**Git Rebase vs Git Merge: Which is Better?**

There’s a lot of debate about git merge and git rebase about which is better. Today in the blog on Git Rebase vs Merge we will clear all your doubts about Git Rebase and Git Merge. Both the techniques are used for the same purpose, so it is a bit tricky to understand them because of their similarities.  By the end of the blog, you will get to know when to use Git Rebase vs Merge.

Git Merge and Git Rebase commands are used to combine the work of multiple developers in one code. The end objective for both these commands is same, but their usage varies.

So, the following are the topics covered in this blog:

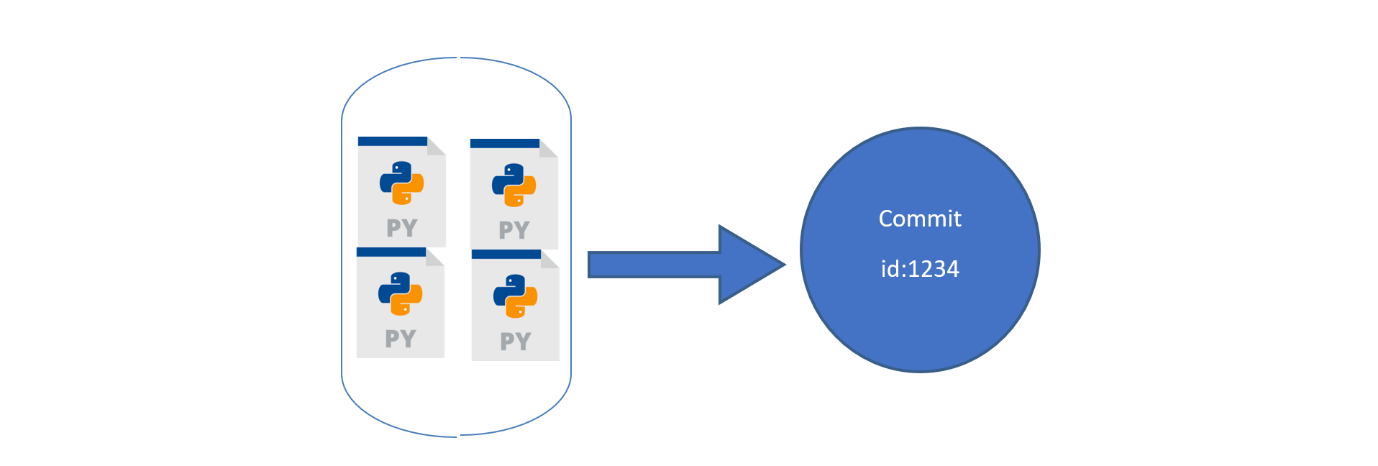
* [How does Git Work?](https://www.edureka.co/blog/git-rebase-vs-merge/#Working-of-Git)
  + [What is a Commit?](https://www.edureka.co/blog/git-rebase-vs-merge/#What-is-a-Commit?)
  + [What is a Branch?](https://www.edureka.co/blog/git-rebase-vs-merge/#What-is-a-branch?)
* [What is Merging?](https://www.edureka.co/blog/git-rebase-vs-merge/#What-is-Merging?)
* [Git Merge](https://www.edureka.co/blog/git-rebase-vs-merge/#Git-Merge?)
* [Git Rebase](https://www.edureka.co/blog/git-rebase-vs-merge/#Git-Rebase?)
* [Git Merge vs Git Rebase.](https://www.edureka.co/blog/git-rebase-vs-merge/#Git-Merge-vs-Git-Rebase.)
* [How can Git Rebase and Git Merge be used together?](https://www.edureka.co/blog/git-rebase-vs-merge/#How-can-Git)

**How does Git Work?**

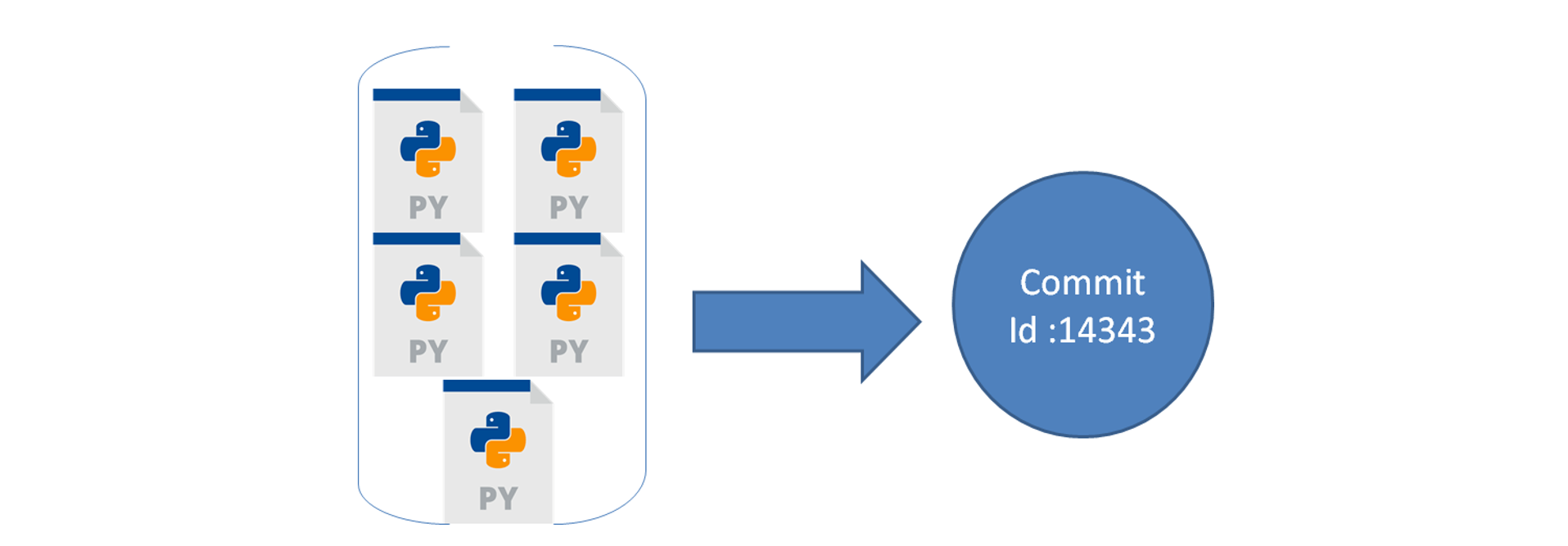
For understanding the working of git, we need to understand the two fundamental concepts in git which is git commit and git branch. Let’s understand these two terms respectively.

**What is a Commit?**

Commit is defined as the location where the code and its changes are stored. Let us take an example and discuss in brief from the diagram shown below:

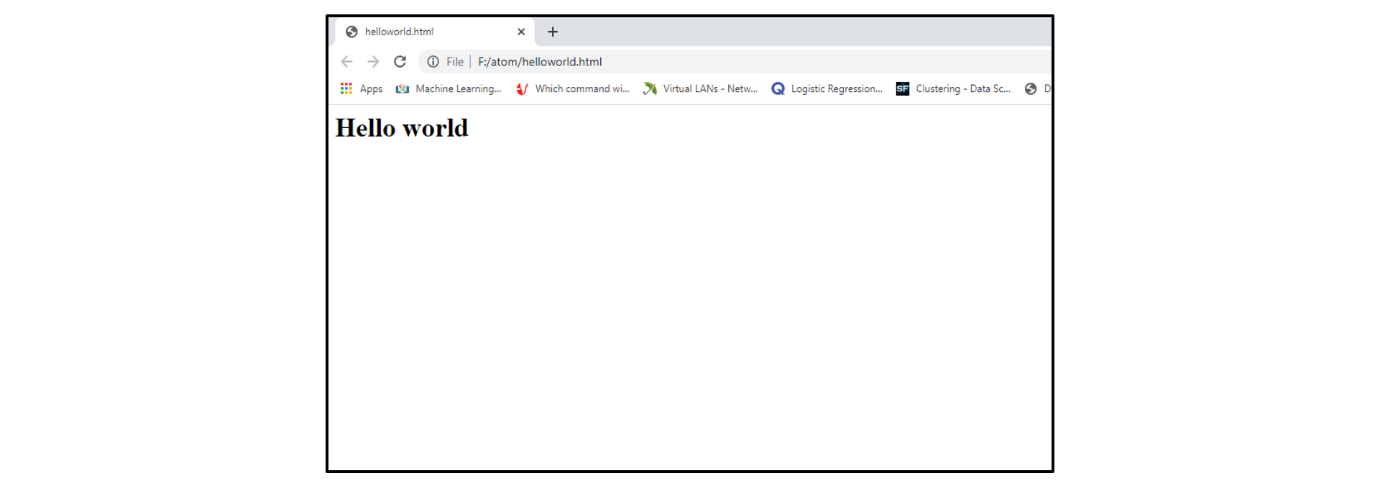
**Fig 1: The changes made are saved in a commit**

In Fig: 1, Let us assume that we have four python files. We saved them on git. These four python files will be saved inside a commit. Each commit has a commit-id, let it be 1234 in our case. Now let us say we have made some changes to the code by adding another python file. These changes in git will be saved as another commit with another commit-id 14343. This can be seen in Fig 2 . So whenever we make any change, it generates a new commit.

**Fig 2: The changes made in the repository will be saved as a new commit with a new commit id**

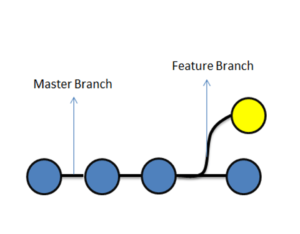
**What is a Branch?**

A Branch is a representation of different isolated versions of code. Let us take an example to understand this, Let’s say you have a website that is currently running. The website looks something like this.

  
**Fig 3: Sample example of the website before changes**

You want to add more features to this website. For this, you will have to change the code of the website. But if you are changing the code, you do not want the changes to be reflected in the main website which is deployed. So, what do you do? Ideally, you will copy the code of this website in a new folder. Make changes in the code, and then once the changes are finalised, you will replace the code in the main folder right? Let us understand how we can do the above thing in git.

So, in git to isolate different versions of code we have branches. By default, all your code is stored on a master branch. So, the website we showed you above is the master branch. Now, we don’t want to touch the master branch code, we want to copy the code of the master branch to a new place, where we can experiment or change the code. Hence, not affecting the master  branch. So, we create a new branch from the master branch, let’s call it a ‘feature branch. Now, any new changes that you will do on the feature branch, will not affect the code on the master branch.



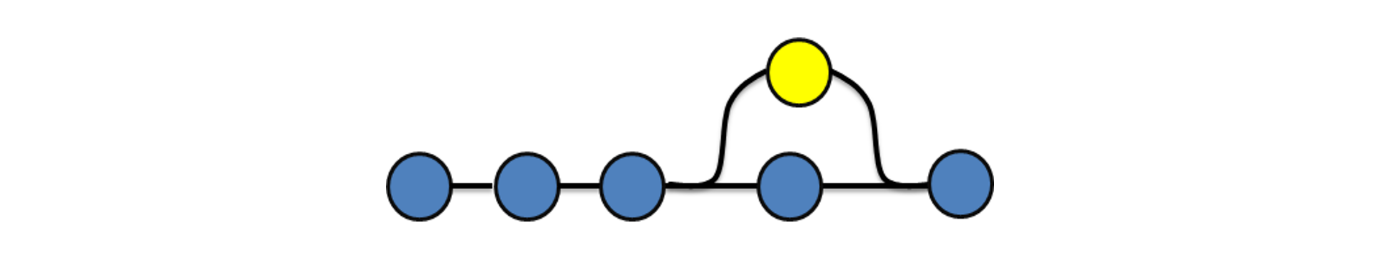
**Fig 4: Master branch and the Feature branch**

Once you are done with the changes, we simply ‘merge’ the changes of the ‘feature’ branch to the ‘master branch’. And now, the changes which were made in the feature branch will exist on the master branch as well.

[[](https://www.edureka.co/devops-engineer-training)](https://www.edureka.co/devops-engineer-training" \t "_blank)

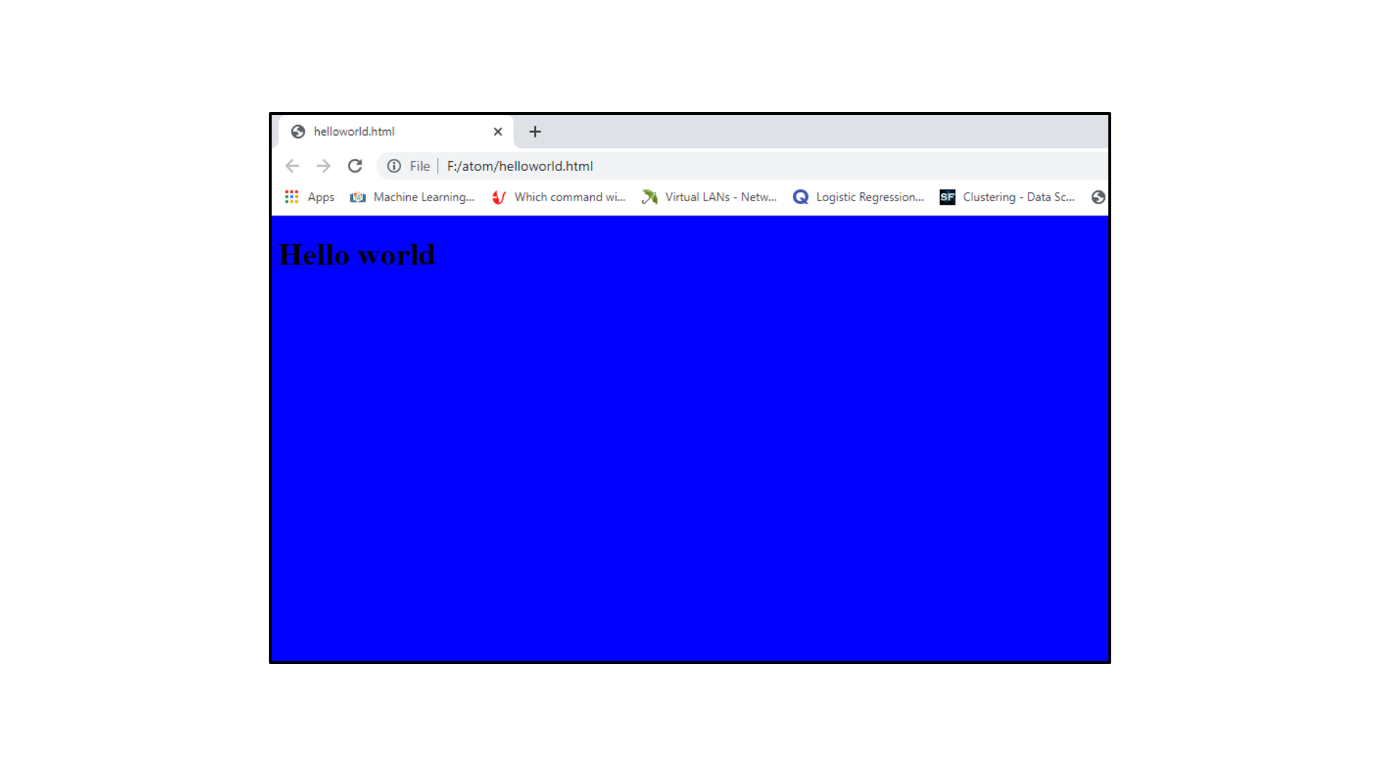
**[DevOps Engineer Masters Program](https://www.edureka.co/devops-engineer-training" \t "_blank)**

[Explore Curriculum](https://www.edureka.co/devops-engineer-training" \t "_blank)



**Fig 5: Merging of feature branch with the master branch**

If we consider the above website example after merging changes can be seen in the diagram below:

  
**Fig 6: After merging the changes into the master branch changes were depicted on the website.**

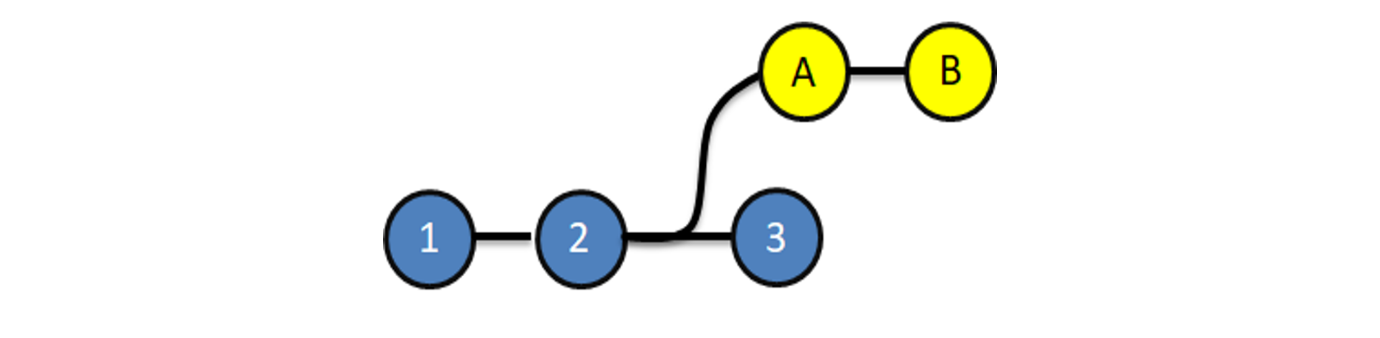
If you don’t understand merging don’t worry everything will be clear in the next section.

**What is Merging?**

In general, merging means combining something into a single entity.

Git Merge is a technique that is used to include the changes from one branch to the other branch.

So let us take an example from the diagram below, which shows the status before and after merging of two branches, feature and master branch. Blue commits are on the master branch, and yellow commits are on the feature branch.

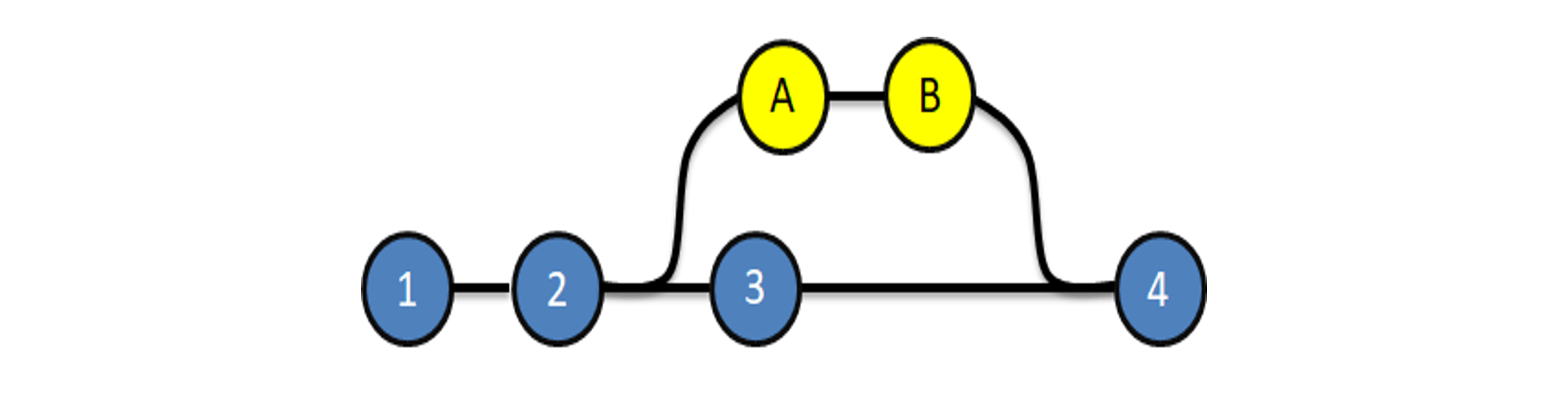
  
**Fig 7:Before merging**

In Fig 7: before merging, we can see that there are some commits on the master branch. We created a feature branch, after commit ‘2’ on master. Later, we did some changes in the feature branch in the form of commits A and B. We also did some changes on the master branch, in the form of commit 3.

In the current scenario, the master branch has the code of Commit 1,2 and 3, but it does not have changes of Commit A and B from the feature branch.

Similarly, since the feature branch was branched out from master on commit ‘2’, it has code changes from Commit 1,2, A, and B, but it does not have changes from Commit 3 on master.

Below, we have merged the feature branch on master, let us understand what happened.

**Fig 8: After merging**

We merged the feature branch on master, resulting in commit 4. Commit 4 on the master, has all the changes of the code i.e Commit 1,2, 3, A, and B. Now, that we understand merging, let’s understand the different types of merging that we can perform in git. In Git, merging is of two types:

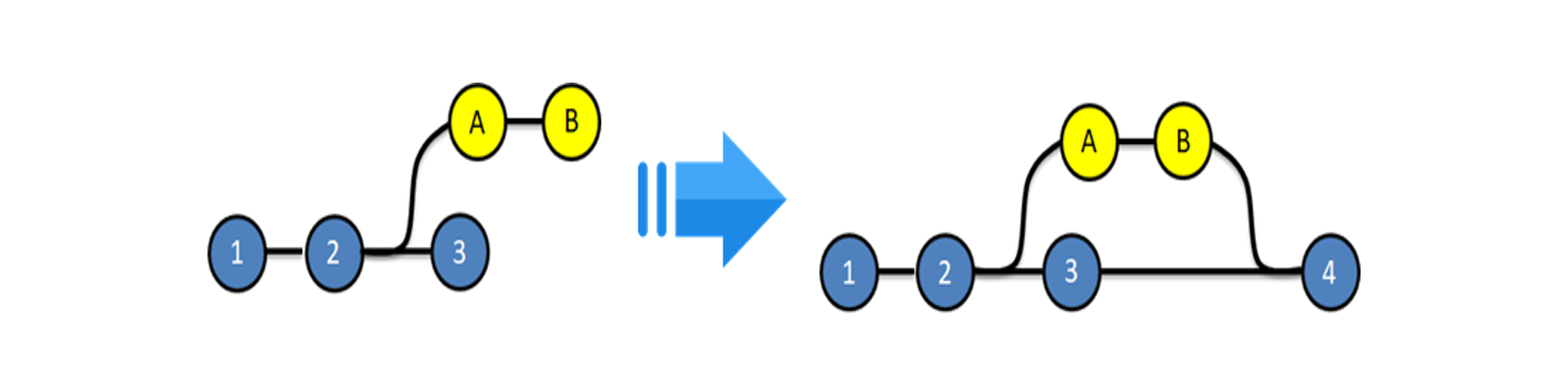
* Git Merge
* Git Rebase

Let’s understand both of them in detail

**Git Merge**  
Git merge is one of the merging techniques in git, in which the logs of commits on branches are intact.

Let us take an example if we have a project with 3 commits on the master branch as commit 1,2,3 and feature branch commits as commit A and B. If we perform a git merge operation then commits A and B will be merged as commit 4 onto the master branch. This is depicted in Fig:9.

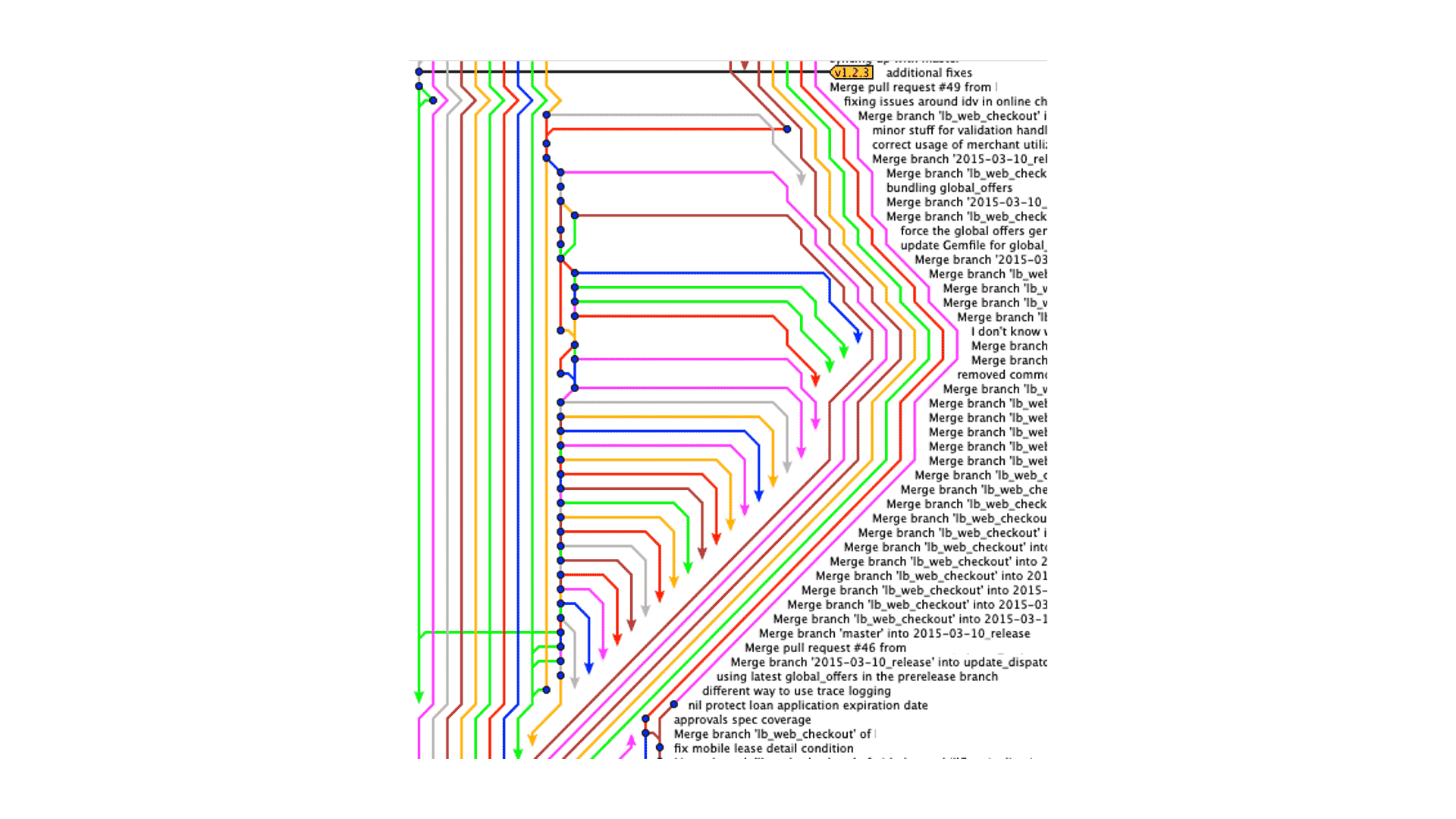
Next

  
**Fig 9: Before and after Git Merge.**

**Advantages:**

* The logs are very exhaustive and can help in understanding the complete history of how and when each merge happened
* It is easy to find mistakes and resolve them.

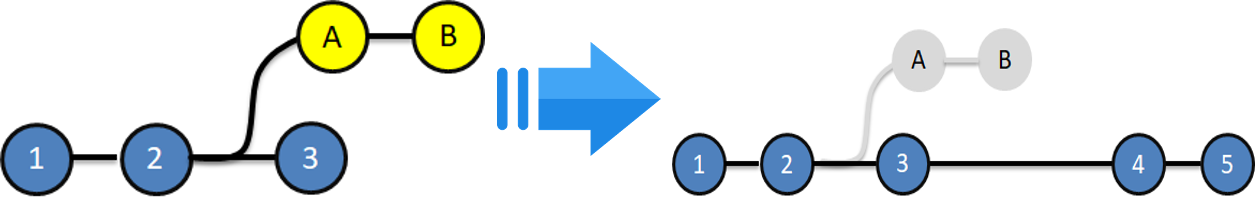
**Disadvantages:**

* Results in a clumsy log / history
* Not very user-friendly

**Fig 10: Example showing a repository with multiple branches which were merged using git-merge**

**Git-Rebase**  
Git Rebase is similar to git merge, but the logs are modified after merge in this technique. Git rebase was introduced to overcome the limitation of merging, i.e., to make logs of repository history look linear.

Let us take an example if we have a project with 3 commits on the master branch as commit 1,2,3 and feature branch commits as commit A and B. If we perform a git rebase operation then the commits A and B will be rebased on to the master branch as commit 4 and 5 and there will be no logs of the feature branches. This is depicted in Fig 12.

  
**Fig 12: Before and After git rebase**

**Advantages:**

* The logs are linear
* It’s easy to move through the project.

**Disadvantages:**

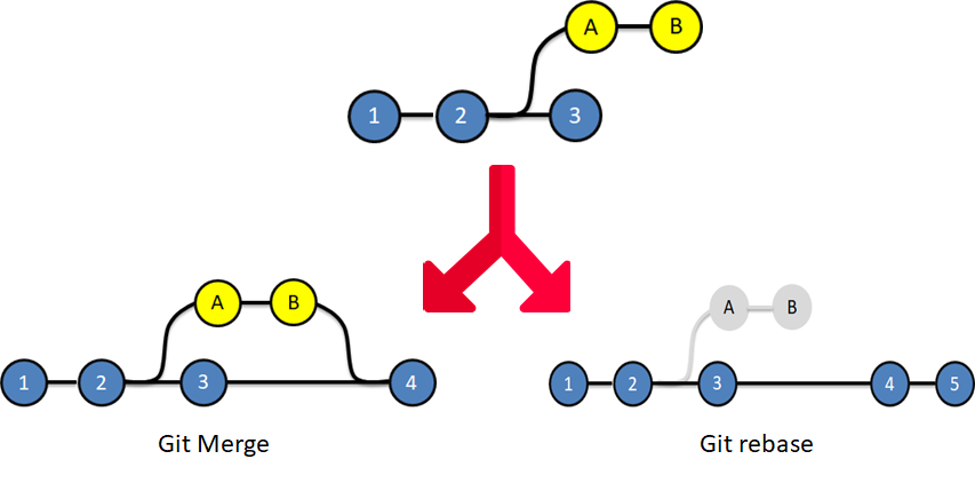
* We cannot track, when and how the commits were merged on the target branch

**Git Merge Vs Git Rebase:**

|  |  |
| --- | --- |
| Merge | Rebase |
| Git merge is a command that allows you to merge branches from Git. | Git rebase is a command that allows developers to integrate changes from one branch to another. |
| In Git Merge logs will be showing the complete history of the merging of commits. | Logs are linear in Git rebase as the commits are rebased |
| All the commits on the feature branch will be combined as a single commit in the master branch. | All the commits will be rebased and the same number of commits will be added to the master branch. |
| Git Merge is used when the target branch is shared branch | Git Rebase should be used when the target branch is private branch |

**When do we use Git Merge vs Git Rebase?**

Let’s understand it using an example, please refer diagram below:

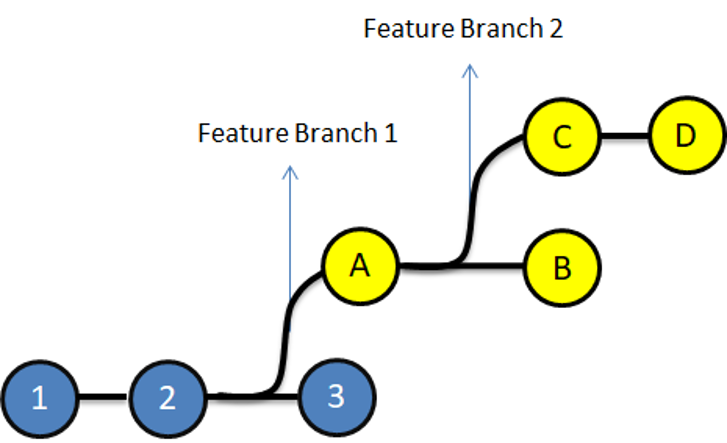
****

**Fig 13: Depicts the difference in the repository after a Git Merge vs Git Rebase**

In git merge, looking at the end diagram, one can make out Commit A and B came from the feature branch. However, in the git-rebase diagram, it is difficult to understand where commits A and B came from. Therefore, we do git merge when we want our team to understand logs in a way where they can identify where each commit is coming from. We use Git Rebase when the logs of the repository will not be referred by anyone else. To summarise, we can use Git Rebase, when we are working on branches, which cannot be seen by other developers. And we use Git Merge when the target and source branch can be viewed by other developers.

**How can Git Rebase and Git Merge be used together?**

Let us take an example if we have a project which consists of a master branch with 3 commits as 1,2,3, and 2 feature branches with commits A, B on feature branch 1 and commits C, D on feature branch 2 which is depicted in Fig 14.

****

**Fig 14: Representation of the above example**

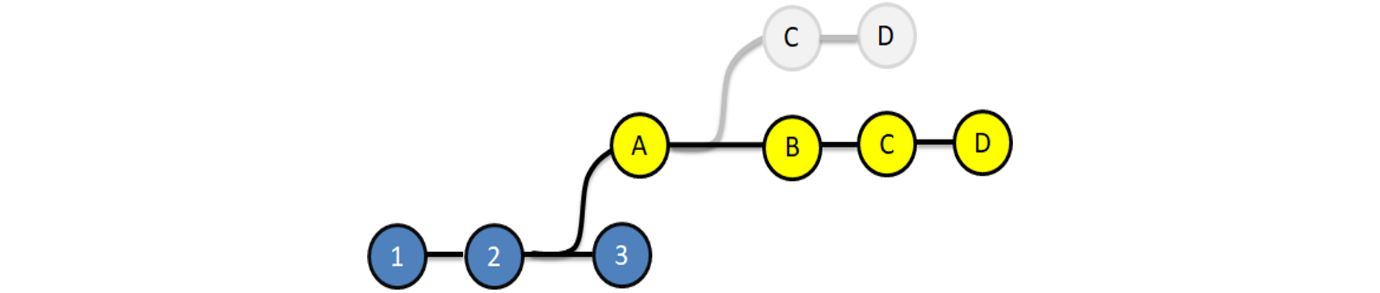
Let us assume that Developer A is working on feature branch 1.

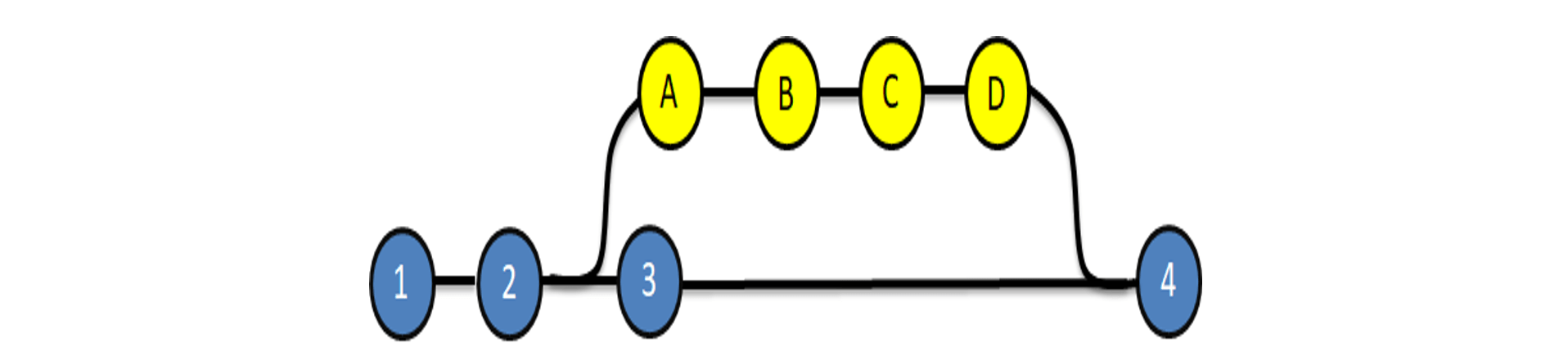
[[](https://www.edureka.co/devops-engineer-training)](https://www.edureka.co/devops-engineer-training" \t "_blank)

**[DevOps Engineer Masters Program](https://www.edureka.co/devops-engineer-training" \t "_blank)**

[Weekday / Weekend](https://www.edureka.co/devops-engineer-training" \t "_blank)

To experiment with the code, he creates another branch Feature branch 2, does some changes in it, and finalizes it with Commits C and D.  
He does not want anyone to know about his private branch, because it’s unnecessary. So, he can rebase Feature 2 on Feature 1.

  
Now he can finally, merge feature 1 branch on master resulting in the below diagram.

The above is how the final log will look like to other developers. They can only see commits being made on feature branch 1, which was then finally merged on the master branch, with commit 4.