GIT

VERSION CONTROL SYSTEM

Introduction to GIT Version Control System

Git is one of the most common terms heard between programmers in the last four-five years.

Linus Torvalds, who started the Linux kernel, is the person who created this software to maintain and track different versions of source code among the programmers.

Different Types of Version Controller

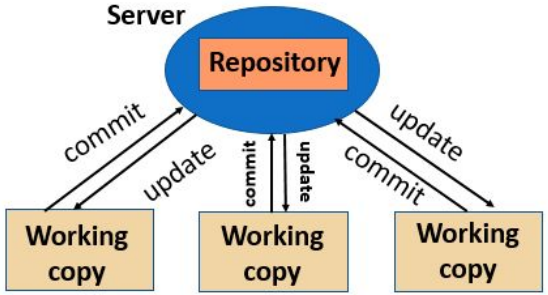
There are different types of tools available and below are some of them:

1. Subversion – Since developed by Apache, used widely by Apache vendors
2. **Git**
3. Bazaar
4. Mercurial

Basically, there are two types of version control system methodologies on which the above tools work upon. They are: **Centralized Version Control System (CVCS) Distributed Version Control System (DVCS)**

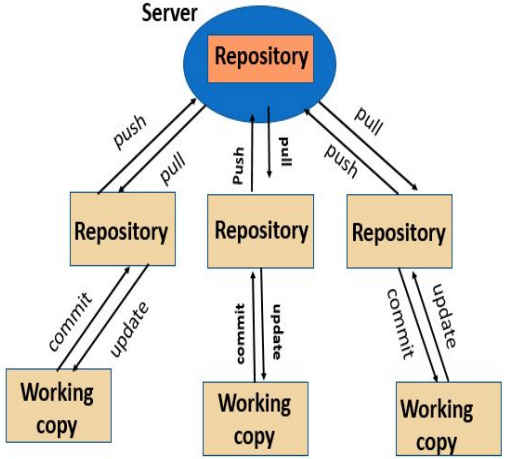
1. CVCS

Here the code written is stored in the centralized repository or in the centralized server. No working copy is available at local machines, which is a huge disadvantage when server failure occurs. I need to have an active server connection always to work on the repo. SVN uses this control system.



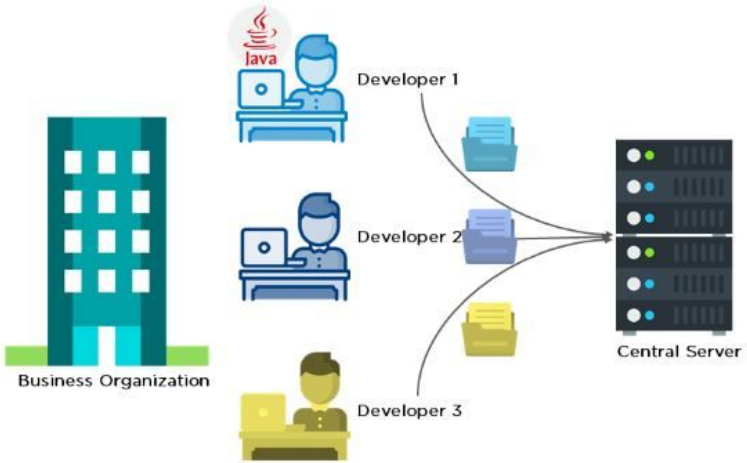
2. DVCS

We also have the source code on the server, but along with that, we have it as a local copy on working machines. So even if there is a failure at the server level, we can mirror back the local working copy to the server when it is restored. This availability of local working copy on each machine is responsible for the term ‘Distributed’ in DVCS. Git, Mercurial uses a distributed version control system.



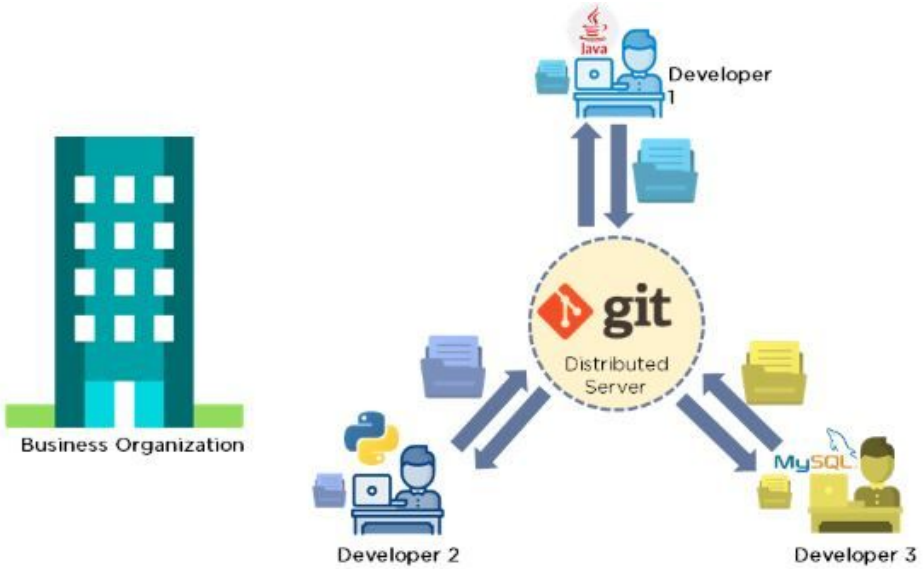
Before diving deep, let's explain a scenario before Git:

- 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



Now let's look at the 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



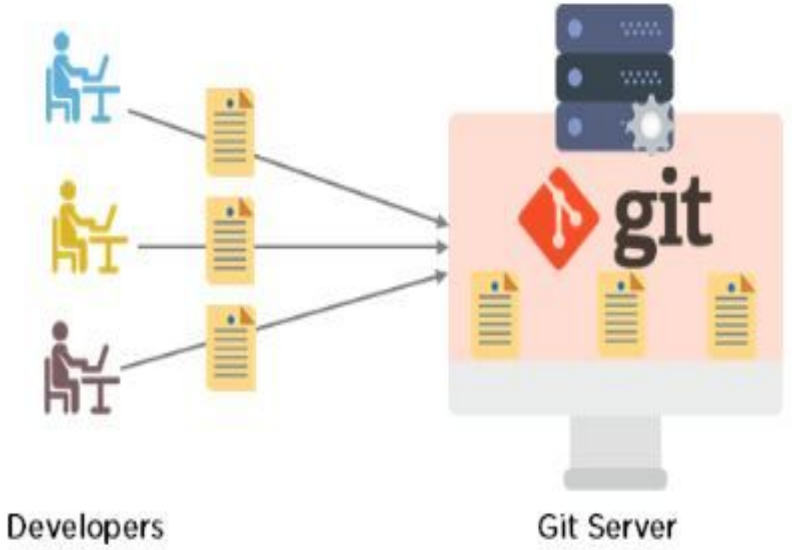
What is Git?

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

Features of Git

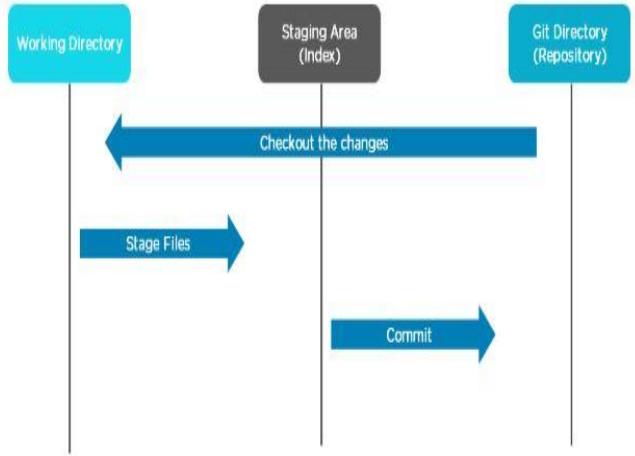
- Tracks history
- Free and open source
- Supports non-linear development
- Creates backups
- Scalable
- Supports collaboration
- Branching is easier
- Distributed development



Git Workflow

The Git workflow is divided into three states:

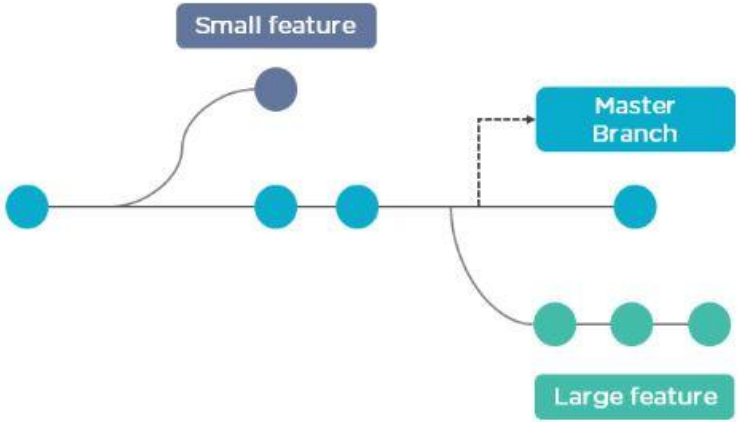
- **Working directory** - Modify files in your working directory
- **Staging area (Index)** - Stage the files and add snapshots of them to your staging area
- **Git directory (Repository)** - Perform a commit that stores the snapshots permanently to your Git directory. Checkout any existing version,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



Branch in Git

Branch in Git is used to keep your changes until they are ready. You can do your work on a branch while the main branch (master) remains stable. After you are done with your work, you can merge it with the main office.

The diagram shows there is a master branch. There are two separate branches called “small feature” and “large feature.” Once you are finished working with the two separate branches, you can merge them and create a master branch.

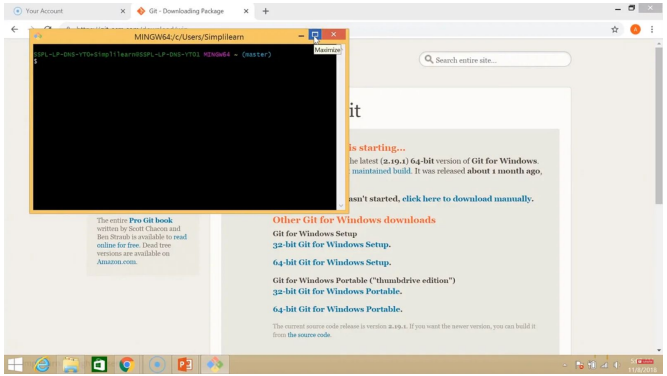


Git Installation on Windows

Let us now look at the various steps in the Git installation on Windows.

Step 1:

Download the latest version of Git and choose the 64/32 bit version. After the file is downloaded, install it in the system. Once installed, select Launch the Git Bash, then click on finish. The Git Bash is now launched.



Step 2:

Check the Git version: $ git --version

Step 3:

For any help, use the following command: $ git help config

This command will lead you to a browser of config commands. Basically, the help the command provides a manual from the help page for the command just following it (here, it's config).

Another way to use the same command is as follows: $ git config --help

Step 4:

Create a local directory using the following command:

$ mkdir test

$ cd test

Step 5:

The next step is to initialize the directory:

$ git init

Step 6:

Go to the folder where "test" is created and create a text document named "demo." Open "demo" and put any content, like "Hello CG G2 Employees." Save and close the file.

Step 7:

Enter the Git bash interface and type in the following command to check the status:

$ git status

Step 8:

Add the "demo" to the current directory using the following command:

$ git add demo.txt

Step 9:

Next, make a commit using the following command:

$ git commit -m "committing a text file"

Step 10:

Link the Git to a Github Account:

$ git config --global user.username

Step 11:

Open your Github account and create a new repository with the name "test_demo" and click on "Create repository." This is the remote repository. Next, copy the link of "test_demo."

Step 12:

Go back to Git bash and link the remote and local repository using the following command:

$ git remote add origin <link>

Step 13:

Push the local file onto the remote repository using the following command:

$ git push origin master

Step 14:

Move back to Github and click on "test_demo" and check if the local file "demo.txt" is pushed to this repository.

All The Git Commands You Need to Know About

While working on Git, we actively use two repositories.

- **Local repository:** The local repository is present on our computer and consists of all the files and folders. This Repository is used to make changes locally, review history, and commit when offline.
- **Remote repository:** The remote repository refers to the server repository that may be present anywhere. This repository is used by all the team members to exchange the changes made.

Both repositories have their own set of commands. There are separate Git Commands that work on different types of repositories.

Git Commands: Working With Local Repositories

git init

- The command git init is used to create an empty Git repository.
- After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.

```
git init
```

git add

- Add command is used after checking the status of the files, to add those files to the staging area.
- Before running the commit command, "git add" is used to add any new or modified files.

```
git add .
```

git commit

- The commit command makes sure that the changes are saved to the local repository.
- The command "git commit –m <message>" allows you to describe everyone and help them understand what has happened.

git commit -m “commit message”

git status

- The git status command tells the current state of the repository.
- The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status. Also, if there are no changes, it will show the message no changes to commit, working directory clean.

git status

git config

- The git config command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository.
- When git config is used with --global flag, it writes the settings to all repositories on the computer.

git config --global user.name “any user name”

git config --global user.email <email id>

git branch

- The git branch command is used to determine what branch the local repository is on.
- The command enables adding and deleting a branch.

 # Create a new branch

```
git branch <branch_name>
```

 # List all remote or local branches

```
git branch -a
```

 # Delete a branch

git branch -d <branch_name> - deletes only in local repository

git push origin -d <branch_name> - deletes only in remote repository

git checkout

- The git checkout command is used to switch branches, whenever the work is to be started on a different branch.
- The command works on three separate entities: files, commits, and branches.

 # Checkout an existing branch

```
git checkout <branch_name>
```

 # Checkout and create a new branch with that name

```
git checkout -b <new_branch>
```

 # Check the available branches

```
git branch -a
```

git merge

- The git merge command is used to integrate the branches together. The command combines the changes from one branch to another branch.
- It is used to merge the changes in the staging branch to the stable branch.

```
git merge <branch_name>
```

git remote

Sometimes we are trying to push some changes to the remote server, but it will show the error like "**error: failed to push some refs to 'https :< remote repository Address>**." There may be the reason that you have not set your remote branch. We can set the remote branch for the local branch. We will implement the following process to set the remote server:

To check the remote server, use the below command:

```
git remote -v
```

Upstream and Downstream

The term upstream and downstream refers to the repository. Generally, upstream is from where you clone the repository, and downstream is any project that integrates your work with other works.

- To push the changes and set the remote branch as default
  - git push --set-upstream origin master

- We can also set the default remote branch by using the git branch command.
  - git branch --set-upstream-to origin master

- To display default remote branches
  - git branch -vv

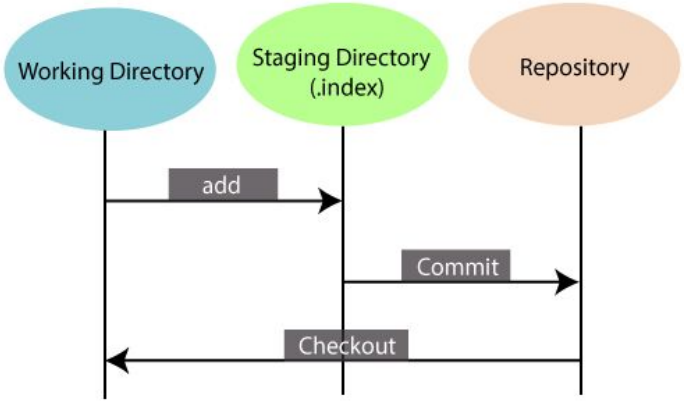
git stash

- 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. Some useful options are given below:

- **Git stash**
- **Git stash save**
- **Git stash list**
- **Git stash apply**
- **Git stash changes**
- **Git stash pop**
- **Git stash drop**
- **Git stash clear**
- **Git stash branch**

Git Index

The Git index is a staging area between the working directory and repository. It is used to build up a set of changes that you want to commit together. To better understand the Git index, then first understand the working directory and repository.



You can check what is in the index by the **git status command**. The git status command allows you to see which files are staged, modified but not yet staged, and completely untracked. Staged files mean, it is currently in the index.

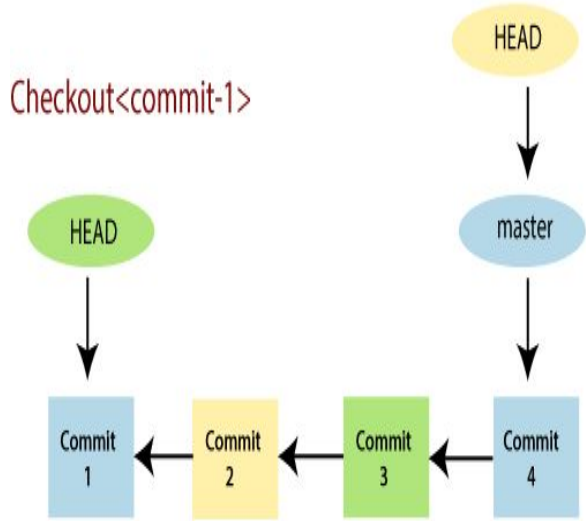
Git Head

The **HEAD** points out the last commit in the current checkout branch. It is like a pointer to any reference. The HEAD can be understood as the "**current branch**." When you switch branches with 'checkout,' the HEAD is transferred to the new branch.

Git Show Head

The **git show head** is used to check the status of the Head. This command will show the location of the Head.

```
git show HEAD
```



git tag

Tagging in GIT refers to creating specific points in the history of your repository/data. It is usually done to mark the release points.

Two main purposes of tags are:

- Make Release point on your code.
- Create historic restore points.

Create a tag - git tag {tag name} / git tag -a {tag name} -m {some message}

See all the created tags - git tag / git show {tag name}

Push tags to remote - git push origin {tag name} / git push –tags

Delete Tags (locally) - git tag -d {tag name} - (can also delete tag locally and remote at the same time) **/ git tag –delete {tag name}**

Delete Tags (remote) - git push origin -d {tag name} / git push origin –delete {tag name}

Staging & Commits

Git Init

Git Add

Git Commit

Git Clone

Git Stash

Git Ignore

Git Fork

Git Repository

Git Index

Git Head

Git Origin Master

Git Remote

Git Tags

Upstream & Downstream

Inspecting Changes

Git Log

git log

git log - - oneline (one commit per line)

git log - - stat (modified)

git log - - patch/-p (modified)

git log -3 (last 3 commit)

git log - - after=”yy-mm-dd”

git log - - after=”5 days ago”

git log - - after=”2022-11-01” - -before=”2022-11-10”

git log - - author=”Author Name”

git log - - grep=”Commit message”

Git Diff

git diff - Track the changes that have not been staged

git diff - - staged - Track the changes that have staged but not committed

git diff HEAD - Track the changes after committing a file

git diff <commit1-sha> <commit2-sha> - Track the changes between two commits

git diff <branch1> <branch2> - Git diff in branches

Git Status

git status

Git Commands: Working With Remote Repositories

git remote

- The git remote command is used to create, view, and delete connections to other repositories.
- The connections here are not like direct links into other repositories, but as bookmarks that serve as convenient names to be used as a reference.

git remote add origin <address>

git clone

- The git clone command is used to create a local working copy of an existing remote repository.
- The command downloads the remote repository to the computer. It is equivalent to the Git init command when working with a remote repository.

git clone <remote_URL>

Fork	Clone
Forking is done on the GitHub Account	Cloning is done using Git
Forking a repository creates a copy of the original repository on our GitHub account	Cloning a repository creates a copy of the original repository on our local machine
Changes made to the forked repository can be merged with the original repository via a pull request	Changes made to the cloned repository cannot be merged with the original repository unless you are the collaborator or the owner of the repository
Forking is a concept	Cloning is a process
Forking is just containing a separate copy of the repository and there is no command involved	Cloning is done through the command ‘git clone‘ and it is a process of receiving all the code files to the local machine

git pull

- The git pull command is used to fetch and merge changes from the remote repository to the local repository.
- The command "git pull origin master" copies all the files from the master branch of the remote repository to the local repository.

```
git pull <branch_name> <remote URL>
```

git push

- The command git push is used to transfer the commits or pushing the content from the local repository to the remote repository.
- The command is used after a local repository has been modified, and the modifications are to be shared with the remote team members.

```
git push -u origin master
```

Git Fetch	Git Pull
Gives the information of a new change from a remote repository without merging into the current branch	Brings the copy of all the changes from a remote repository and merges them into the current branch
Repository data is updated in the .git directory	The local repository is updated directly
Review of commits and changes can be done	Updates the changes to the local repository immediately.
No possibility of merge conflicts.	Merge conflicts are possible if the remote and the local repositories have done changes at the same place.
Command for Git fetch is git fetch<remote>	Command for Git Pull is git pull<remote><branch>
Git fetch basically imports the commits to local branches so as to keep up-to-date that what everybody is working on.	Git Pull basically brings the local branch up-to-date with the remote copy that will also updates the other remote tracking branches.

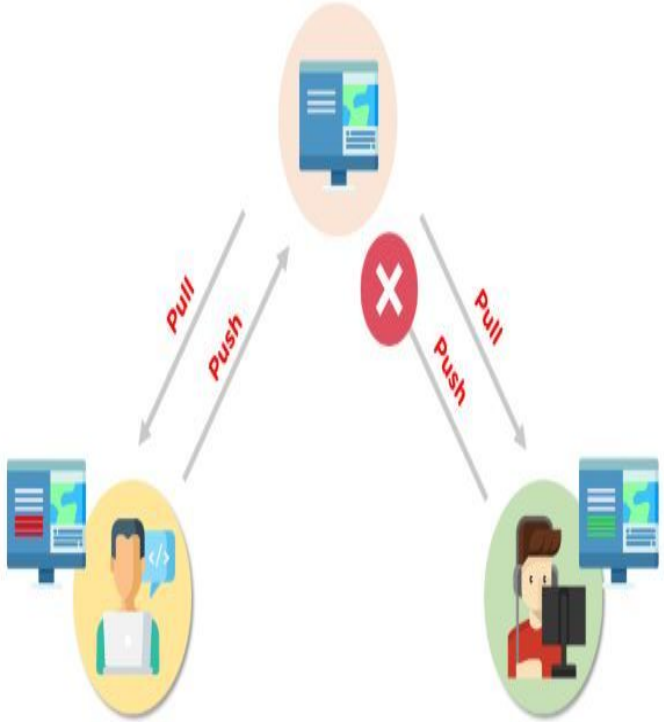
What is a Git Merge Conflict?

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.

The following is an example of how a Git merge conflict works:

Let's assume there are two developers: Developer A and Developer B. Both of them pull the same code file from the remote repository and try to make various amendments in that file. After making the changes, Developer A pushes the file back to the remote repository from his local repository. Now, when Developer B tries to push that file after making the changes from his end, he is unable to do so, as the file has already been changed in the remote repository.

To prevent such conflicts, developers work in separate isolated branches. The Git merge command combines separate branches and resolves any conflicting edits.



Types of Git Merge Conflicts

There are two points when a merge can enter a conflicted state:

1. Starting the Merge Process

If there are changes in the working directory stage area for the current project, merging won't start.

2. During the Merge Process

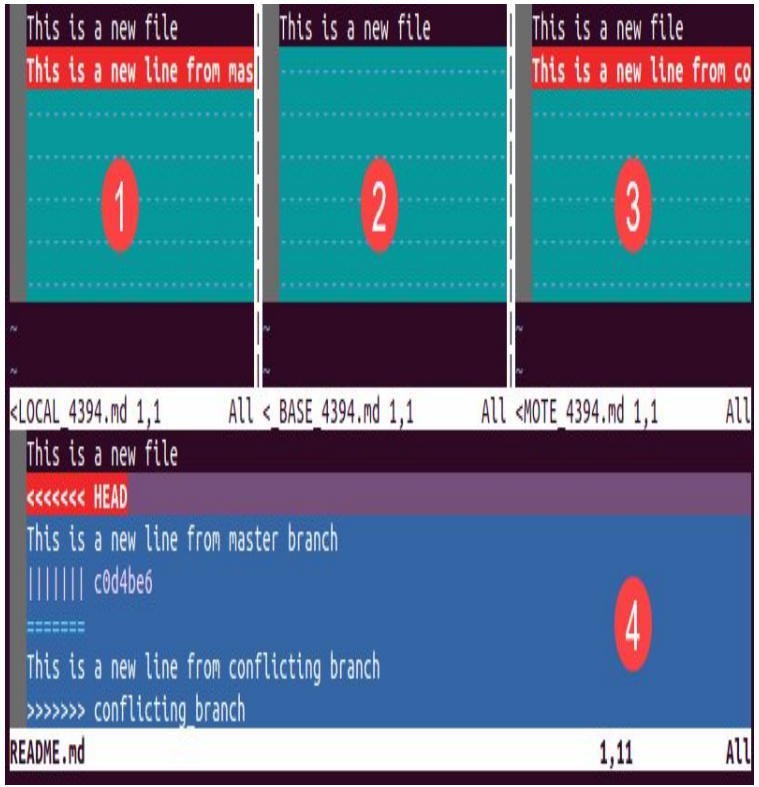
The failure during the merge process indicates that there is a conflict between the local branch and the branch being merged.

In this case, Git resolves as much as possible, but there are things that have to be resolved manually in the conflicted files.

How to Resolve Merge Conflicts in Git?

There are a few steps that could reduce the steps needed to resolve merge conflicts in Git.

1. The easiest way to resolve a conflicted file is to open it and make any necessary changes
2. After editing the file, we can use the git add a command to stage the new merged content
3. The final step is to create a new commit with the help of the git commit command
4. Git will create a new merge commit to finalize the merge



As you can see in the MERGED pane (bottom), the content where the conflict is present is wrapped around the lines **<<<<<<< HEAD and >>>>> mybranch separated by =======**
We can manually resolve the merge conflict by editing the content in the bottom pane, and then saving the file using Otherwise, if you need only content of the file from one of**:wqa (Write and Quit all files).**
the branches, and not a mixture of both, you can use the below commands:

To get LOCAL version to MERGED :**diffg LO**
To get REMOTE version to MERGED :**diffg RE**
To get BASE version to MERGED :**diffg BA**

Once the conflict resolution is successful, the merged file will be staged for commit.

Tips On How to Prevent Merge Conflicts

Merge conflicts only happen when the computer is not able to resolve the problem automatically.

Here are some tips on how to prevent merge conflicts:

- Use a new file instead of an existing one whenever possible.
- Avoid adding changes at the end of the file.
- Push and pull changes as often as possible.
- Do not beautify code or organize imports on your own.
- Avoid the solo programmer mindset by keeping in mind the other people who are working on the same code.

HANDS ON - GIT COMMANDS

Git config command

Git init command

Git clone command

Git add command

Git commit command

Git status command

Git push Command

Git pull command

Git stash command

Git tag command

Git diff command

Git branch Command

Git merge Command

Git log command

Git remote command

Git fetch command