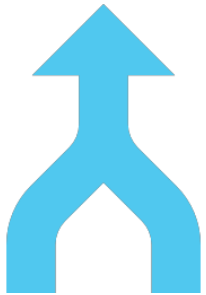
Merging Branches

Once the developer has finished his code/feature on his branch, the code will have to be combined with the master branch. This can be done using two ways:



Merging Branches – git merge

Git Merge

Git Rebase

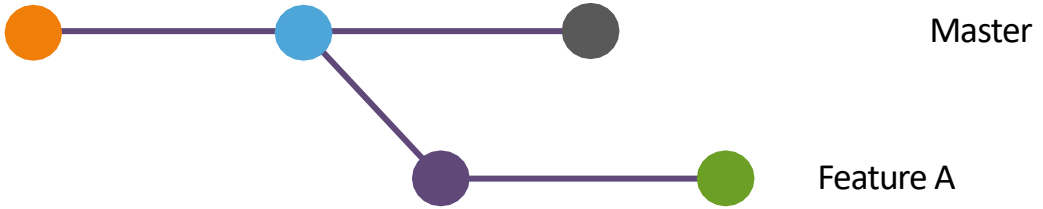
- If you want to apply changes from one branch to another branch, one can use merge command
- Should be used on remote branches, since history does not change
- Creates a new commit, which is a merger of the two branches
- Syntax: git merge <source-branch>

Merging Branches – git merge

Git Merge

Git Rebase

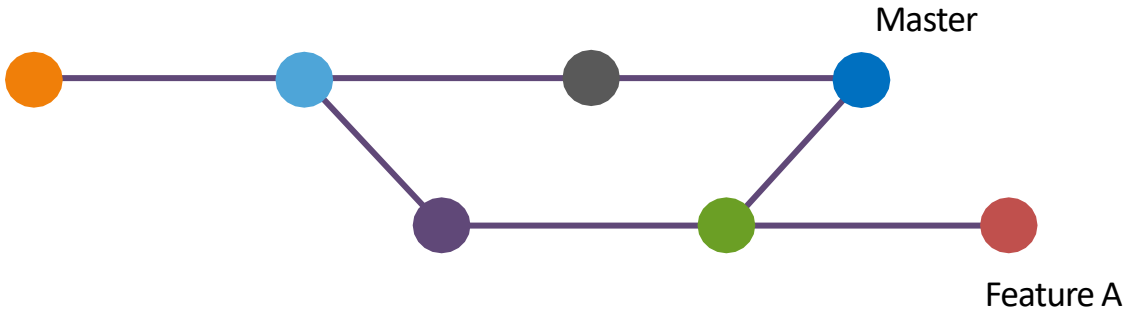
Imagine, you have a Master branch and a Feature A branch. The developer has finished his/her work in the feature A branch and wants to merge his work in the master.



Merging Branches – git merge

If he is using **git merge**, a new commit will be created, which will have the changes of Feature A and Master branch combined.

Any new commits to the Feature branch will be isolated from the master branch



Merging Branches – git merge

This command can be executed using the syntax

git merge <source-branch-name>

```
ubuntu@ip-172-31-33-5:~/devops$ ls
1.txt  2.txt  3.txt
ubuntu@ip-172-31-33-5:~/devops$ git status
On branch branch1
nothing to commit, working tree clean
ubuntu@ip-172-31-33-5:~/devops$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
ubuntu@ip-172-31-33-5:~/devops$ ls
1.txt  2.txt
ubuntu@ip-172-31-33-5:~/devops$ git merge branch1
Updating 6f13532..dd6974e
Fast-forward
 3.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 3.txt
ubuntu@ip-172-31-33-5:~/devops$ ls
1.txt  2.txt  3.txt
ubuntu@ip-172-31-33-5:~/devops$
```

Merging Branches – git merge

The history of the branch will look something like this, if we are using **git merge**

```
ubuntu@ip-172-31-26-120:~/n$ git log --graph --pretty=oneline
*   d92f22eeb6bb7fefd1706b397abe804dc557ec88 (HEAD -> master) Merge branch
'
|\
| * aebc77927892bd1c74ffd9b3d9af7f3b763ee8da (test) 1st on test
* | b62c11b6a12e4c0431bf4ae7f9fe90f744d485b7 second on master
|/
* 071f9bd946e502d4643d2fc7e2dd7c26dea0eaf9 first commit in master
```

Merging Branches – git merge

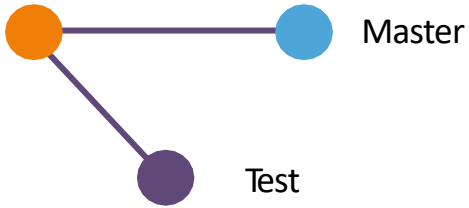
Git Merge

Git Rebase

- This is an alternative to git merge command
- Should be used on local branches, since history does change and will be confusing for other team members
- Does not create any new commit, and results in a cleaner history
- The history is based on common commit of the two branches (base)
- The destination's branch commit is pulled from it's "base" and "rebased" on to the latest commit on the source branch

Merging Branches – git rebase

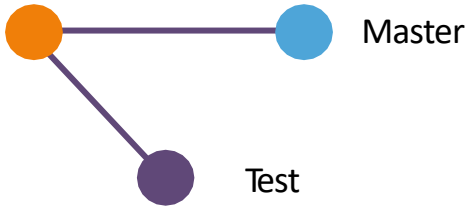
- Imagine, you have a Master branch and a test branch(local branch)
- But the master moved forward, while the code was being developed
- The developer has finished his/her work in the test branch
- Code being developed is related to the new commit added in master



Merging Branches – git rebase

- Therefore you want all the changes from master in feature.
- Since, it is a local branch, you would want a cleaner or linear history, you decide to use git rebase

Syntax: **git rebase <source branch>**



Merging Branches – git rebase

Git Merge

Git Rebase

This is how the output lookslike:

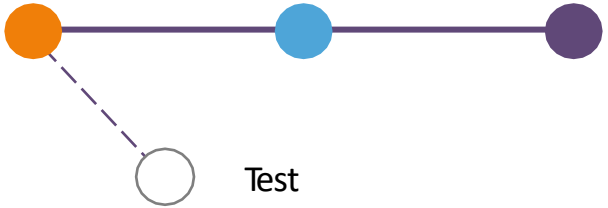
```
ubuntu@ip-172-31-26-120:~/n$ git checkout test
Switched to branch 'test'
ubuntu@ip-172-31-26-120:~/n$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: 1st in test
```

Merging Branches – git rebase

Git Merge

Git Rebase

This is how the commits look like, after a rebase. The commit was “rebased” from the first commit to the next commit



And looking at the history we can clearly see, it’s a clean linear history, without any branches

```
ubuntu@ip-172-31-26-120:~/n$ git log --graph --pretty=oneline
* 3885b20a7f8880acf4b7a785a638e95d1759dcf2 (HEAD -> test) 1st in test
* cce38fa142699171d08b08b27ed44f49052ac134 (master) 2nd in master
* 7d77f726ad1d0b64f6f20c2587560dc18123082d 1st in master
```