GIT Commands

**1. git init**

**Usage: git init [repository name]**

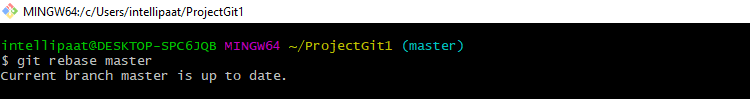
We have to navigate to our project directory and type the command **git init** to initialize a Git repository for our local project folder. Git will create a hidden **.git** directory and use it for keeping its files organized in other subdirectories.

### git rebase

**Usage: git rebase [base]**

Rebase is the process of moving and combining a sequence of commits to a new base commit. Rebasing is changing the base of our branch from one commit to another, making it appear as if we’ve created our branch from a different commit. Internally, Git accomplishes this by creating new commits and applying them to the specified base. It’s very important to understand that even though the branch looks the same, it is composed of entirely new commits.

The git rebase command performs an automatic git checkout <branch> before doing anything else. Otherwise, it remains on the current branch.

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand26.png)  
Consider a situation where we have branched off from the master and have created a feature branch, but the master branch is still having more commits. We want to get the updated version of the master branch in our feature branch, keeping our branch’s history clean, so that it appears as if we are working on the latest version of the master branch.

**Note**: We don’t rebase public history. We should never rebase commits once they are pushed to a public repository. Why because the rebase would replace the old commits with the new ones, and it would appear that a part of our project history got abruptly vanished.

# How to Git rebase a branch to master by example

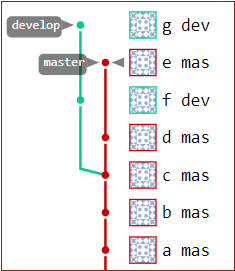
Most code integrations happen with a merge, but occasionally a developer wants to get their local code caught up with the master branch through a rebase instead. In this example we will demonstrate how to git rebase a branch to master instead of [using the merge](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-merge-master-into-any-branch-in-GitLab) command, while also pointing out the impact of performing a git rebase to master operation.

### What does Git rebase to master mean?

In this tutorial, we will take the branch named develop and rebase it onto the tip of master.

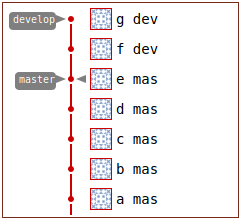
The develop branch broke off from master at commit C, so both branches share the files a.html, b.html and c.html. See the image below to help visualize this.

Since the branch occurred, master has added commits D and E. This means master has two files that develop does not have, namely d.html and e.html.



This is how the Git repository looks before a git rebase to master.

Furthermore, since the split, there have been two new commits on the develop branch, adding the files f.html and g.html. These are files the develop branch has that the master branch does not.



This is how the GitLab repository looks after the git rebase to master.

### Impact of the Git rebase

After a successful develop branch to master rebase:

* The files in the master branch will not change
* The develop branch will additionally acquire all of the master branch’s new files
* The develop stream’s branch point will change.
  + It will appear as though the develop branch split after commit E on the master branch.
  + Before the rebase, the develop branch split from master at commit C.

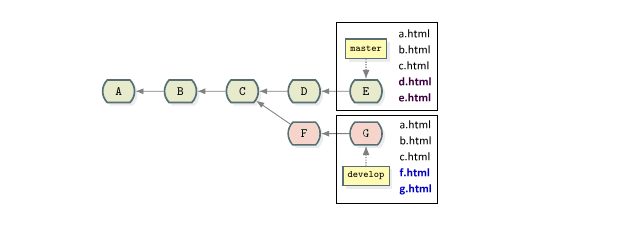
### Git rebase to master command syntax

The operation to perform a Git rebase to master is straight forward. Simply append to the end of the command the name of the source branch and then the name of the branch to rebase. To rebase develop to master the command is as follows:

git rebase master develop

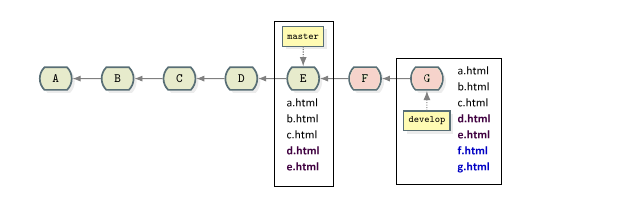
Warning: There is a git rebase onto switch which sometimes developers incorrectly believe they need to include with the rebase command. Doing so will cause commits to be discared and files to be lost. Do not make the mistake of performing a git rebase onto operation.

Prior to the rebase, the develop branch had only five files.



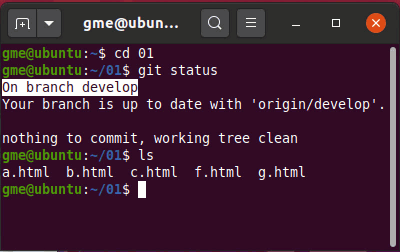
Each branch has five files before the git rebase to master operation.

After the Git rebase to master, the develop branch has seven files. It has retained all of its original files and acquired two new files from the tip of the master branch. However, the number of files in the master branch remains unchanged.



After the git rebase of develop to master, the develop branch acquires all of master’s files.

The fact that the master branch did not acquire any files from the develop branch often throws new Git users off, but this is the expected behavior. A git rebase does not synchronize between branches. It simply catches one branch up with the latest changes from another.



The number of files in develop increase after the git rebase to master.

### Proceed with caution

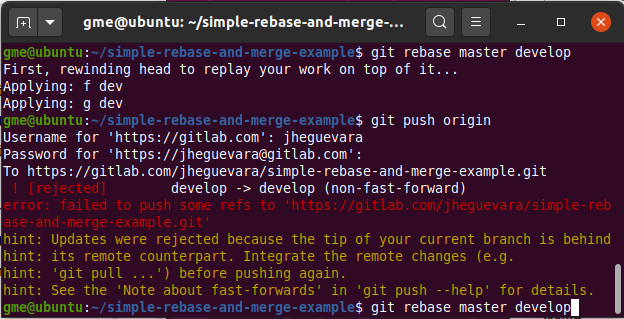
When a Git rebase occurs, the repository’s commit history is irreparably changed. After the rebase in our example, commits F and G are assigned brand new commit ids, and the old commit ids are discarded. For this reason, you will often see rebased commits marked as F’ and G’ to emphasize the fact that new commit ids have been assigned.

If you rebase a branch shared with another developer and push your changes back to [GitHub](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-rebase-GitHub-repos) or GitLab, a fellow developer will run into a variety or problems when they attempt to pull the rebased repository into their local development environment. Commits they have locally will have disappeared on the remote, and the remote branch will have an incompatible branch history. As you could imagine, rebasing a shared branch or [squashing Git commits](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-squash-git-commits-example) can wreak havok.

### Rebase to GitHub or GitLab

In fact, if you rebase and try to [push to GitLab](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-git-push-GitLab-commits-to-origin) or GitHub, the server won’t allow the operation to be performed. To rebase to GitHub or GitLab, a developer must add the –force switch to the git push command to compel the changes to be accepted.

git push origin --force



A GitLab or GitHub rebase push will be rejected unless forced.

The accepted logic is to only git rebase to master branches local to your personal workspace. That way your [Git clean up commands](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-clean-up-Git-branches-and-commits) don’t impact anyone else. Do not perform a git rebase to master on branches shared with other developers. A [rebase onto master](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-Git-rebase-a-branch-to-master-example) is fine, just not the other way around. Otherwise you’re likely to break the build and push or pull operations will be put on stand-by, thwarting continued development and making you persona non grata with the rest of the [developers and DevOps](https://www.theserverside.com/video/Developer-vs-DevOps-roles-Whats-the-difference) team.