# **Rebasing:**

# There are two main ways to integrate changes from one branch into another.

# Merge

# Rebase.

# 1. The easiest way to integrate the branches is the merge command. It performs a three-way merge between the two latest branch snapshots and the most recent common ancestor of the two, creating a new snapshot and commit.

# 2. With the rebase command, you can take all the changes that were committed on one branch and replay them on a different branch.

# For this example, you would check out the experiment branch, and then rebase it onto the master branch as follows:

# $ git checkout experiment

# $ git rebase master

# First, rewinding head to replay your work on top of it.

# Applying: added staged command

# How rebasing works:

# This operation works by going to the common ancestor of the two branches (the one you are on and the one you are rebasing onto), getting the diff introduced by each commit of the branch you are on, saving those diffs to temporary files, resetting the current branch to the same commit as the branch you are rebasing onto, and finally applying each change in turn.

# **At this point, you can go back to the master branch and do a fast-forward merge.**

# $ git checkout master

# $ git merge experiment

# **Question: What is the need of fast-forward merge at this stage when rebase works.**

# There is no difference in the end product of the integration, but rebasing makes for a cleaner history. If you examine the log of a rebased branch, it looks like a linear history: it appears like all the work happened in series, even when it originally happened in parallel.

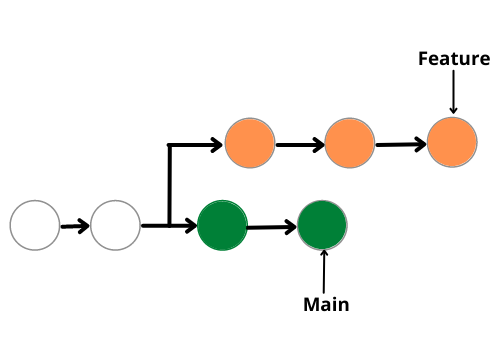
# This is often done to make sure your commits apply cleanly on a remote branch-- perhaps in a project to which you are trying to contribute but that you don't maintain. In this case, you'd do your work in a branch and then rebase your work onto origin/master when you were ready to submit your patches to the main project. That way, the maintainer doesn't have to do any integration work -- just a fast-forward or a clean apply.

# In merge and rebase the snapshot pointed to by the final commit will be same. It is only the history that is different. Rebasing replays changes from one line of work onto another in the order they wre introduced, whereas merging takes the endpoints and merges them together.

# Difference Between Merging and Rebasing

Both Git rebase and git merge perform the same task, merge the feature branch to the working branch, but in a very different way. In this article, we will discuss both of them and their benefits and difference, and their typical workflow.

### Where it is used?

Consider a situation where you are working on a feature in your branch while your friend is working on the other, and both of you need to merge changes into the main repo. In this condition, you will need to go with either of the Git rebase or git merge features, but which suits you perfectly. Let’s have a deeper look to analyze it. The pictorial view for the above condition could be as below:

### Git merge

The easiest option to merge the branches is using the git merge command. Git merge safeguards the histories of both the repositories. You can use the following merge command to merge your branch. Move to your main directory and then write these commands:

git checkout feature

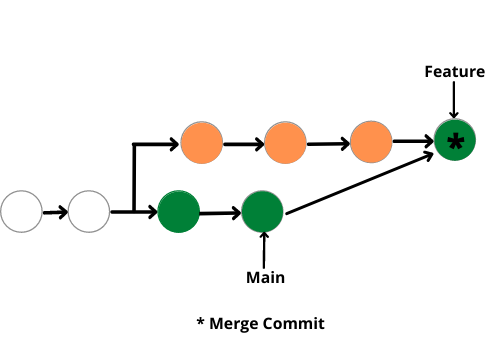
git merge main

Or, you can write

git merge feature main

**How does it work?**

It basically, creates a new “feature commit”, safeguarding the history of both the branches and giving it a structure like this:-



**Advantage**

Merge command is a non-destructive command. In this command, the existing branches are not changed in any way and thus it covers all the pitfalls or demerits of Git Rebase.

**Disadvantage**

Besides being easy, there are many few demerits also git merge also. Consider a condition where you have to work regularly on the main branch and have a very active main branch, in such conditions the commit history will become messy, thus resulting in a painful experience for the developers to understand the log or the commit history.

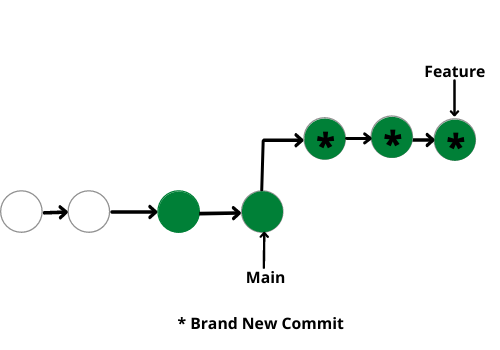
### Git Rebase

The alternative to git merge is the git rebase option. In this, we rebase the entire feature branch to merge it with the main branch. Follow the following commands to perform merge commit:-

git rebase main

**How does it work?**

Git rebase actually rebases the feature branch and merges it with the main branch. In simple words, it moves the entire feature branch to the tip of the main branch. The pictorial representation looks a bit like this:-



**Advantage**

The major benefit of using git rebase is it provides a cleaner merge history. It works linearly, removing the unnecessary merge commits, unlike git merge. It makes it easier to move along the log history and understand the changes.

**Disadvantage**

The biggest pitfall with git rebase is it you can’t see when the upstream changes were made, and they were incorporated into the feature branch. If many developers are working on this and you rebase it, then git might consider that the upstream changes were made is different than that they were previously working on.

### Git – Difference Between Merging and Rebasing

| **Git Merge** | **Git Rebase** |
| --- | --- |
| Git Merge merges two branches to create a “feature” branch. | Git Rebase rebases the feature branch to add the feature branch to the main branch. |
| Git Merge is comparatively easy. | Git Rebase is comparatively harder. |
| Git Merge safeguards history. | Git Rabse doesn’t safeguard history. |
| Git Merge is more suitable for projects with the less active main branch. | Git Rebase is suitable for projects with frequently active main branches. |
| Git Merge forms a chain-like structure. | Git Rebase forms a linear structure. |
| Git Merge is preferable for large no. of people working on a project. | Git Rebase is preferable for small groups of people. |
| Single line command is:  *git merge feature main* | Single line command is:  *git rebase main* |