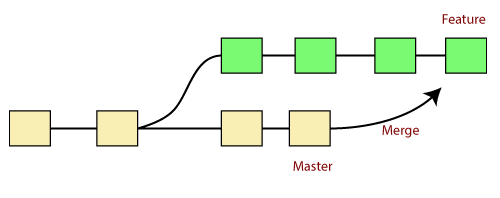
Git Merge and Merge Conflict

In Git, the merging is a procedure to connect the forked history. It joins two or more development history together. The git merge command facilitates you to take the data created by git branch and integrate them into a single branch. Git merge will associate a series of commits into one unified history. Generally, git merge is used to combine two branches.



It is used to maintain distinct lines of development; at some stage, you want to merge the changes in one branch. It is essential to understand how merging works in Git.

In the above figure, there are two branches **master** and **feature**. We can see that we made some commits in both functionality and master branch, and merge them. It works as a pointer. It will find a common base commit between branches. Once Git finds a shared base commit, it will create a new "merge commit." It combines the changes of each queued merge commit sequence.

The "git merge" command

The git merge command is used to merge the branches.

The syntax for the git merge command is as:

1. $ git merge **<query>**

It can be used in various context. Some are as follows:

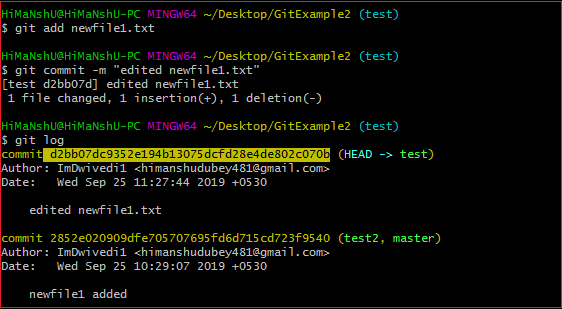
**Scenario1: To merge the specified commit to currently active branch:**

Use the below command to merge the specified commit to currently active branch.

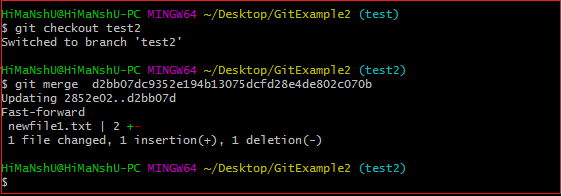
1. $ git merge **<commit>**

The above command will merge the specified commit to the currently active branch. You can also merge the specified commit to a specified branch by passing in the branch name in <commit>. Let's see how to commit to a currently active branch.

See the below example. I have made some changes in my project's file **newfile1.txt** and committed it in my **test** branch.



Copy the particular commit you want to merge on an active branch and perform the merge operation. See the below output:



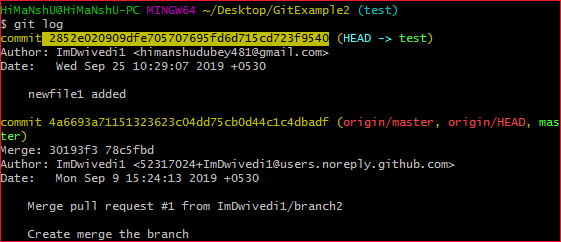
In the above output, we have merged the previous commit in the active branch test2.

**Scenario2: To merge commits into the master branch:**

To merge a specified commit into master, first discover its commit id. Use the log command to find the particular commit id.

1. $git log

See the below output:



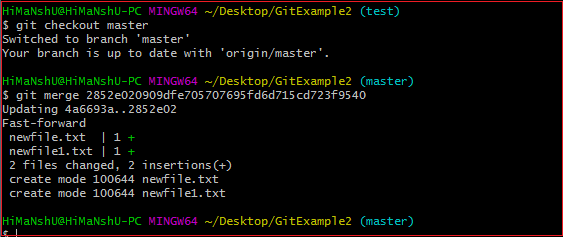
To merge the commits into the master branch, switch over to the master branch.

1. $ git checkout master

Now, Switch to branch 'master' to perform merging operation on a commit. Use the git merge command along with master branch name. The syntax for this is as follows:

1. $ git merge master

See the below output:



As shown in the above output, the commit for the commit id ***2852e020909dfe705707695fd6d715cd723f9540*** has merged into the master branch. Two files have changed in master branch. However, we have made this commit in the **test** branch. So, it is possible to merge any commit in any of the branches.

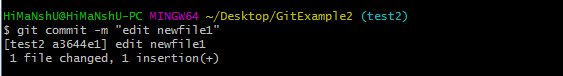
Open new files, and you will notice that the new line that we have committed to the test branch is now copied on the master branch.

**Scenario 3: Git merge branch.**

Git allows merging the whole branch in another branch. Suppose you have made many changes on a branch and want to merge all of that at a time. Git allows you to do so. See the below example:

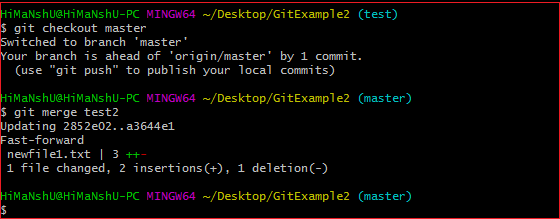
Git Merge and Merge Conflict

In the given output, I have made changes in newfile1 on the test branch. Now, I have committed this change in the test branch.



Now, switch to the desired branch you want to merge. In the given example, I have switched to the master branch. Perform the below command to merge the whole branch in the active branch.

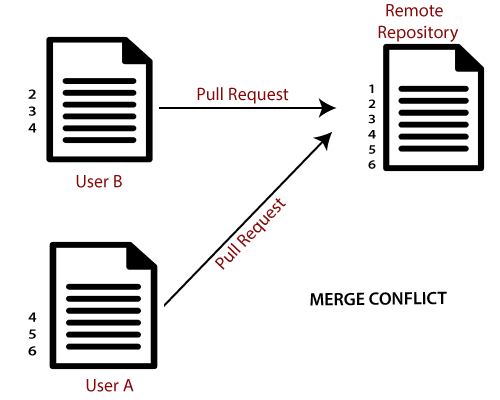
1. $ git merge **<branchname>**



As you can see from the given output, the whole commits of branch test2 have merged to branch master.

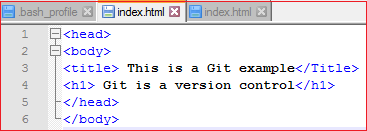
Git Merge Conflict

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.



Let's understand it by an example.

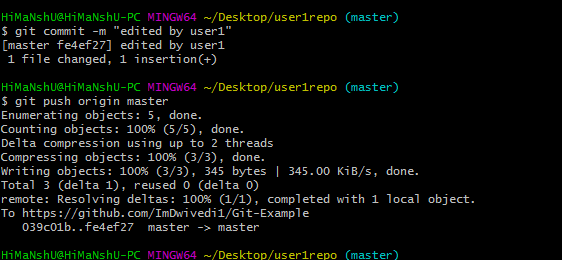
Suppose my remote repository has cloned by two of my team member **user1** and **user2**. The user1 made changes as below in my projects index file.



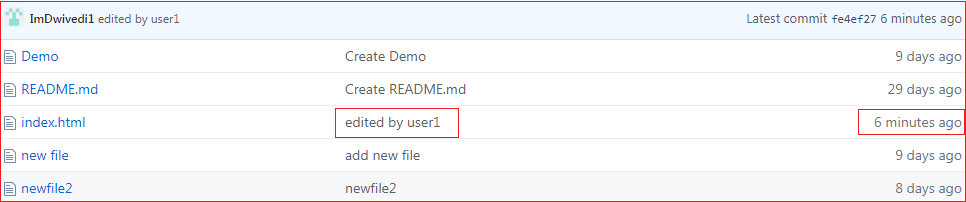
Update it in the local repository with the help of git add command.

Git Merge and Merge Conflict

Now commit the changes and update it with the remote repository. See the below output:

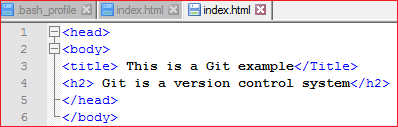


Now, my remote repository will look like this:

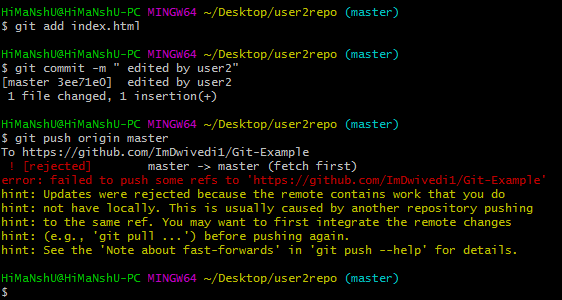


It will show the status of the file like edited by whom and when.

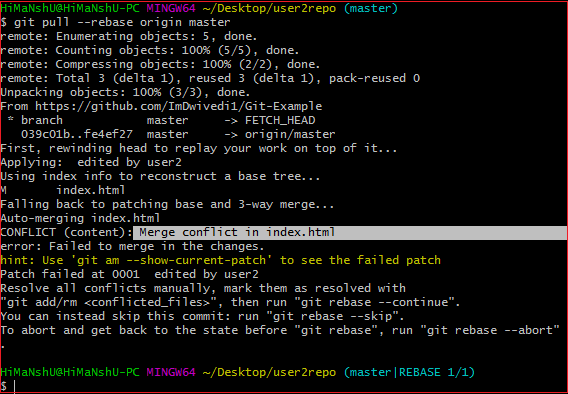
Now, at the same time, **user2** also update the index file as follows.



User2 has added and committed the changes in the local repository. But when he tries to push it to remote server, it will throw errors. See the below output:



In the above output, the server knows that the file is already updated and not merged with other branches. So, the push request was rejected by the remote server. It will throw an error message like **[rejected] failed to push some refs to <remote URL>**. It will suggest you to pull the repository first before the push. See the below command:



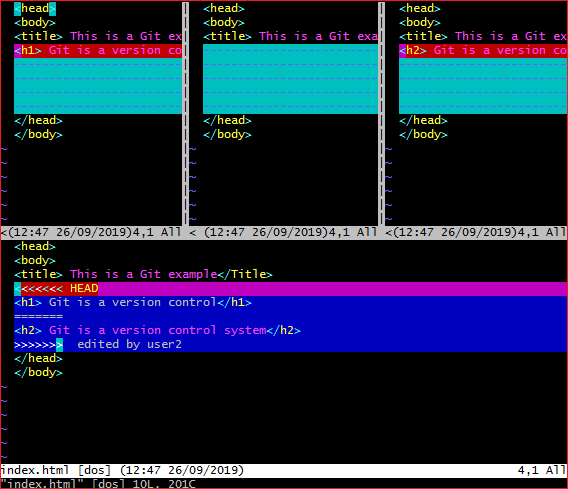
In the given output, git rebase command is used to pull the repository from the remote URL. Here, it will show the error message like **merge conflict in <filename>**.

Resolve Conflict:

To resolve the conflict, it is necessary to know whether the conflict occurs and why it occurs. Git merge tool command is used to resolve the conflict. The merge command is used as follows:

1. $ git mergetool

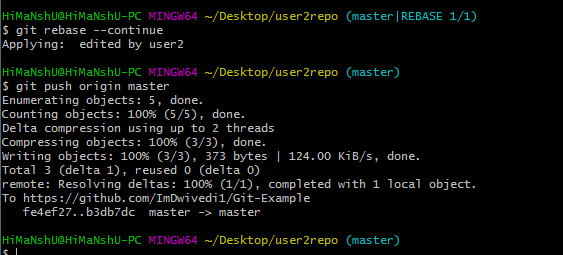
In my repository, it will result in:



The above output shows the status of the conflicted file. To resolve the conflict, enter in the insert mode by merely pressing **I key** and make changes as you want. Press the **Esc key**, to come out from insert mode. Type the: **w!** at the bottom of the editor to save and exit the changes. To accept the changes, use the rebase command. It will be used as follows:

1. $ git rebase --continue

Hence, the conflict has resolved. See the below output:



In the above output, the conflict has resolved, and the local repository is synchronized with a remote repository.

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**. In the above example, user1 wrote "<h1> Git is a version control</h1>" in the base or HEAD branch and user2 wrote "<h2> Git is a version control</h2>".

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.