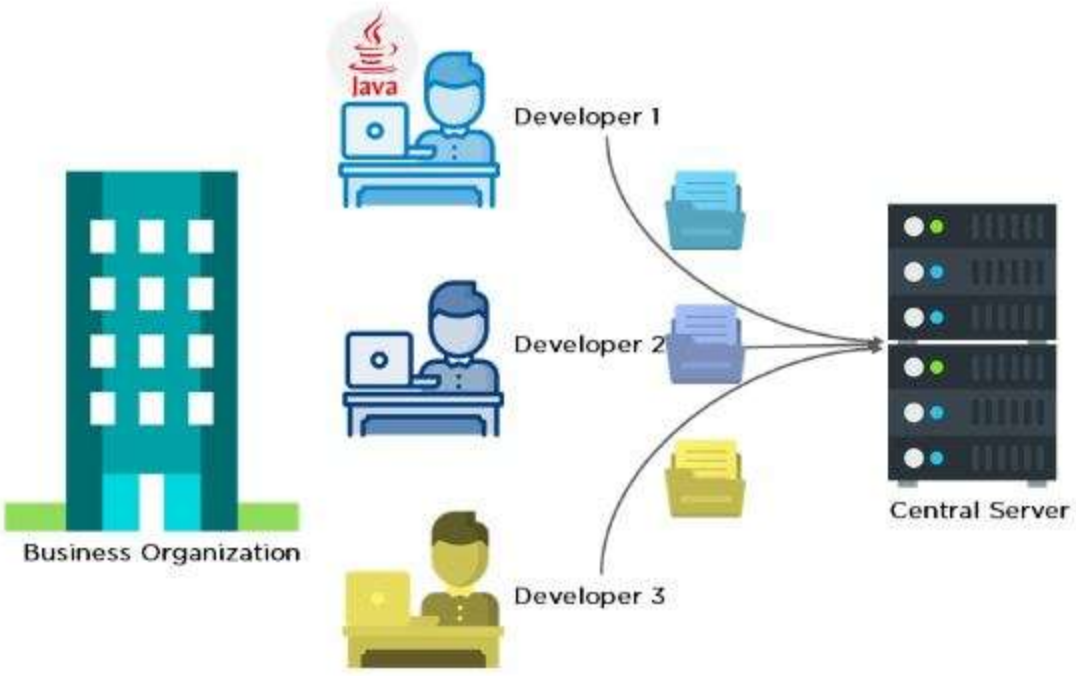
# GIT

## GIT- GLOBAL INFORMATION TRACKING

- Git is a DevOps tool used for source code management.
- It is a free and open-source version control system used to handle small to very large projects efficiently.
- Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development.
- Linus Torvalds created Git in 2005 for the development of the Linux kernel

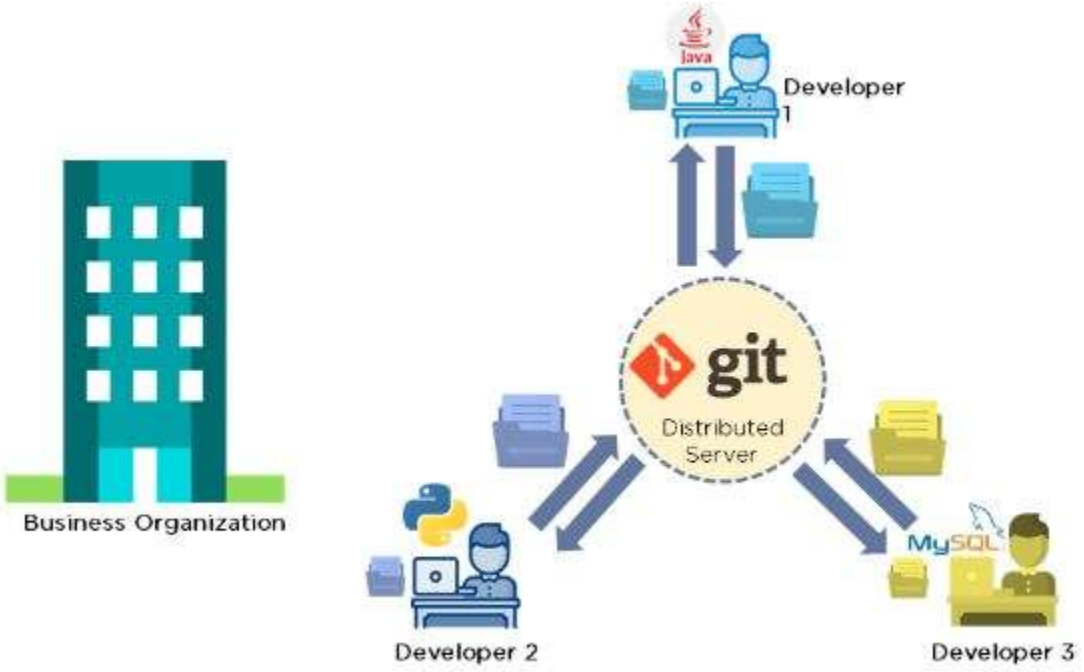
**Scenario before GIT:**

- Developers used to submit their codes to the central server without having copies of their own.
- Any changes made to the source code were unknown to the other developers.
- There was no communication between any of the developers.



**Scenario after GIT:**

- Every developer has an entire copy of the code on their local systems
- Any changes made to the source code can be tracked by others
- There is regular communication between the developers




**What is Git?**

Git is a version control system used for tracking changes in computer files. It is generally used for source code management in software development.

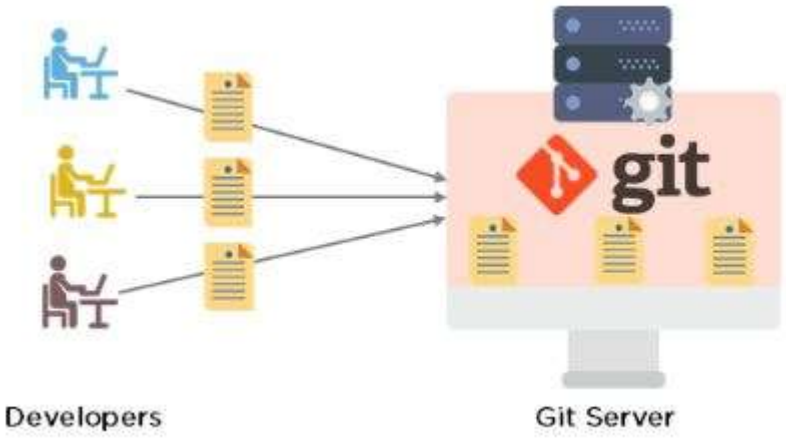
- Git is used to tracking changes in the source code
- The distributed version control tool is used for source code management
- It allows multiple developers to work together
- It supports non-linear development through its thousands of parallel branches

**Why Git?**

- Developers can work together from anywhere in the world.
- Developers can see the full history of the project.
- Developers can revert to earlier versions of a project.

**Features of Git**

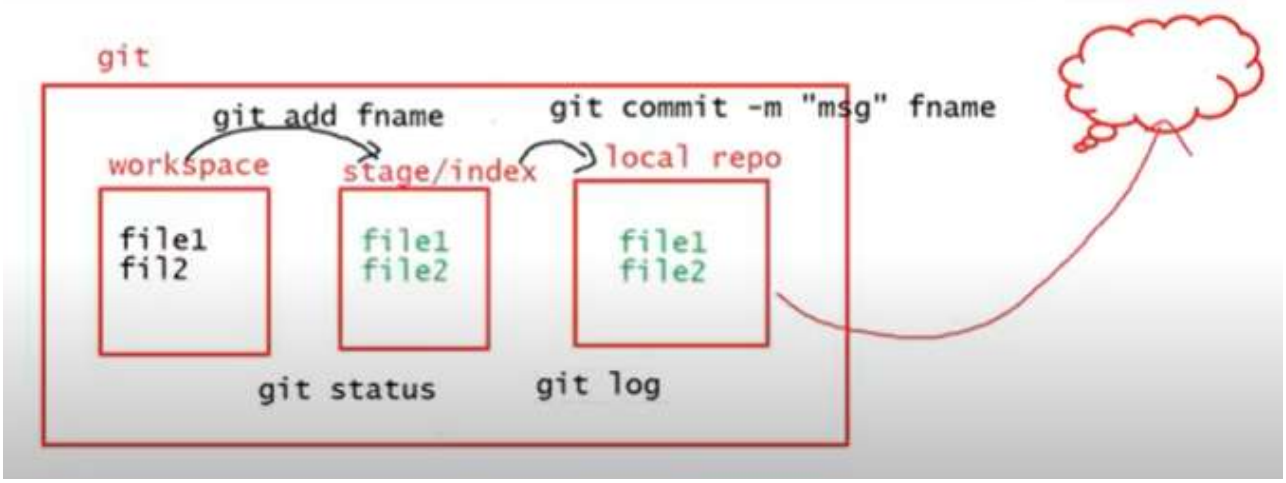
- Tracks history
- Free and open source
- Supports non-linear development
- Creates backups
- Scalable
- Supports collaboration
- Branching is easier
- Distributed development




**GIT Flowchart:**

The Git workflow is divided into three states:

- Working directory - Modify files in your working directory
- Staging area (Index) - Stage the files and add snapshots of them to your staging area
- Git directory (Repository) - Perform a commit that stores the snapshots permanently to your Git directory. Checkout any existing version, makes changes, stage them and commit.




**Install Git Bash On Windows**

1. Open your browser and visit Git Bash downloads page by clicking the following link: Git Download Page
2. Once the page loads, click the **Download for Windows** button.
3. Next, you will be redirected to a different page and click on **Click here to download** the latest (**2.42.0**) **64-bit** version of **Git for Windows**
4. Go into your **Downloads** directory or the location where your browser download things in. Click the Git Bash executable (.**exe** file) to run the installer.
5. Windows will prompt you with the message "Do you want to allow this app to make changes to your device?", click **Yes**
6. Now click on next next and click on install
7. After the installation has finished, select **Launch Git Bash** and click **Finish** to launch Git Bash.
8. The Git Bash terminal will now open and you will be able to enter Git and Bash commands.
9. Check whether git installed properly or not by Right-click on any folder, anywhere and it will have the launch Git Bash option on the context menu.
10. Click on git bash here and check git version by command **git --version** tells you the Git version currently installed on your system.

## Basic linux commands

| | |
|---|---|
| ls<br>ls –a ,ls –lrth,ls –la,ll | Displays information about files in the current directory. |
| pwd | Displays the current working directory |
| touch filename | Creates empty file |
| mkdir dirname | Creates folder |
| cd<br>cd dirname,cd . . | To navigate into different folders |
| rm –rf <filename> or <dirname> | Delete files and directories |
| cat | Display file contents on terminal |
| Clear or ctrl+l | Clear terminal |
| Ctrl+c | cancels or terminates the currently-running program |
| Ctrl++ | To enlarge the gitbash screen |

## GIT COMMANDS

| | |
|---|---|
| git init | to start initialize new empty repository |
| git config --global user.name “user name”<br>git config --global user.email “user email” | Sets the author name and email address respectively to be used with your commits. |
| git config --list | List the all configurations |
| touch <filename><br>touch filename1 filename2…. | Creates a file |
| git status | List which files are staged, unstaged, and untracked. |
| git add filename or git add . | To add files to the staging area. |
| git commit –m “message” | To describe what changes are done in in file |
| git log | To view the history of commits for a repository |

| | |
|---|---|
| git clone <url><br>eg:git clone https://github.com/devopstrainginghub/b12projectrepo.git | Clone (download) a repository that already exists on GitHub, including all of the files, branches, and commits |
| git push | Push the changes done to a file to central repository |
| git show <commit id> | the log message and the changes that occurred in this specific commit |
| git pull | It is used to pull all changes from a remote repository into the branch you are working on |
| git pull --rebase | a method of combining your local unpublished changes with the latest published changes on your remote |
| git fetch | the command that tells the local repository that there are changes available in the remote repository without bringing the changes into the local repository.It will be in origin |
| git merge | It merges changes from origin to local repository |
| git branch | a branch is a new/separate version of the main repository |
| git tag | To specify the milestone of project |
| git fork | A fork is a rough copy of a repository. Forking a repository allows you to freely test and debug with changes without affecting the original project |
| git commit --amend -m “commit msg to change ” | Commit --amend is used to modify the most recent commit. It combines changes in the staging environment with the latest commit, and creates a new commit. This new commit replaces the latest commit entirely. |

**Class notes**

git init

git config --global user.name "madhukiran"

git config --global user.email "devopstraininghub@gmail.com"

git config –list

---

touch file1

git status --> Untracked , working directory , red

git add --> work space to staging

git status --> staging , Green

git commit -m "msg" file1

git status

git log

---

**Commands to create and push to central repository**

git clone https://github.com/devopstraininghub/b12projectrepo.git

ll

cd b11projectrepo

touch madhu

git add .

git commit -m "madhu commit"

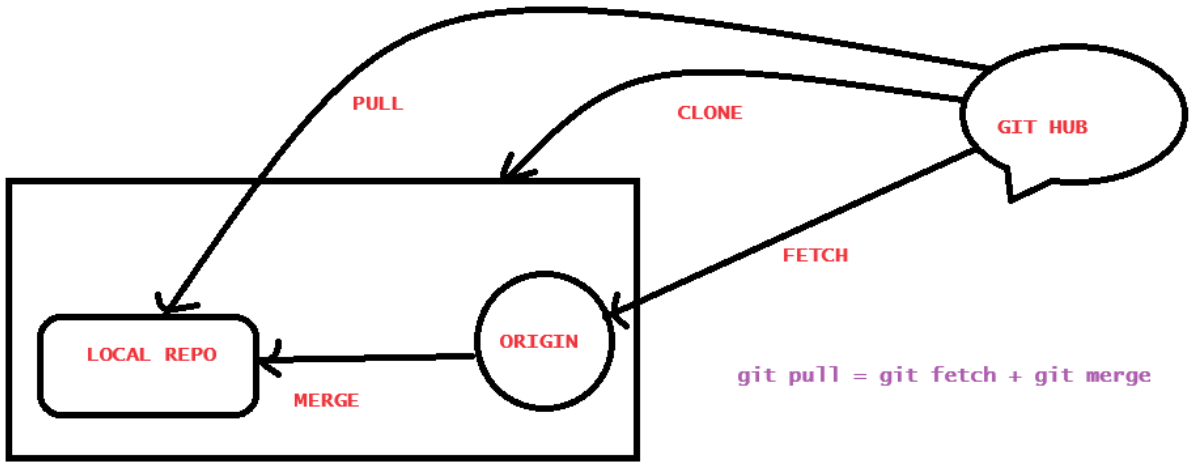
git push

git pull --rebase

git push

## Git Pull / Pull Request

The term pull is used to receive data from GitHub. It fetches and merges changes from the remote server to your working directory. The git pull command is used to pull a repository.




### Differences between git fetch and git pull

To understand the differences between fetch and pull, let's know the similarities between both of these commands. Both commands are used to download the data from a remote repository. But both of these commands work differently. Like when you do a git pull, it gets all the changes from the remote or central repository and makes it available to your corresponding branch in your local repository. When you do a git fetch, it fetches all the changes from the remote repository and stores it in a separate branch in your local repository. You can reflect those changes in your corresponding branches by merging.

So basically,

git pull = git fetch + git merge

## Git Fetch vs. Pull

Some of the key differences between both of these commands are as follows:

| git fetch | git pull |
|---|---|
| Fetch downloads only new data from a remote repository. | Pull is used to update your current HEAD branch with the latest changes from the remote server. |
| Fetch is used to get a new view of all the things that happened in a remote repository. | Pull downloads new data and directly integrates it into your current working copy files. |
| Fetch never manipulates or spoils data. | Pull downloads the data and integrates it with the current working file. |
| It protects your code from merge conflict. | In git pull, there are more chances to create the merge conflict. |
| It is better to use git fetch command with git merge command on a pulled repository. | It is not an excellent choice to use git pull if you already pulled any repository. |

# GITHUB

GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.

## GITHUB ACCOUNT CREATION

1. Open https://github.com in a web browser, and then select Sign up.
2. Enter your email address.
3. Create a password for your new GitHub account, and Enter a username, too. Next, choose whether you want to receive updates and announcements via email, and then select Continue.
4. Verify your account by solving a puzzle. Select the Start Puzzle button to do so, and then follow the prompts.
5. After you verify your account, select the Create account button.
6. Next, GitHub sends a launch code to your email address. Type that launch code in the Enter code dialog, and then press Enter.
7. GitHub asks you some questions to help tailor your experience. Choose the answers that apply to you in the following dialogs:

    - How many team members will be working with you?
    - What specific features are you interested in using?

8. On the Where teams collaborate and ship screen, you can choose whether you want to use the Free account or the Team account. To choose the free account, select the Skip personalization button.
9. GitHub opens a personalized page in your browser.
10. Create an own repository.

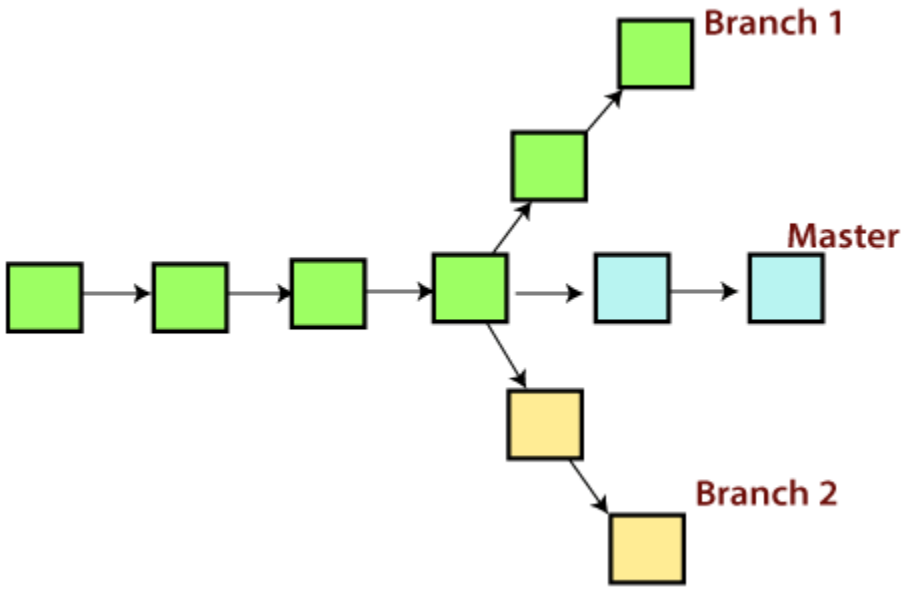
## Difference between GIT and GITHUB

| S.No. | Git | GitHub |
|---|---|---|
| 1. | Git is software. | GitHub is a service. |
| 2. | Git is a command-line tool | GitHub is a graphical user interface |
| 3. | Git is installed locally on the system | GitHub is hosted on the web |
| 4. | Git is maintained by Linux. | GitHub is maintained by Microsoft. |
| 5. | Git is focused on version control and code sharing. | GitHub is focused on centralized source code hosting. |
| 6. | Git is a version control system to manage source code history. | GitHub is a hosting service for Git repositories. |
| 7. | Git was first released in 2005. | GitHub was launched in 2008. |
| 8. | Git has no user management feature. | GitHub has a built-in user management feature. |
| 9. | Git is open-source licensed. | GitHub includes a free-tier |

| S.No. | Git | GitHub |
|---|---|---|
| | | and pay-for-use tier. |
| 10. | Git has minimal external tool configuration. | GitHub has an active marketplace for tool integration. |
| 11. | Git provides a Desktop interface named Git Gui. | GitHub provides a Desktop interface named GitHub Desktop. |
| 12. | Git competes with CVS, Azure DevOps Server, Subversion, Mercurial, etc. | GitHub competes with GitLab, Bit Bucket, AWS Code Commit, etc. |

# GIT BRANCH

A branch is a version of the repository that diverges from the main working project. It is a feature available in most modern version control systems. A Git project can have more than one branch. These branches are a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug, you spawn a new branch to summarize your changes. So, it is complex to merge the unstable code with the main code base and also facilitates you to clean up your future history before merging with the main branch.




## Git Master Branch

The master branch is a default branch in Git. It is instantiated when first commit made on the project. When you make the first commit, you're given a master branch to the starting commit point. When you start making a commit, then master branch pointer automatically moves forward. A repository can have only one master branch.

Master branch is the branch in which all the changes eventually get merged back. It can be called as an official working version of your project.

**Branch Operations**

| | |
|---|---|
| git branch | A branch is a version of the repository that diverges from the main working project |
| git branch <branch name> | Creates a new branch with specified name |
| git branch --list | List all of the available branches in your repository |
| git branch -d<branch name><br><br>git branch -D<branch name> | To delete the branch in you repository<br><br>To delete the branch forcefully in you repository |
| git checkout<branch name> | switch from master to any other branch available on your repository |
| git branch –m<br><old branch name><new branch name> | Rename the branch |
| git merge | Used to combine two branches. |
| git push origin <branch name> | To push branch to central repo |
| git cherry-pick <commit id> | To push only one file to master from branch |

**Class commands**

GIT BRANCH

git branch

git branch <br.name>

git checkout <br.name>

git checkout -b <br.name>

git merge <br.name>

git push origin <br.name>

git push https://github.com/devopstrainginghub/b12projectrepo.git <br.name>

git branch -d <br.name>

git branch -D <br.name>

git push origin -d <br.name>

**Git Tags**

Tags make a point as a specific point in Git history. Tags are used to mark a commit stage as relevant. We can tag a commit for future reference. Primarily, it is used to mark a project's initial point.

Tags are much like branches, and they do not change once initiated. We can have any number of tags on a branch or different branches. The below figure demonstrates the tags on various branches.

**When to create a Tag:**

- When you want to create a release point for a stable version of your code.
- When you want to create a historical point that you can refer to reuse in the future.

**Tag commands**

| | |
|---|---|
| git tags | mark the milestone |
| git tag <tag name> | Creates a new tag with specified name |
| git tag | List all of the available tag in your repository |
| git tag –d <tag name> | To delete the tag in you repository |
| git checkout <tag name> | switch from master to any other branch available on your repository |
| git push origin<tag name> | To push tag to central repo |

| | |
|---|---|
| git tag –m <old tag name><new tag name> | To rename the tag |
| git tag <tag name> -m "< Tag message>" | Create a tag with a message. This tags contain some additional information like author name and other project related information |

## Git commands

| Git Inspecting changes commands | |
|---|---|
| git log --oneline | one commit per line .the first seven characters of the SHA |
| git log --oneline -n 7 | it displays first 7 commits only |
| git log –n 7 --oneline | |
| git log –stat | display the files that have been modified |
| git log –graph | viewing your git log as a graph |
| git log –since "dd-mm-yyyy" | display all the commits made from the given date to till date |
| git log –after="dd-mm-yyyy" –before="dd-mm-yyyy " | display the commits made between the dates |
| git log –author "author name" | displays the commit made by specified author |
| git diff <commit-id1><commit-id2> | display the changes between two commits |
| git diff <branch1><branch2> | display the changes between two branches |
| git alias<br><br>git config --global alias.new name "git command"<br><br>eg:<br><br>git status<br><br>git config --global alias.ss "status" | Aliases are the user-friendly names given by a us to a git command so that the we doesn't need to memorize the whole lengthy syntax of a git commands.<br><br>now git status command is replaced by ss |

| | |
|---|---|
| | |

## Git Stash

Sometimes you want to switch the branches, but you are working on an incomplete part of your current project. You don't want to make a commit of half-done work. Git stashing allows you to do so. The git stash command enables you to switch branches without committing the current branch.

Generally, the stash's meaning is "store something safely in a hidden place." The sense in Git is also the same for stash; Git temporarily saves your data safely without committing.

Stashing takes the messy state of your working directory, and temporarily save it for further use. Many options are available with git stash.

| **git stash commands** | |
|---|---|
| git stash | The work is saved with git stash command |
| git stash save “stash message” | The changes can be stashed with a message |
| git stash list | To check the stored stashes |
| git stash apply | Restores the last/recent stash and keeps the stash in the stack after it is applied |

| | |
|---|---|
| git stash pop | Git stash pop command is quite similar to git stash apply. The main difference between both of these commands is stash pop command that deletes the stash from the stack after it is applied |
| git stash drop | To delete a stash from the queue |
| git stash clear | It will delete all the stashes that exist in the repository |
| git stash branch <branch name> | Create a new branch and transfer the stashed work on that |

## Git reset

The term reset stands for undoing changes. The git reset command is used to reset the changes. The git reset command has two core forms of invocation. These forms are as follows.

- Soft
- Mixed

| | |
|---|---|
| git reset | Reset the status of the Head, and it will not delete any data from the staging area to match the position of the Head |
| git reset HEAD fname | To move from changes staging to workspace |
| git reset --soft commit id(previous commit id ) | To move from local repos to staging |
| git reset --mixed cid(previous commit id) | To move changes from local repos to workspace |

# Git Rebase vs. Merge:

## What Is Git Rebase?

Git rebase is a command that lets users integrate changes from one branch to another, and the logs are modified once the action is complete. Git rebase was developed to overcome merging's shortcomings, specifically regarding logs.

## What Is Git Merge?

Git merge is a command that allows developers to merge Git branches while the logs of commits on branches remain intact.

## The Working of Git Rebase and Merge

Git rebase takes all the changes, and compresses them into a single patch and integrates this new patch onto the target branch. Then, it moves the completed work from one branch to another, typically the master branch. In the process, rebase flattens the history, removing unwanted entries.

Git merge, on the other hand, only changes the target branch and creates a commit, preserving the history of the source branch.

## Git Rebase and Git Merge Similarities

Both the rebase and merge commands exist for the same reason: they combine the work of multiple developers into a single entity, integrating changes between branches. That about sums up their similarities. On the other hand, their differences are considerably more profound.

## What's the Difference between Merge and Rebase?

The main difference between git merge and git rebase is that git merge is a way of combining

changes from one branch (source branch) into another branch (target branch) where as git rebase is a way of moving the changes from one branch onto another branch.

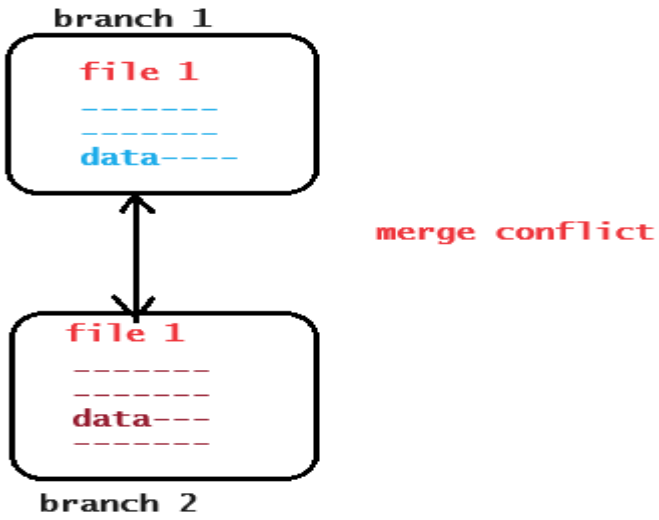
A head-to-head comparison chart is the best way to illustrate the differences between Git merge and Git rebase.

| **Merge** | **Rebase** |
|---|---|
| Git Merge lets you merge different Git branches. | Git Rebase allows you to integrate the changes from one branch into another. |
| Git Merge logs show you the complete history of commit merging. | Git Rebase logs are linear. As the commits are rebased, the history is altered to reflect this. |
| All the commits on a feature branch are combined into a single commit on the master branch. | All commits are rebased, and the same number of commits is added to the master branch. |
| Merge is best used when the target branch is supposed to be shared. | Rebase is best used when the target branch is private. |
| Merge preserves history. | Rebase rewrites history. |

**Git Merge Conflict**

When two branches are trying to merge, and both are edited at the same time and in the same file, Git won't be able to identify which version is to take for changes. Such a situation is called merge conflict. If such a situation occurs, it stops just before the merge commit so that you can resolve the conflicts manually.

Below figure shows te merge conflict:




```
Madhu Kiran@Madhukiran MINGW64 ~/Desktop/20oct (master)
$ git merge rel
Auto-merging hero
CONFLICT (add/add): Merge conflict in hero
Automatic merge failed; fix conflicts and then commit the result.

Madhu Kiran@Madhukiran MINGW64 ~/Desktop/20oct (master|MERGING)
$
```

**Merge conflict solution**

```
<<<<<<< HEAD
ramesh is a hero

pavani is a hero

david is a hero
=======
Kishore is a Hero

Mamata is a hero

Ahmad is hero
>>>>>>> rel
~
~
```

To see that which is the first edited text of the merge conflict in your file, search the file attached with conflict marker <<<<<<<. You can see the changes from the **HEAD** or base branch after the line <<<<<<< **HEAD** in your text editor. Next, you can see the divider like =======. It divides your changes from the changes in the other branch, **followed by** >>>>>>> **BRANCH-NAME**

Decide whether you want to keep only your branch's changes or the other branch's changes, or create a new change. Delete the conflict markers <<<<<<<, =======, >>>>>>> and create final changes you want to merge.

**Differnce between GIT and SVN**

| **GIT** | **SVN** |
|---|---|
| Git is open source distributed vice control system developed by Linus Torvalds in 2005. It emphasis on speed and data integrity | Apache Subversion is an open source software version and revision control system under Apache license. |
| Git has a Distributed Model. | SVN has a Centralized Model. |
| In git every user has their own copy of code on their local like their own branch. | In SVN there is central repository has working copy that also make changes and committed in central repository. |
| In git we do not required any Network to perform git operation. | In SVN we required Network for runs the SVN operation. |
| Git is more difficult to learn. It has more concepts and commands. | SVN is much easier to learn as compared to git. |
| Git deals with large number of files like binary | SVN control the large number of binary |

| GIT | SVN |
| --- | --- |
| files that change quickly that why it become slow. | files easily. |
| In git we create only .git directory. | In SVN we create .svn directory in each folder. |
| It does not have good UI as compared to SVN. | SVN has simple and better user interface . |
| Features of GIT:<br>• Distributed System.<br>• Branching.<br>• Compatibility.<br>• Non-linear Development.<br>• Lightweight.<br>• Open source. | Features of SVN:<br>• Directories are versioned<br>• Copying, deleting, and renaming.<br>• Free-form versioned metadata .<br>• Atomic commits.<br>• Branching and tagging.<br>• Merge tracking.<br>• File locking. |

# VI EDITOR

## What is Vi ?

The vi editor is elaborated as visual editor. It is installed in every Unix system. In other words, it is available in all Linux distros. It is user-friendly and works same on different distros and platforms. It is a very powerful application. An improved version of vi editor is vim.

The vi editor has two modes:

- Command Mode: In command mode, actions are taken on the file. The vi editor starts in command mode. Here, the typed words will act as commands in vi editor. To pass a command, you need to be in command mode.
- Insert Mode: In insert mode, entered text will be inserted into the file. The Esc key will take you to the command mode from insert mode.

By default, the vi editor starts in command mode. To enter text, you have to be in insert mode, just type 'i' and you'll be in insert mode. Although, after typing i nothing will appear on the screen but you'll be in insert mode. Now you can type anything.

To exit from insert mode press Esc key, you'll be directed to command mode.

## Using vi

The vi editor tool is an interactive tool as it displays changes made in the file on the screen while you edit the file.

In vi editor you can insert, edit or remove a word as cursor moves throughout the file.

Commands are specified for each function like to delete it's x or dd.

The vi editor is case-sensitive. For example, p allows you to paste after the current line while P allows you to paste before the current line.

**vi syntax: vi <filename>**

In the terminal when you'll type vi command with a file name, the terminal will get clear and content of the file will be displayed. If there is no such file, then a new file will be created and once completed file will be saved with the mentioned file name.

Let's understand vi through an example:

To start vi open your terminal and type vi command followed by file name. If your file is in some other directory, you can specify the file path. And if in case, your file doesn't exist, it will create a new file with the specified name at the given location.

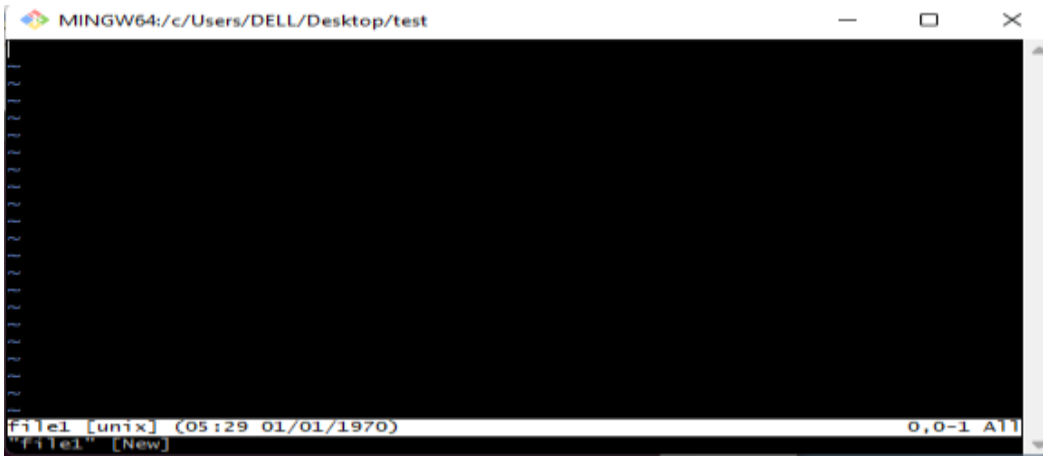
Example:

1. vi file


Look at the above snapshot; we are creating a new file file

**Command mode**

This is what you'll see when you'll press enter after the above command. If you'll start typing, nothing will appear as you are in command mode. By default vi opens in command mode.




Look at the above snapshot; it is blank as it is a new file. To start typing, you have to move to the insert mode. At the end of the terminal window, directory name and file name are displayed.

**Insert mode**

To move to the insert mode press **esc i .**


Look at the above snapshot, after pressing i we have entered into insert mode. Now we can write anything. To move to the next line press enter.

Once you have done with your typing, press esc key to return to the command mode.

**To save and quit**

You can save and quit vi editor from command mode. Before writing save or quit command you have to press colon (:wq) .

**exit vi table commands:**

| Commands | Action |
|---|---|
| :wq | Save and quit |
| :w | Save |
| :q | Quit |
| :w fname | Save as fname |
| :wq! | save and quit forcefully |
| :q! | Quit discarding changes made |
| :w! | Save (and write to non-writable file) |

To exit from vi, first ensure that you are in command mode. Now, type :wq and press enter. It will save and quit vi.

Type :wq to save and exit the file


Look at the above snapshot, command : wq will save and quit the vi editor. When you'll type it in command mode, it will automatically come at bottom left corner.

## REFEERNCES

https://www.atlassian.com/git/glossary#commands

https://git-scm.com/docs

https://www.javatpoint.com/git-commands

## GIT INTERVIEW QUESTIONS

https://www.javatpoint.com/git-interview-questions

https://www.simplilearn.com/tutorials/git-tutorial/git-interview-questions

+917732056555 -Hema
+919160179395-Santhosh
+917396627149-Madhukiran
devopstraininghub@gmail.com