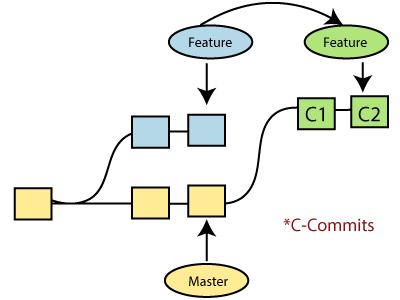
Git Rebase

Rebasing is a process to reapply commits on top of another base trip. It is used to apply a sequence of commits from distinct branches into a final commit. It is an alternative of git merge command. It is a linear process of merging.

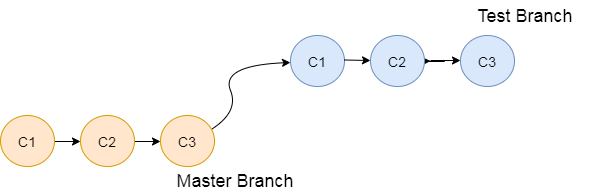
In Git, the term rebase is referred to as the process of moving or combining a sequence of commits to a new base commit. Rebasing is very beneficial and it visualized the process in the environment of a feature branching workflow.

It is good to rebase your branch before merging it.



Generally, it is an alternative of git merge command. Merge is always a forward changing record. Comparatively, rebase is a compelling history rewriting tool in git. It merges the different commits one by one.

Suppose you have made three commits in your master branch and three in your other branch named test. If you merge this, then it will merge all commits in a time. But if you rebase it, then it will be merged in a linear manner. Consider the below image:



The above image describes how git rebase works. The three commits of the master branch are merged linearly with the commits of the test branch.

Merging is the most straightforward way to integrate the branches. It performs a three-way merge between the two latest branch commits.

How to Rebase

When you made some commits on a feature branch (test branch) and some in the master branch. You can rebase any of these branches. Use the git log command to track the changes (commit history). Checkout to the desired branch you want to rebase. Now perform the rebase command as follows:

**Syntax:**

1. $git rebase **<branch** name**>**

If there are some conflicts in the branch, resolve them, and perform below commands to continue changes:

1. $ git status

It is used to check the status,

1. $git rebase --continue

The above command is used to continue with the changes you made. If you want to skip the change, you can skip as follows:

1. $ git rebase --skip

When the rebasing is completed. Push the repository to the origin. Consider the below example to understand the git merge command.

Suppose that you have a branch say **test2** on which you are working. You are now on the test2 branch and made some changes in the project's file **newfile1.txt**.

Add this file to repository:

1. $ git add newfile1.txt

Now, commit the changes. Use the below command:

1. $ git commit -m "new commit for test2 branch."

The output will look like:

[test2 a835504] new commitfor test2 branch

1 file changed, 1 insertion(+)

Switch the branch to master:

1. $ git checkout master

**Output:**

Switched to branch 'master.'

Your branch is up to date with 'origin/master.'

Now you are on the master branch. I have added the changes to my file, says **newfile.txt**. The below command is used to add the file in the repository.

1. $ git add newfile.txt

Now commit the file for changes:

1. $ git commit -m " new commit made on the master branch."

**Output:**

[master 7fe5e7a] new commit made on master

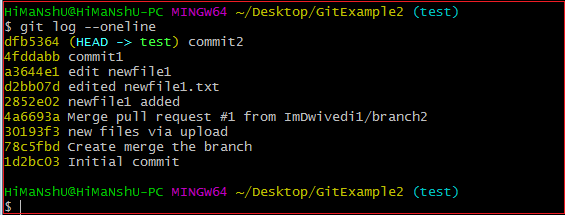
1 file changed, 1 insertion(+)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)

To check the log history, perform the below command.

1. $ git log --oneline

**Output:**



As we can see in the log history, there is a new commit in the master branch. If I want to rebase my test2 branch, what should I do? See the below rebase branch scenario:

Rebase Branch

If we have many commits from distinct branches and want to merge it in one. To do so, we have two choices either we can merge it or rebase it. It is good to rebase your branch.

From the above example, we have committed to the master branch and want to rebase it on the test2 branch. Let's see the below commands:

1. $ git checkout test2

This command will switch you on the test2 branch from the master.

**Output:**

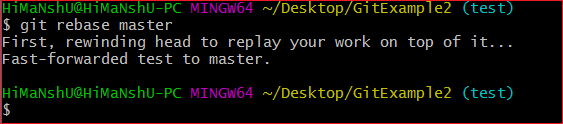
Switched to branch 'test2.'

Now you are on the test2 branch. Hence, you can rebase the test2 branch with the master branch. See the below command:

1. $ git rebase master

This command will rebase the test2 branch and will show as **Applying: new commit on test2 branch**. Consider the below output:

**Output:**



Git Interactive Rebase

Git facilitates with Interactive Rebase; it is a potent tool that allows various operations like **edit, rewrite, reorder,** and more on existing commits. Interactive Rebase can only be operated on the currently checked out branch. Therefore, set your local HEAD branch at the sidebar.

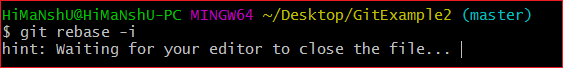
Git interactive rebase can be invoked with rebase command, just type **-i** along with rebase command. Here '**i**' stands for interactive. Syntax of this command is given below:

**Syntax:**

1. $ git rebase -i

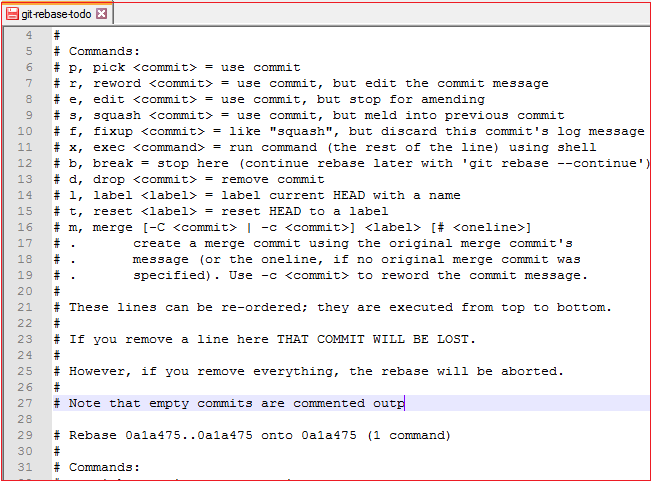
It will list all the available interactive options.

**Output:**



After the given output, it will open an editor with available options. Consider the below output:

**Output:**



When we perform the git interactive rebase command, it will open your default text editor with the above output.

The options it contains are listed below:

* Pick
* Reword
* Edit
* Squash
* Fixup
* Exec
* Break
* Drop
* Label
* Reset
* Merge

The above options perform their specific tasks with git-rebase. Let's understand each of these options in brief.

**Pick (-p):**

Pick stands here that the commit is included. Order of the commits depends upon the order of the pick commands during rebase. If you do not want to add a commit, you have to delete the entire line.

**Reword (-r):**

The reword is quite similar to pick command. The reword option paused the rebase process and provides a chance to alter the commit message. It does not affect any changes made by the commit.

**Edit (-e):**

The edit option allows for amending the commit. The amending means, commits can be added or changed entirely. We can also make additional commits before rebase continue command. It allows us to split a large commit into the smaller commit; moreover, we can remove erroneous changes made in a commit.

**Squash (-s):**

The squash option allows you to combine two or more commits into a single commit. It also allows us to write a new commit message for describing the changes.

**Fixup (-f):**

It is quite similar to the squash command. It discarded the message of the commit to be merged. The older commit message is used to describe both changes.

**Exec (-x):**

The exec option allows you to run arbitrary shell commands against a commit.

**Break (-b):**

The break option stops the rebasing at just position. It will continue rebasing later with '**git rebase --continue**' command.

**Drop (-d):**

The drop option is used to remove the commit.

**Label (-l):**

The label option is used to mark the current head position with a name.

**Reset (-t):**

The reset option is used to reset head to a label.

GitMerge vs. Rebase

It is a most common puzzling question for the git user's that when to use merge command and when to use rebase. Both commands are similar, and both are used to merge the commits made by the different branches of a repository.

Rebasing is not recommended in a shared branch because the rebasing process will create inconsistent repositories. For individuals, rebasing can be more useful than merging. If you want to see the complete history, you should use the merge. Merge tracks the entire history of commits, while rebase rewrites a new one.

Git rebase commands said as an alternative of git merge. However, they have some key differences:

|  |  |
| --- | --- |
| **Git Merge** | **Git Rebase** |
| Merging creates a final commit at merging. | Git rebase does not create any commit at rebasing. |
| It merges all commits as a single commit. | It creates a linear track of commits. |
| It creates a graphical history that might be a bit complex to understand. | It creates a linear history that can be easily understood. |
| It is safe to merge two branches. | Git "rebase" deals with the severe operation. |
| Merging can be performed on both public and private branches. | It is the wrong choice to use rebasing on public branches. |
| Merging integrates the content of the feature branch with the master branch. So, the master branch is changed, and feature branch history remains consistence. | Rebasing of the master branch may affect the feature branch. |
| Merging preserves history. | Rebasing rewrites history. |
| Git merge presents all conflicts at once. | Git rebase presents conflicts one by one. |