we want to make some changes to code, it is always recommended to take 'git pull'

Alternative command for Git Pull $ git fetch && git merge

Scenario:

Q) Giving Permission to other Git users to do make changes in Public repo:

- Go to Repository > Settings > Collaborators > Add people
- Enter Others GitHub Gmail ID
- The collaborator will get 'Pending Invite' in Email and Need to accept it.
  - Now acts as Collaborator/Other user.
  - Copy the repo link and Clone the repo
  - Make some changes and commit repo.
  - That time we use 'git pull' : To get the latest code repo

Note: when we execute 'git pull' there is a chance of getting conflicts. We need to resolve the conflicts and we should push the code without conflicts.

Git Fork

1. To get someone's entire repository to our GIT-HUB server
   - In Git-Hub > search Git-Hub Id > overall Git-Hub search > users > select repo > FORK > Create fork
2. To get someone's entire repo in our GIT server
   - In Git-Hub > search Git-Hub ID - overall GitHub search - users - select repo – CODE – copy code
   - Now, Go to Git terminal and Run $ git clone <paste URL>

Git Push

Used to push changes from Git local repo to Git central repo.

$ git push -u origin <main/master> (for the first time, we have to mention branch)

$ git push (next time no need to mention branch)

Steps to Push code Local to Central Repo:

- Create one Public repo in GitHub
- Clone GitHub Repo Using "git Clone" Command $ git clone <Repo URL>
- Navigate to Repo folder
- Create one file in Repo folder $ touch Demo.java
- Check status of file using 'git status' command $ git status (untracked file)
- Add file to staging area using 'git add' command $ git add .
- Commit file to git local repo $ git commit -m 'commit-msg'
- Push file from git local repo to git central repo using 'git push' command $ git push

Note: if you're doing 'git push' for first time it will ask your GitHub Credentials.

- When we do git commit then it will generate a commit-id with 40 character length
- From this commit id it will display 7 char in git hub central repo
- We can check commit history using 'git log' command

Q) How To Attach/change Another Repo to Git

- Git connects only single repository (we can attach new repo by removing connected repo)
- Removing attached repo $ git remote rm origin
- Now add another new repo $ git remote add origin <GitHub new repo link>
- Push File from Git to Git-Hub $ git push -u origin <branch> - username and password

Upload/Adding Files in GitHub Branch

- Open repo – add file – upload (local files)/create new files – choose files – commit changes

Creating Folders & Files in GitHub Branch

- Open repo – add file – upload (local files)/create new files →folder1/folder2/…<file1> - commit

Adding data in a file

- In GIT Terminal: $ cat>filename - data add > $ git commit > $ git push → server files
- In Git-Hub: we can add/modify the data (open file > click on right-side pencil > commit changes)

Merge In Git-Hub

In Git-Hub, open repo > compare & pull request > select comparing branches

Here we have two conditions

- If there is no diff b/w two branches – it shows → able to merge – create pull req – merge conform – merged
- If there is any diff in branches – it shows → unable to merge – create pull req – resolve conflicts – edit data – mark as resolve – commit merge – confirm merge – merged

Creating A New Branch in Git-Hub

Suppose we are in master branch in Git-Hub, Click on Master branch > In search bar - type new branch name - click on create branch:<DevOps> from master

Deleting a Branch in Git-Hub

Click on Branch in GitHub – select view all branches – click on delete [bucket symbol]

Git-Hub Commit ID History

$ git log origin/master → To get commit history of Git-Hub

$ git log --oneline --all → To get every branch committed history details

$ git log --graph --oneline --all → To get every branch commit history details in graphical way

Some other commands

$ git remote show origin → To get list of GitHub branches in git server.

$ git push --delete origin <branch name> → To delete GitHub branch in git server.

Git Commands

git help – It will Display Frequently used git commands.

$ git help <Command Name> : It will Open Documentation for given Command

Git Config – It will Configure your Email and Name in Git CLI with Commands

$ git config --global user.email "naveensilver136@gmail.com"

$ git config --global user.name "Naveen Silver"

$ git init : To Initialise our folder as git Working Tree.

$ git status : It will display staged, un-staged and un-tracked files

$ git status

Staged files – The files which are added for commit (.git)

Un-staged files – The files which are modified but not added for commit

Un-tracked files – Newly Created files

Note : To commit a file, we should add to staging area.

$ git add : It is used to add files to staging area

$ git add <filename>

$ git add */.

$ git restore : Used to Un-Stage the files

$ git commit : It is used to commit staged files to git local repo.

$ git commit -m "reason for commit"

$ git remote add :

Git Merge vs Git Rebase

These commands are used to merge changes from one branch to another branch

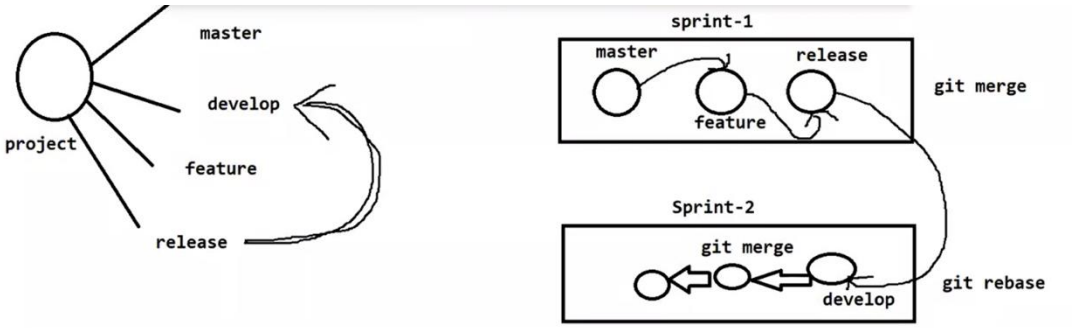
Git merge will maintain commit history

Git rebase will not maintain that rebase history

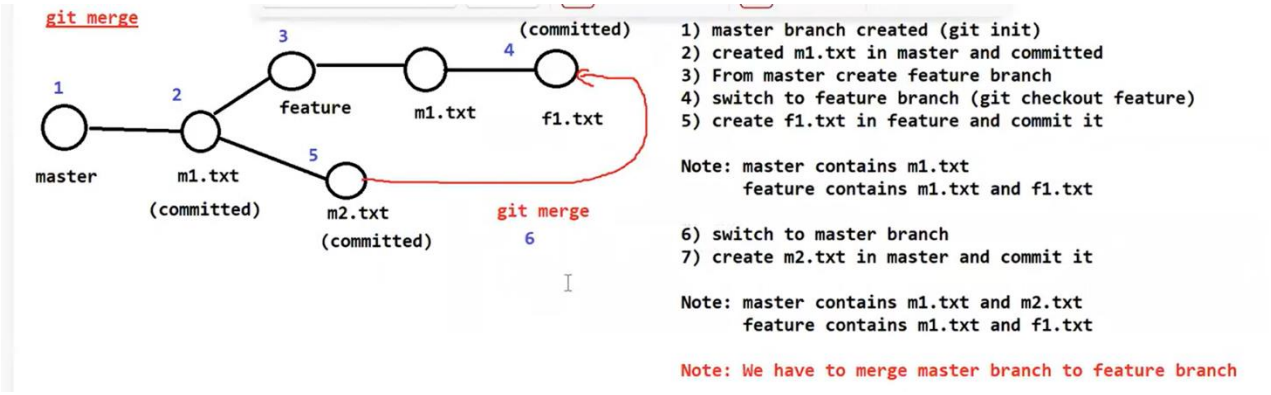
When we are working on particular sprint and we want to merge changes from one branch to another branch then we will use 'git merge' command

Once sprint-1 is delivered then we want to take latest code of sprint-1 to start sprint-2 development.

In this scenario we don't need commit history so we will use 'git rebase' command



Git Merge:



Currently I am in the Feature Branch,

$ git merge master

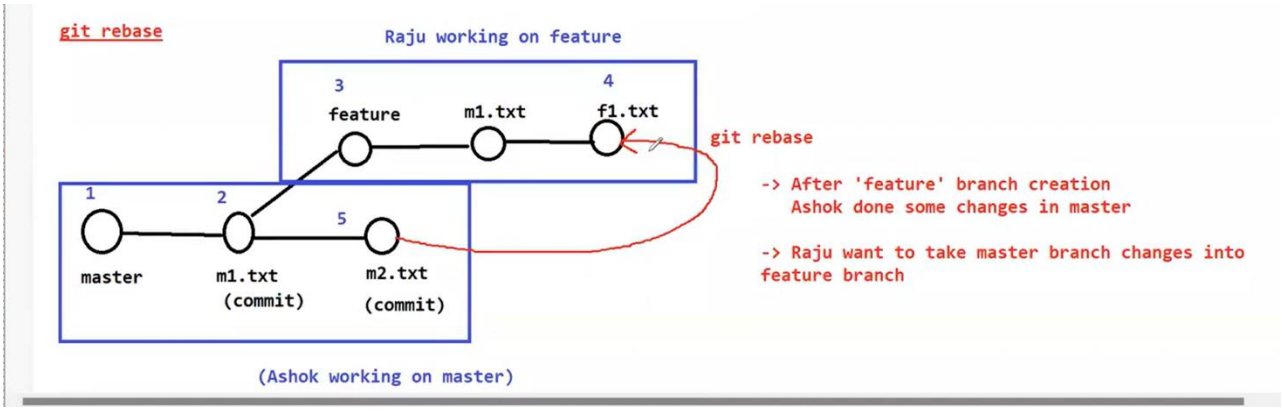
Save the file :wq

M2.txt will created in Master branch

$ git log – we can see all the committed history details.

```
ashok@DESKTOP-BDGOOU7 MINGW64 ~/classes/01-DevOps/git-work (feature)
$ git log
commit 62cec291847d05b89b2dbef9084bcfee9b2e01b1 (HEAD -> feature)
Merge: 2174744 0be0086
Author: Ashok <ashokitschool@gmail.com>
Date:   Thu May 26 21:26:24 2022 +0530

    Merge branch 'master' into feature

commit 0be0086f1208a8206220592a13332a7a1a7d8ee7 (master)
Author: Ashok <ashokitschool@gmail.com>
Date:   Thu May 26 21:19:50 2022 +0530

    added m2.txt

commit 217474450e03427c836a331ab74a833933832639
Author: Ashok <ashokitschool@gmail.com>
Date:   Thu May 26 21:11:46 2022 +0530

    added f1.txt

commit 88bbddf9d0d73083e4a019f8cb90d9129f44ec82
Author: Ashok <ashokitschool@gmail.com>
Date:   Thu May 26 21:07:39 2022 +0530

    added m1.txt

ashok@DESKTOP-BDGOOU7 MINGW64 ~/classes/01-DevOps/git-work (feature)
$
```

GIT Rebase :



Currently I am in the Feature Branch,

$ git rebase master

Save the file :wq

M2.txt will created in Master branch

$ git log – we can see all the committed history details.

```
ashok@DESKTOP-BDGOOU7 MINGW64 ~/classes/01-DevOps/git-work (feature)
$ git rebase master
Successfully rebased and updated refs/heads/feature.

ashok@DESKTOP-BDGOOU7 MINGW64 ~/classes/01-DevOps/git-work (feature)
$ git log
commit 5a49195a3800d56f335311fdbbbbc7bb74dc66cc (HEAD -> feature)
Author: Ashok <ashokitschool@gmail.com>
Date:   Thu May 26 21:38:40 2022 +0530

    added f1.txt

commit 91613f63194aa3b5e45a594668467249087a5aab (master)
Author: Ashok <ashokitschool@gmail.com>
Date:   Thu May 26 21:40:13 2022 +0530

    added m2.txt

commit 9edec000ea07c073f2600e89d6cdb8c4143a324a
Author: Ashok <ashokitschool@gmail.com>
Date:   Thu May 26 21:36:26 2022 +0530

    added m1.txt

ashok@DESKTOP-BDGOOU7 MINGW64 ~/classes/01-DevOps/git-work (feature)
$
```