**Git Fast-forward merge vs three-way merge**

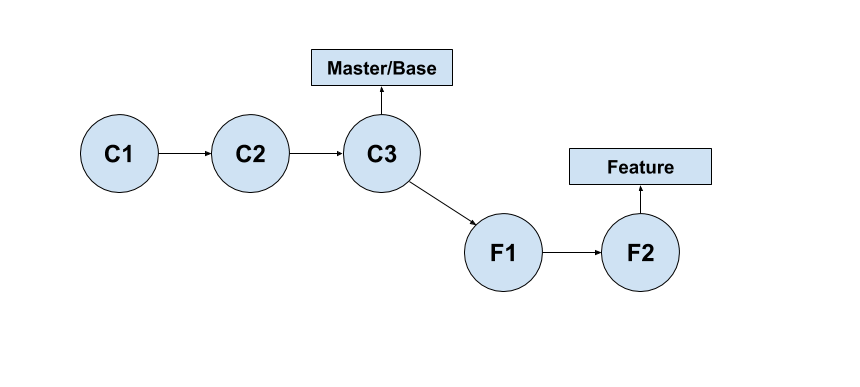
Imagine you are working on master branch and had to create a new feature branch for a user story you need to work on. This is so simple, you’ll have to just run the below command:

git checkout -b <new-feature-branch>

This will create and switch to the branch at the same time along with your working changes. This is tracked by Git. It has to be aware of the commit or the snapshot at which a new branch is created to do a merge. This becomes the base/ancestor when doing a three-way merge, which we will discuss in a while. It’s not always the case that the base becomes the commit at which a new branch is created.

**Two-way merge:**

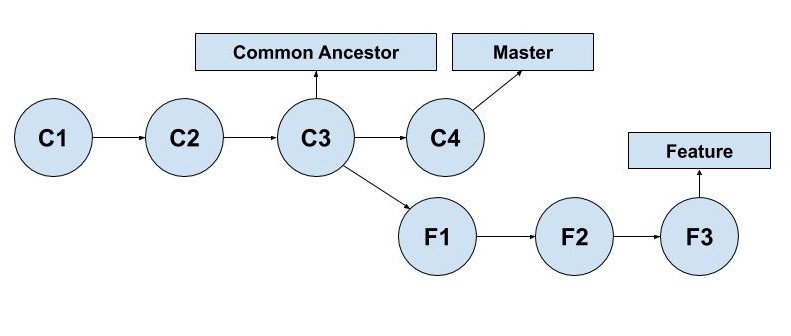
Two-way merge is a simple case where merging involves only two snapshot. Let’s clear it up with an example. Let’s say that you wish to merge your feature branch with master. Assume that the master branch has no more commits from the time you created a new branch.



When the feature branch has to be merged with the master branch, the two recent commits on either of these branches C3 and F2 are compared and merged automatically unless there are no conflicts, for which manual resolution is required. This is also called a fast-forward merge as the master branch is behind two commits before merge and a merge moves it forward.

**Three-way merge:**

A three-way merge involves three snapshots. Two are the ones that are involved in a two-way merge, and the third one is the base file or the common ancestor with which these two files will be compared.



As you can see, C3 is the common ancestor with which C4 and F3 will be compared for merging. One may ask why this has to be like this? Why can’t F3 and C4 be compared and merged?