# GitHub:

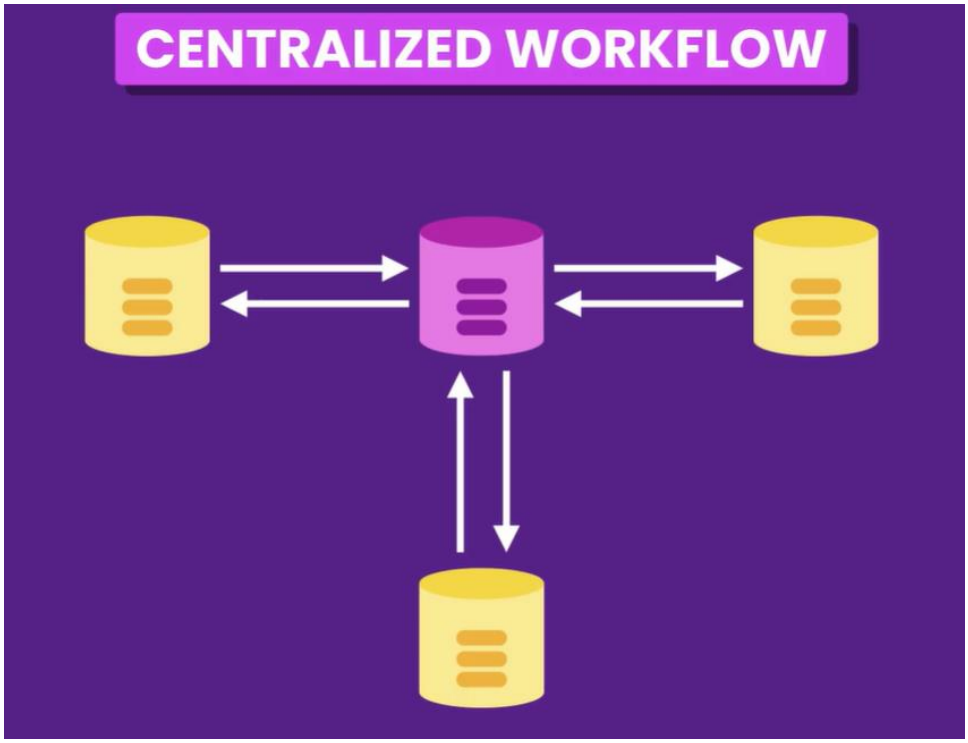
Git is the most popular version control system in world. Git record the changes that it made to our code overtime in special database called repository. We can look into history and look what changes has happened and if something screwed up, we can easily, revert the project back to an earlier state. So, with version control system we can track the project history and work together with other developers.

Git is the most popular program for version control system, and it has many pros such as:

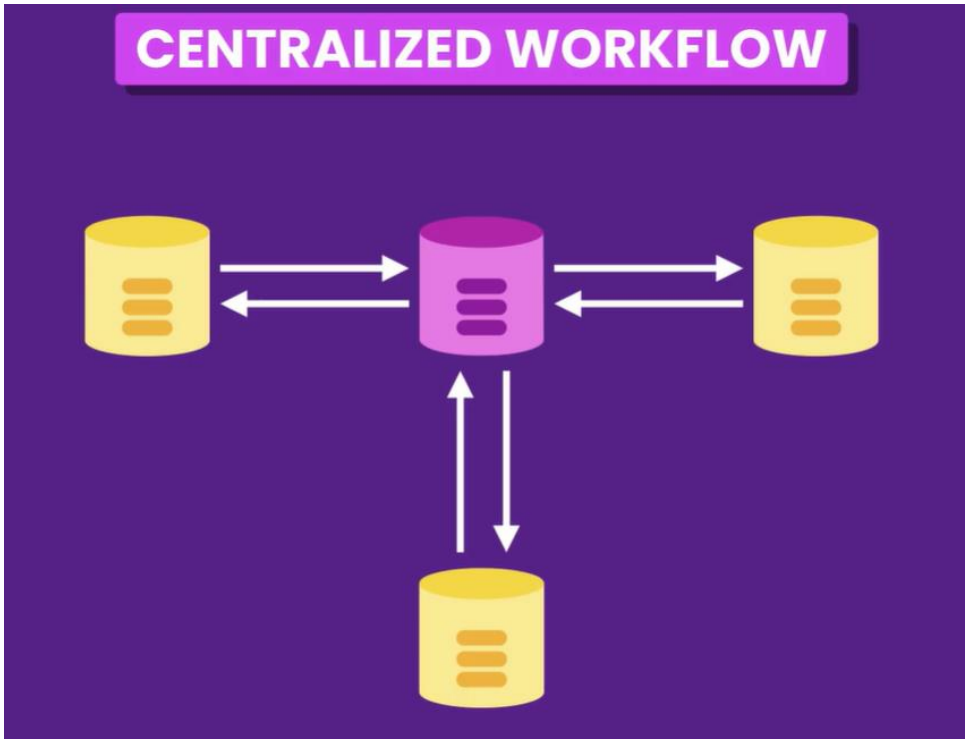
1. Free
2. Open source
3. Super-fast
4. Scalable
5. Cheap branching/merging

Version control system has two types: Centralized and Distributed

Centralized: All team members connect with central server to get the latest copy of the code and to share the changes with others. The only problem with centralized server is, if single failure happens and server goes offline, we cannot save collaborate or save snapshots of the project.

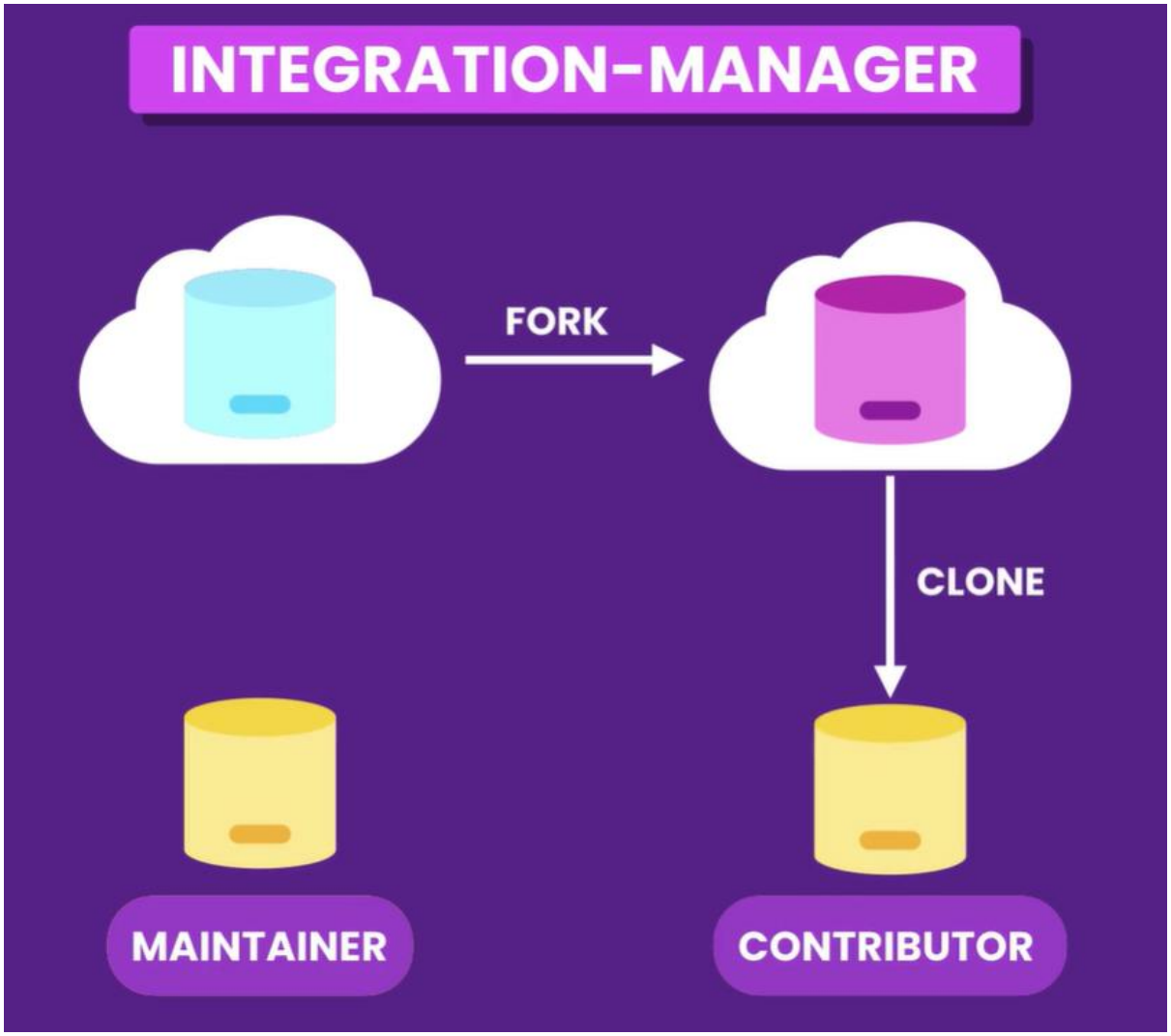



Distributed: In distributed server we don't have this problem because every member of the team would have a copy with its history on his machine. So, we can save snapshots of the project and if the server is offline, we can synchronise the work directly with other manually.

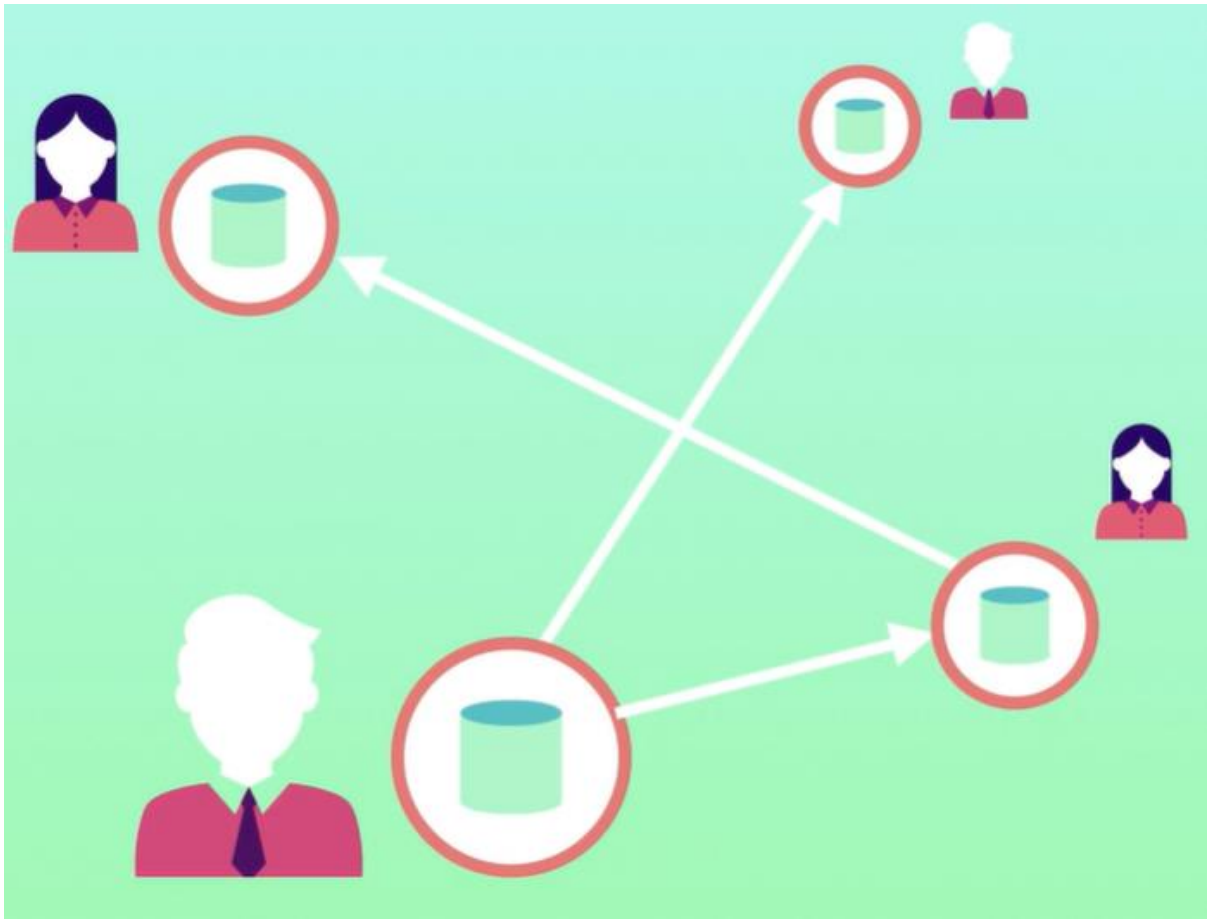



How it works: So, git works actually following the steps we clone the repository from git to cloud or private server.  Then we make updates, bug fixes, and changes. Then we commit it to the staging area after that we push it the cloud or private server.


Git Rules: Of course, each company has its own rules and policies. Some companies have maintainer this person is responsible for doing the push to latest version when he thinks everything is really good and working. Here is example how workflow works. As you can see. This collaboration called Integration-manger.




Other companies they can have their own rules and they decide the way the workflow work. They really have to communicate really well to avoid problems and confusion.



# Git commands:

Git has really a lot of commands I tried here to put the best once that are mostly used in list on left side the command self and right side is the reason why. In summary Command file you can see all command I have experienced while I did this Git course.

| Short Cuts: (Command most be used in small letters.) | Why. |
|---|---|
| CMD | To open Command prompt. |
| GIT --VERSION | To check which version you have on the machine. |
| GIT CONFIG --SYSTEM | System for all users. |
| GIT CONFIG --LOCAL | Global all repositories of the current user. |
| GIT CONFIG --GLOBAL | Local the current repository. |
| GIT CONFIG --GLOBAL CORE.EDITOR | To set the default editor. |
| GIT CONFIG –GLOBAL –E | To open all default editor to edit all global settings. |
| GIT CONFIG --HELP | To open browser. |
| GIT CONFIG -H | Short summary on cmd. |
| MKDIR (NAME OF THE FILE) | To create a file. |
| CD | To get into the file. |
| Git init | To initialise a file. |

| | |
|---|---|
| dir./d<br>dir /a<br>dir ./a<br>dir /ad<br>dir \h | To open and look inside directory file. |
| RM –RF (NAME) | To remove a file. |
| GIT ADD. | To add or updating the staging area. |
| GIT COMMIT –M “Meaning full message” | To commit file to the staging area. |
| GIT STATUS | To check the status of the repository/directory. |
| GIT COMMIT –AM ” Meaning full message” | To commit everything in once and escaping the stage area. |
| GIT LS-FILES | To look into the files in directory. |
| GIT RM (NameOfTheFile) *. (filesort) | To remove file from directory and stage area at same time. |
| MV (FileName) (NewName) | To Rename a file |
| GIT MV (FileName) (NewName) | To rename a file right way without updating to stage area. |
| .gitignore | To create file that ignores the useless file and does not update to the stage area or repository |
| GIT RM –CACHED (FileName) | To remove a file from staging area. |
| GIT STATUS –S | To inspect status in directory and repository. Its stands for short status actually. |
| GIT DIFF --STAGED | To see what we have in staging area and what is going in next commit. |
| GIT DIFF | To look at current changes in your working copy, past changes in commits |
| GIT DIFFTOOL | To view the changes. |
| GIT DIFFTOOL --STAGED | To view the changes in staged area. |
| GIT LOG | To view the history logs. |
| GIT LOG --ONELINE | For a short summary of the commit. |
| GIT LOG --ONELINE --REVERSE | To see the first commit to last commit. |

# Log:

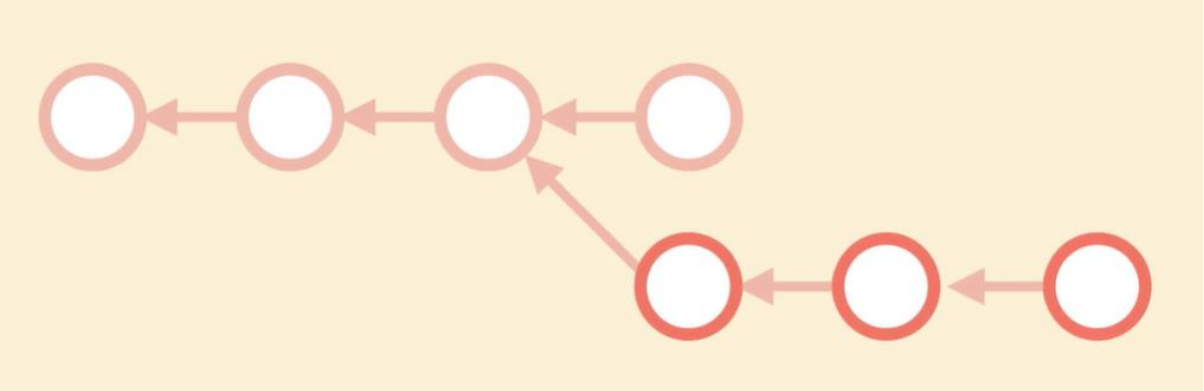
Git log is to show the commit logs and via logs you can do see for example: the Author, date and the commit code. Git log has wiled futures with git log –h you can look the futures that you can use.

```
C:\Users\abood\Desktop\C#-progress>git log
commit ed23d5b835b17104f87f7facae5409a224e9cc9a (HEAD -> master)
Author: AbdulRahman Bani Almarjeh <aboodaysy123@gmail.com>
Date:   Thu Mar 7 18:16:48 2024 +0100

    deleting ignorefiel

commit 6e0a5e5b4643fba39cfd758f5e9dd53bc9b50f8b
Author: AbdulRahman Bani Almarjeh <aboodaysy123@gmail.com>
Date:   Thu Mar 7 17:56:22 2024 +0100

    testing igonre file

commit 1b8c3ca26fb5f82b8b0af344cc3a3db5b3f74ff8
Author: AbdulRahman Bani Almarjeh <aboodaysy123@gmail.com>
Date:   Thu Mar 7 17:46:09 2024 +0100

    Adding gitignore

commit db994ba62f987e907957debeac78940ab0a4b136
Author: AbdulRahman Bani Almarjeh <aboodaysy123@gmail.com>
Date:   Thu Mar 7 16:51:04 2024 +0100

    HKU Docments commit

C:\Users\abood\Desktop\C#-progress>
```

| | |
|---|---|
| GIT SHOW | To view a commit. |
| GIT SHOW~(How many steps you want to go back.) | To view specific commit. |
| GIT LS-TREE HEAD~1 | To list the files that are in specific commit in this case head commit~1. |
| GIT RESTORE | To restore files from staging area. |
| GIT CLEAN | To remove untrack files. |
| GIT RESTORE --SOURCE ..... | To restore a file to an earlier version. |
| GIT CONFIG --GLOBAL ALIAS.LG “COMMANDCODE” | To create short cut for long command in GitHub |
| GIT DIFF HEAD~2 HEAD | To see the changes across two commits. |
| GIT CHECKOUT | To move between commits from heads to master. Heads are each commit that I did in past. Master is last commit. |
| GIT BISECT | Used to find bugs quickly. |
| GIT SHORTLOG –H | Finding Contributors. |
| GIT BLAME | To blame whoever wrote bad code ;) |

# What is branching:

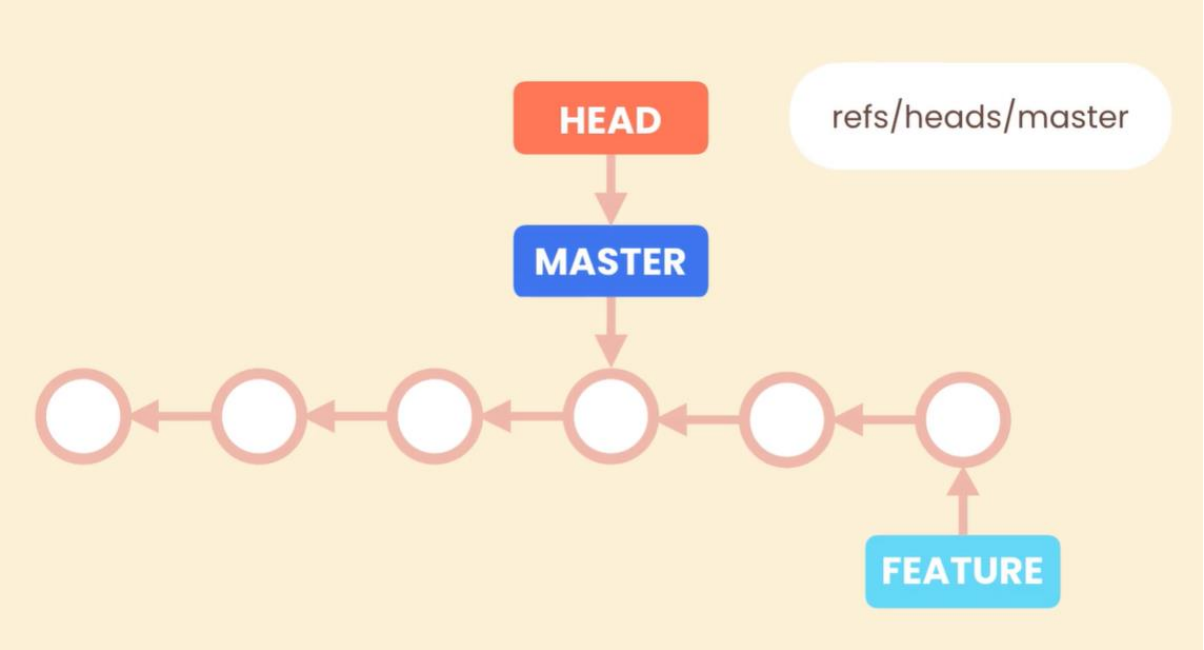
**branching allows us to diverge from the main line of work and work on something else in isolation. After finishing working on branch, you can merge the diverge with master branch whenever you want to. Here are some useful commands that can be used with branches.**



| Branch Commands | What For |
|---|---|
| Git branch | To view the branches. |
| Git branch (Name) | To create branch. |
| Git status | To see on which branch you are currently working on. |
| Git switch (Name) | To switch between branches. |
| Git branch –m (OldName) (NewName) | To change the name of the branch. |
| Git branch –d (Name) | To delete branches. |
| Git stash push –m “testing code” | Storing changes in safe place. |
| Git stash push –am | To stash another second stash. |
| Git stash apply | To apply what changes that is in stash to the repository. |
| Git stash drop (stash number) | To get rid of stash. |
| Git stash clear | To delete all stash at once. |
| Git merge --no--ff (BranchName) | To merge branch in fast-forward merges. |
| Git merge (BranchName) | To merge branch with three-waymerged. |
| Git branch --merged / Git branch --no-merged | To viewing merged and unmerged branches. |
| Git merge --abort | To aborting a merge and go back to earlier state before to the merge. |
| Git reset --hard HEAD~1 | To go back to one commit earlier. |
| Git revert –M 1 HEAD | To revert back to a previous branch. |
| Git merge --squash | To merger sub-branches without committing (commit in master branch required to fully merge) Deleting the sub-branch is recommended to avoid conflicts. |
| Git rebase master | To rebasing branch to the master branch. (Require a lot of attention before doing it). |
| Git cherry-pick(commitID) | To move some changes from branch to master. |
| Git restore --source=(BranchName) -- (FileName) | To pick up a file from another branch. |

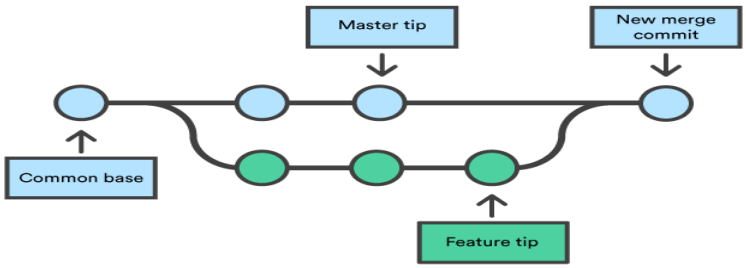
| | |
|---|---|
| Git branch –vv | To view the local and the remote branches. |

Head is the pointer of the branch. For example, we are working now in master branch

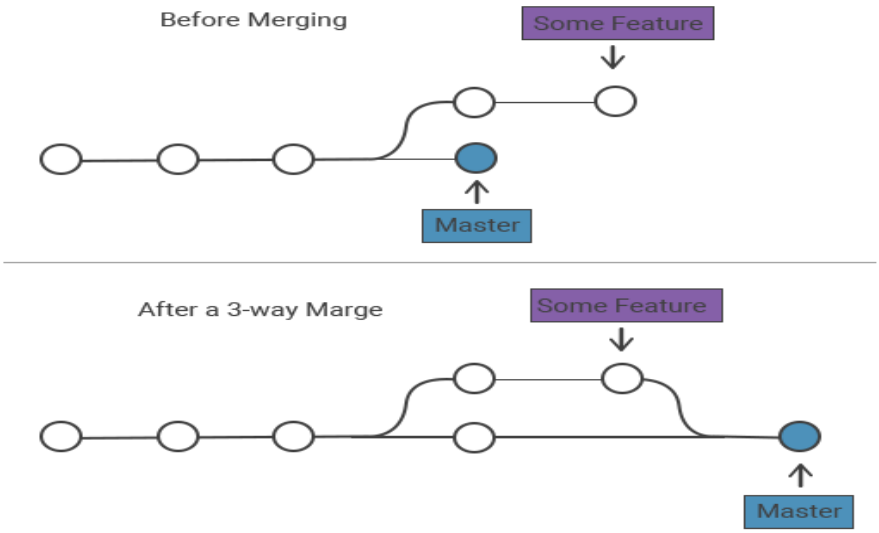



Once we are done from updating there is two ways to merge branch to the main branch fast-forward merge and rebase merge. But you can also take specific file or something that you need from branch to main it really depends on what you need. So, you can approach both options. Both options are fine. Here under you can see the difference:

FastForward-Merge




Three-way-merge


Before Merging
Some Feature
Master
After a 3-way Marge
Some Feature
Master