Git fetch

<https://www.geeksforgeeks.org/git-difference-between-git-fetch-and-git-pull/>

Git Fetch is the command that tells the local repository that there are changes available in the remote repository without bringing the changes into the local repository.

Let us create a file called ***demo.txt***with “***Hello Geeks”*** content inside it initialize the directory to a git repository and push the changes to a remote repository.

* *git init*
* *git add <Filename>*
* *git commit -m <Commit Message>*
* *git remote add origin <Link to your remote repository>*
* *git push origin <branch name>*

Now, we have my ***demo.txt*** in the remote repository.

The local and the remote repositories are now in sync and have the same content at both places. Let’s now update our ***demo.txt*** in the remote repository.

Now since we have updated our ***demo.txt*** remotely, let’s bring the changes to our local repository. Our local repository has only 1 commit while the remote repository now has 2 commits (*observe the second commit starting from****4c4fcb8***). Let’s use the ***git fetch***command to see in the local repository whether we have a change in the remote repository or not. Before that let’s use the ***git log***command to see our previous commits.

We can see that after using ***git fetch***we get the information that there is some commit done in the remote repository. (*notice the****4c4fcb8****which is the initials of our 2nd commit in a remote repository*). To merge these changes into our local repository, we need to use the ***git merge origin/<branch name>***command.

Let us have a look at our commits in the local repository usingthe ***git log***command.

Git init

Git remote add origin <>

Git branch -M main

Git add .

Git commit -m

Git push origin main

Make changes in remote repo

Commit the changes

Git fetch origin main

Git merge origin/main (Git merge source-branch)

Changes will be reflected in the logs and working directory

Git pull

Git Pull on the other hand brings the copy of the remote directory changes into the local repository.

Git init

Git remote add origin <>

Git branch -M main

Git add .

Git commit -m

Git push origin main

Make changes in remote repo

Commit the changes

Git pull origin main

***git pull origin <branch name>*** command.

Git pull source-branch

**git fetch origin**

... and then do:

**git diff main origin/main**

... in order to see the difference between your master, and the one on GitHub. If you're happy with those differences, you can merge them in with **git merge origin/main**, assuming **main** is your current branch.

If you're not interested in the details that git diff outputs, you can just run git cherry which will output a list of commits your remote tracking branch has ahead of your local branch.

For example:

git fetch origin

git cherry master origin/master

Will output something like:

+ 2642039b1a4c4d4345a0d02f79ccc3690e19d9b1

+ a4870f9fbde61d2d657e97b72b61f46d1fd265a9

It indicates that there are two commits in my remote tracking branch that haven't been merged into my local branch.

This also works the other way:

git cherry origin/master master

It will show you a list of local commits that you haven't pushed to your remote repository yet.

Git branch

To create a branch

**Git branch <branch-name>**

Or

**Git checkout -b <branch-name>**

To switch branch

**Git checkout <branch-name>**

To delete a branch

**git branch -d branch-to-delete**

Force delete

**git branch -D branch-to-delete**

to delete remote branch

**git push origin --delete remote-branch-to-delete**

How to come out of git log: press Q

Git rebase

Git rebase source-branch

`rebase` is used on the feature branch whereas `merge` is performed of the base branch.

how to unstage a file in git

git reset HEAD <file>

e.g.: git reset HEAD example.txt

This will unstage the file and discard any changes made to it, reverting it to the state of the last committed version.

Top of Form

git checkout -- example.txt

git rm file vs git checkout file

**git rm <file>:**

* The **git rm** command is used to remove a file from both the working directory and the staging area (index). It is used when you want to stop tracking a file in your Git repository.

**git checkout <file>:**

* The **git checkout** command, when used with a file, is generally used to discard changes made to that file in the working directory. It retrieves the last committed version of the file and overwrites the changes.

**Key Differences:**

* **git rm** removes a file from both the working directory and the staging area, preparing it for a commit to stop tracking the file.
* **git checkout** (with a file) is used to discard changes in the working directory, reverting the file to the last committed version.

**Important Note:**

* If you want to keep the file in your working directory but stop tracking it, you should use **git rm --cached <file>**. This command removes the file from the staging area but leaves it in the working directory.
* After using **git rm** or **git checkout**, you need to commit the changes using **git commit** to make them part of your Git history.

Git bare repository

https://blog.opstree.com/2022/06/21/what-is-a-bare-git-repository/

[Repositories](https://www.geeksforgeeks.org/what-is-a-git-repository/) in [Git](https://www.geeksforgeeks.org/git-lets-get-into-it/) are a snapshot of the folder in which you are working on your project. You can track the progress and changes made to the project by making commits and also revert changes if not satisfactory.  
Repositories can be divided into two types based on the usage on a server. These are:

* Non-bare Repositories
* Bare Repositories

**What is a bare repository?**  
A bare repository is the same as default, but no commits can be made in a bare repository. The changes made in projects cannot be tracked by a bare repository as it doesn’t have a working tree. A working tree is a directory in which all the project files/sub-directories reside. Bare repository is essentially a .git folder with a specific folder where all the project files reside.  
Practically speaking everything in the repository apart from .git is a part of working tree.

 Since you cannot commit, or make changes to it, bare repositories are pretty useless on their own. But then why does it exist? When people collaborate to work on a project, they need a central repository where all the tracked changes are stored and prevent any conflict between the versions of the project on other’s computers. A central repository also means that any new contributor can clone the repository into a local one without getting any unsaved changes or conflicting work of others (in short, no mess). A central repository was strictly supposed to be something like a reference repository.

This requires one to use a remote repository as a central one, and initially, only Bare repositories could be used as remote repositories. With the latest changes in git, central repositories need not be bare, hence not many people know about it properly.  
The only possible operations on the Bare Repository are Pushing or Cloning.

Generally, a git bare repository contains an extension of .git. In this repository you cannot commit any changes hence it cannot keep a track of the changes made in your files. But you can definitely push, pull, fetch or clone from it. In other words, you can use any git commands applicable to a remote repository.

But you can definitely push, pull, fetch or clone from it.

git clone https://github.com/Sudha-Agarwal/MEAN\_Phase1\_Dec.git cloned

cd cloned/

ls -la

git clone --bare https://github.com/Sudha-Agarwal/MEAN\_Phase1\_Dec.git cloned\_bare

cd cloned\_bare/

ls-la

git status

git add .

cd ..

ls-l

git clone cloned\_bare/ cloned\_nb – this will clone the repo as well as working directory

cd cloned\_nb/

ls-la

git remote -v

git status

git push origin

git init –bare

Git HEAD

**Featured snippet from the web**

HEAD can be termed as **a special ref that points to the current commit**.

The **HEAD** points out the last commit in the current checkout branch. It is like a pointer to any reference. The HEAD can be understood as the "**current branch**."

The **git show head** is used to check the status of the Head. This command will show the location of the Head.

git show HEAD –oneline

 It will tell you what HEAD is currently referencing, but also list all other branches that are also referencing the same commit;

For example, instead of typing $ git show HEAD --oneline, you can instead write $ git show @ --oneline.

Git Index

The Git index is **a critical data structure in Git**. It serves as the “staging area” between the files you have on your filesystem and your commit history. When you run git add , the filesfrom your working directory are hashed and stored as objects in the index, leading them to be “staged changes”.

The staged file means it is in the staging area, i.e., in the index. you can also unstaged your changes using the command:

git restore --staged <file>

Git master

In Git, "master" is **a naming convention for a branch**. After cloning (downloading) a project from a remote server, the resulting local repository has a single local branch: the so-called "master" branch. This means that "master" can be seen as a repository's "default" branch.