Git

Merge & Rebase

Git Merge & Rebase

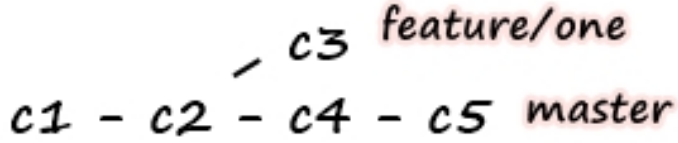
Both git merge and rebase, kind of perform the similar task of combining/joining the development histories together.

Now, what does that mean?
Simple, in git we have concept of branches, we usually have a default (master or main) branch, and then as part of our regular development/fix work we create feature/release branches out of it. Once we are done making our changes in our feature branches we want it to be published to remote branches and also combined with the main or master branch to keep it unified and accessible to others with their changes.

Hence, we need some feature to join these changes/history from different branches.

Both Merge & Rebase let you do that. The end goal is to combine the changes but how it is done is what makes Merge & Rebase different.

Let's understand them with a simple example -



You basically created a feature branch 'feature/one' from master when they were both at commit 'c2' and then you added a new commit 'c3' to the feature and also added 2 new commits 'c4' and 'c5' to master branch.

So, at the moment feature is unaware of the commits c4 & c5 which are on the master and also the master branch has no clue of commit c3 which is on feature.

Ideally that's fine as both are different branches, but if there is a need to combine them into one destination, then it is needed to know about all these commits.

Git Merge & Rebase

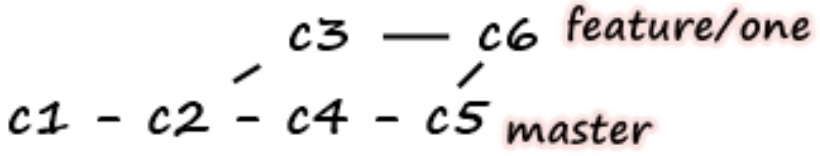
Now, suppose we want to combine the changes from the master branch to feature/one branch.

**Use the image in last slide as starting point for each of the below operation.

1. **Let's perform Merge –**

   git checkout feature/one

   git merge master



You are at your feature branch and performed 'git merge master' to merge the changes of master branch into your feature branch. So, in order to merge the changes a new merge commit 'c6' got created which binds histories from both the branches on feature branch.

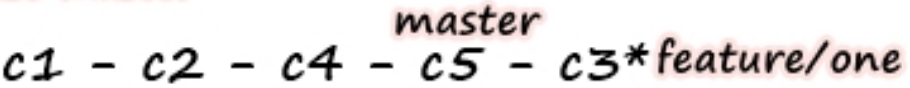
So if you now do a 'git log' you will see a chronological order of commits.

For e.g -> c1 - c2 - c3 - c4 - c5 - c6 (assuming c3 commit was done prior to c4 wrt time)

2. **Let's perform Rebase –**

   git checkout feature/one

   git rebase master



Rebase, as its name suggests, it actually resets the base. So earlier 'c2' was common between feature and master branch, we can call it the base.

Now, when we fire rebase command, it will actually pick all commits in feature after 'c2' which is just 'c3' commit. First it will apply all commits from master into the feature and on top it it applies the 'c3' commit. Why c3* & not c3?

Will explain it.

Git Merge with example - Part1

- We are on the master branch, and it has 2 commits.

```
\gitdemo Let us learn/g/git/merge_and__rebase (master)
$ git log --oneline
b161997 (HEAD -> master, origin/master) Message 2
a7cf16b Message 1
```

- Now, let's create a feature branch 'feature/test_merge' from master branch and switch to it. The commits are same as master branch.

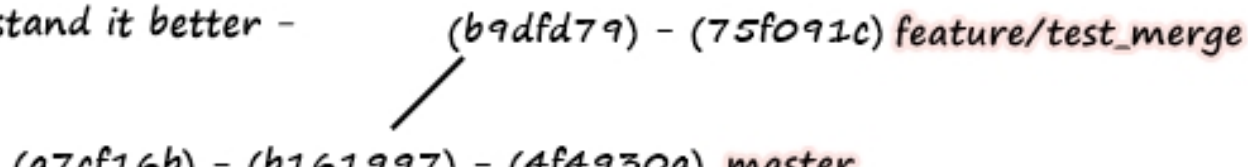
```
\gitdemo Let us learn/g/git/merge_and__rebase (feature/test_merge)
$ git log --oneline
b161997 (HEAD -> feature/test_merge, origin/master, origin/feature/test_merge, master) Message 2
a7cf16b Message 1
```

- Let's add 2 new commits to the feature branch and 1 new commit to the master.

```
\gitdemo Let us learn/g/git/merge_and__rebase (feature/test_merge)
$ git log --oneline
75f091c (HEAD -> feature/test_merge, origin/feature/test_merge) Merge Branch Message 2
b9dfd79 Merge Branch Message 1
b161997 (origin/master, master) Message 2
a7cf16b Message 1
```

```
\gitdemo Let us learn/g/git/merge_and__rebase (master)
$ git log --oneline
4f4930a (origin/master, master) Message 3
b161997 (HEAD -> master, origin/master) Message 2
a7cf16b Message 1
```

Let's draw an image which depicts the commits at the moment to understand it better -



- It's time to do the Merge.

```
\gitdemo Let us learn/g/git/merge_and__rebase (master)
$ git merge feature/test_merge
Updating 4f4930a..59d43fe
Fast-forward
 file2.txt | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 file2.txt
```

We are on the master branch and we merged feature into master.

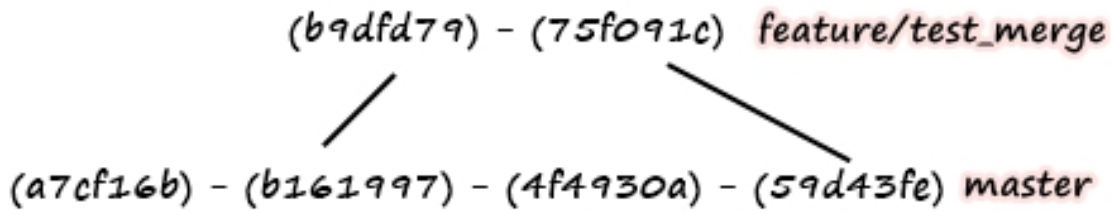
Git Merge with example - Part2

- As we merged the feature branch into the master. Master should now contain all the changes from feature. Btw, when you ran the merge command a text editor was opened for you to give the merge commit message.

```
Merge branch 'master' into feature/test_merge
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
~
```

- After the merge, history looks like -

```
\gitdemo Let us learn/g/git/merge_and__rebase (master)
$ git log --oneline
59d43fe (HEAD -> master, origin/feature/test_merge, feature/test_merge) Merge branch 'master' into feature/test_merge
4f4930a (origin/master) Message 3
75f091c Merge Branch Message 2
b9dfd79 Merge Branch Message 1
b161997 Message 2
a7cf16b Message 1
```



- So, this is how the merge operation is performed.
You can clearly notice one important thing here, that in the history every commit hash is consistent. In other words no change in the history took place, but the only change is addition of a new merge commit (59d43fe in the above scenario)

- I am sure you can imagine the simplicity in the overall operation while performing the merge. Looks like just a game of pointers.

Git Rebase with example - Part1

- Let's see how Rebase works and how it is different from Merge.

Again, our objective is same. We will create a feature out of master. We will be adding new changes/commits to both feature and master. And then we will do a rebase this time to combine changes from feature to the master branch.

```
\gitdemo Let us learn/g/git/merge__and_rebase (master)
$ git log --oneline
ae8deb9 (HEAD -> master, origin/master) Message 2
728f3f8 Message 1
```

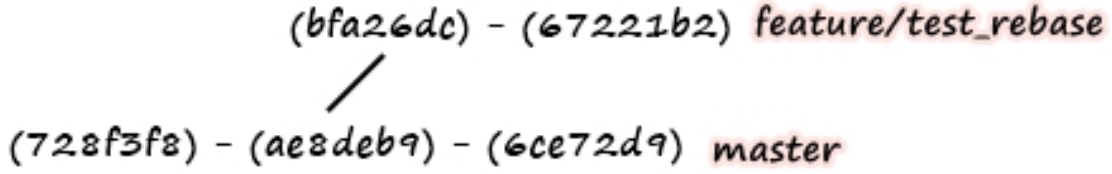
Both 'master' and 'feature/test_rebase' are in-sync at the moment.

```
\gitdemo Let us learn/g/git/merge__and_rebase (feature/test_rebase)
$ git log --oneline
ae8deb9 (HEAD -> feature/test_rebase, origin/master, origin/feature/test_rebase, master) Message 2
728f3f8 Message 1
```

- Now, let's add 2 new commits in the feature branch and 1 new commit in the master branch.

```
\gitdemo Let us learn/g/git/merge__and_rebase (feature/test_rebase)
$ git log --oneline
67221b2 (HEAD -> feature/test_rebase, origin/feature/test_rebase) Rebase branch Message 2
bfa26dc Rebase branch Message 1
ae8deb9 (origin/master, master) Message 2
728f3f8 Message 1
```

```
\gitdemo Let us learn/g/git/merge__and_rebase (master)
$ git log --oneline
6ce72d9 (HEAD -> master) Message 3
ae8deb9 (origin/master) Message 2
728f3f8 Message 1
```



Git Rebase with example - Part2

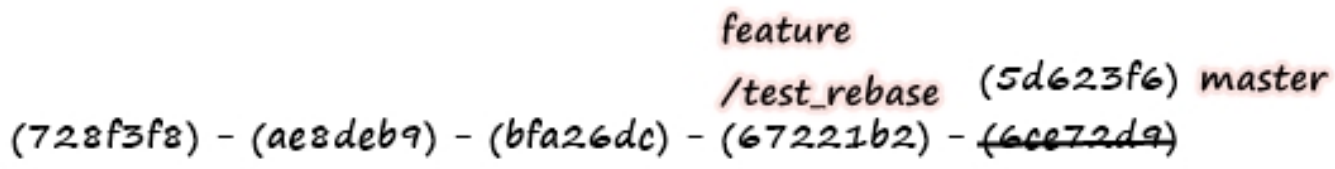
- It's time do perform Rebase.

```
\gitdemo Let us learn/g/git/merge__and_rebase (master)
$ git rebase feature/test_rebase
First, rewinding head to replay your work on top of it...
Applying: Message 3
```

- What exactly happened here?

1. The base has been reset, from the last base to a new base.
Earlier it was (ae8deb9) and now has been reset to (67221b2)
2. The new commits from master (which were not in feature) are applied on top.
Message 3 in this case.

```
\gitdemo Let us learn/g/git/merge__and_rebase (master)
$ git log --oneline
5d623f6 (HEAD -> master) Message 3
67221b2 (origin/feature/test_rebase, feature/test_rebase) Rebase branch Message 2
bfa26dc Rebase branch Message 1
ae8deb9 (origin/master) Message 2
728f3f8 Message 1
```



- Rebase operation is performed on the master branch. You can clearly see what happened. Firstly all the commits of the 'feature/test_rebase' are applied on the last common commit between master and feature (ae8deb9). And then on top of it the new commit from master is applied (Message 3). But wait, the commit hash looks different. Earlier it was 6ce72d9 and now it is 5d623f6. Why?

To Summarize -

It looks like both Rebase and Merge did there job well. And yes that's true.
In both the cases we got the feature branch changes combined into master branch.

But, one thing which is still not clear, why the commit hash changed in case of Rebase operation.
Yes, that's how Rebase is done. When you did the merge, no change was done wrt history (no commit hash changed), just a new merge commit was added.
But in case of Rebase, it actually rewrites the commit history. So, it is not basically a lift and shift of pointers but also rewiting a new commit of same changes.

We first get the new base and then on top of it we write again the changes as new commits.

And that's what makes Rebase a little risky and complicated operation if not performed carefully as it is rewriting the history.

It is always suggested to avoid doing Rebase in a branch/repo being used publicly or shared with other developers as it could create a mess. Use Merge in such scenarios.

You can use Rebase when your changes are on your feature and don't affect others. Like performing squash of commits etc.

Git pull vs Git pull --rebase

git pull or git pull --merge

Performs : git fetch + git merge

This is the default case.

In this case, your local changes are merged with the remote changes. A new merge commit is created pointing to the latest local commit and the latest remote commit.

git pull --rebase

Performs : git fetch + git rebase

In this case, your local changes are reapplied on top of the remote changes. So, the new changes on your local are picked, first the remote changes are applied which gives a new base and then your changes that were picked are applied on top as new commits.

Bonus command :P

git checkout -

To switch to the last branch you were at.

This command can be used to switch to and fro between the branches.

Resolving Conflicts

To generate a merge conflict - We created a text file 'conflict.txt' on both master and feature branch. Added 2 different lines in file on each branch. Now we are trying to merge master changes into the feature branch.

```
\gitdemo Let us learn/g/git/merge__and_rebase (feature/test_rebase)
$ git merge master
CONFLICT (add/add): Merge conflict in conflict.txt
Auto-merging conflict.txt
Automatic merge failed; fix conflicts and then commit the result.

\gitdemo Let us learn/g/git/merge__and_rebase (feature/test_rebase|MERGING)
$ cat conflict.txt
<<<<<<< HEAD
conflict message xyzzzz
conflict message abcdd
=======
conflict message 123
conflict message 1234444
>>>>>>> master
```

The moment we did 'git merge master' on feature, it gave us merge failed error and is expecting us to resolve the conflicts manually.
We checked the content of conflict.txt file.
The content between "<<<<<<<<< HEAD" & "=======" is from our current branch which is feature here. And the stuff between "======" & ">>>>>>> master" is from the branch which we are merging which is master here.

Suppose we want to keep the change from feature branch, hence we will remove what's coming from the master and add this file again.

```
\gitdemo Let us learn/g/git/merge__and_rebase (feature/test_rebase|MERGING)
$ cat conflict.txt
conflict message xyzzzz
conflict message abcdd

\gitdemo Let us learn/g/git/merge__and_rebase (feature/test_rebase|MERGING)
$ git add conflict.txt

\gitdemo Let us learn/g/git/merge__and_rebase (feature/test_rebase|MERGING)
$ git merge --continue
[feature/test_rebase 06d9ae0] Merge branch 'master' into feature/test_rebase

\gitdemo Let us learn/g/git/merge__and_rebase (feature/test_rebase)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
```

git merge --continue

To continue the merge operation.

git merge --abort

To abort the merge process.

It will try to reconstruct the pre-merge state.

Thanks!!!