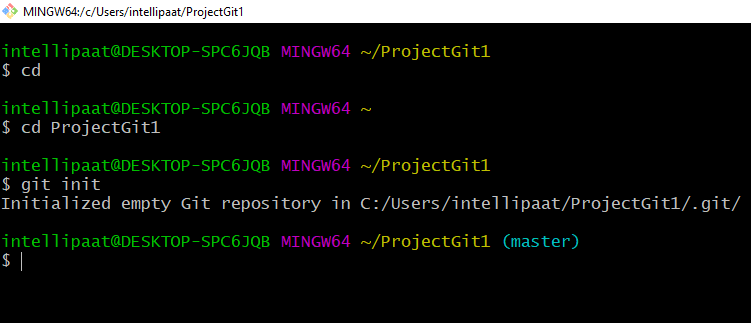
**1. git init**

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

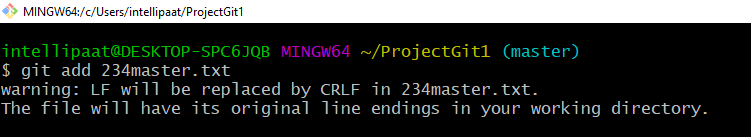
We have to navigate to our project directory and type the command **git init** to initialize a Git repository for our local project folder. Git will create a hidden **.git** directory and use it for keeping its files organized in other subdirectories.

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand1.png)

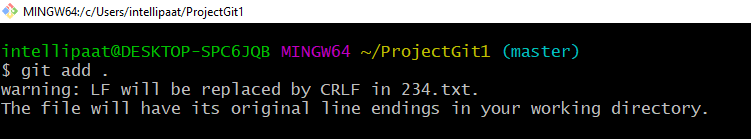
**2. git add**

**Usage (i): git add [file(s) name]**

This will add the specified file(s) into the Git repository, the staging area, where they are already being tracked by Git and now ready to be committed.

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand2.png)  
**Usage (ii): git add . or git add \***

This will take all our files into the Git repository, i.e., into the staging area.

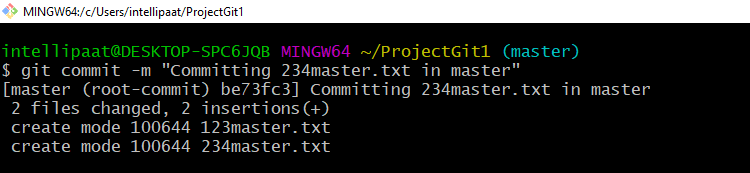
[](https://intellipaat.com/mediaFiles/2019/07/GItCommand3.png)  
We can use this command as **git add -A** as well.

Bottom of Form

**3. git commit**

**Usage: git commit -m “message”**

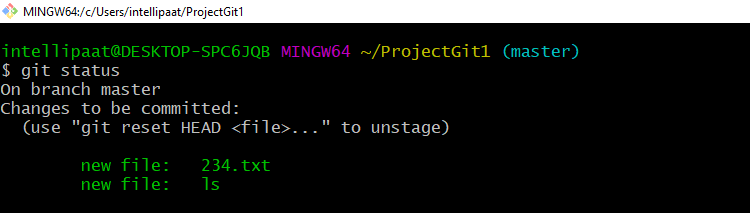
This command records or snapshots files permanently in the version history. All the files, which are there in the directory right now, are being saved in the Git file system.

**[](https://intellipaat.com/mediaFiles/2019/07/GItCommand4.png)**

**4. git status**

**Usage: git status**

This command will show the modified status of an existing file and the file addition status of a new file, if any, that has to be committed.

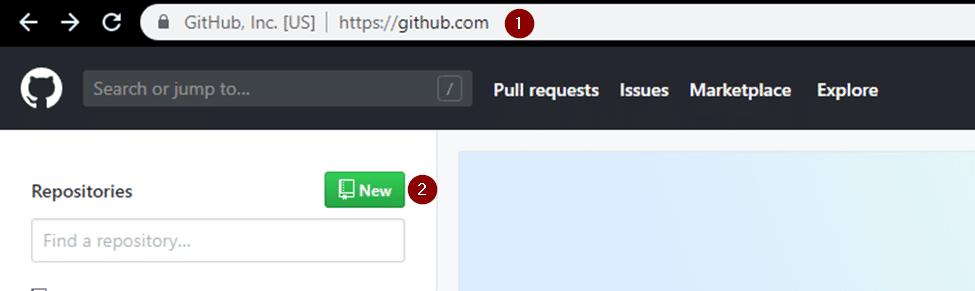
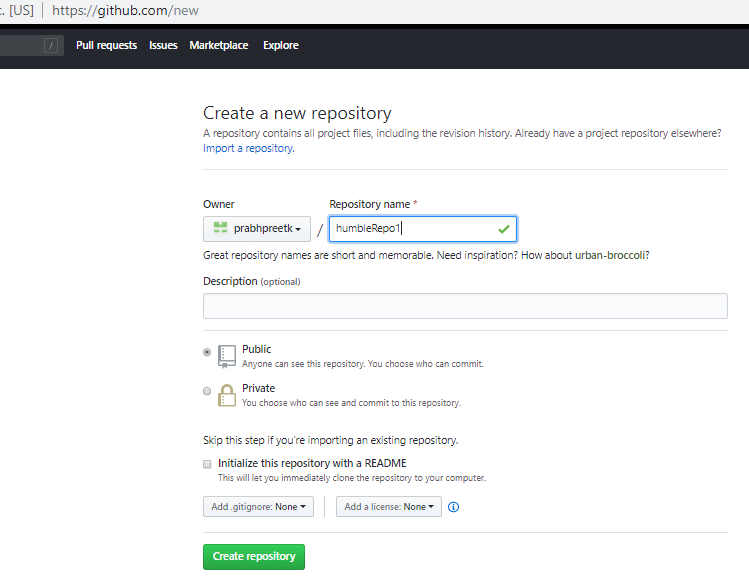
[](https://intellipaat.com/mediaFiles/2019/07/GItCommand5.png)

**5. git remote**

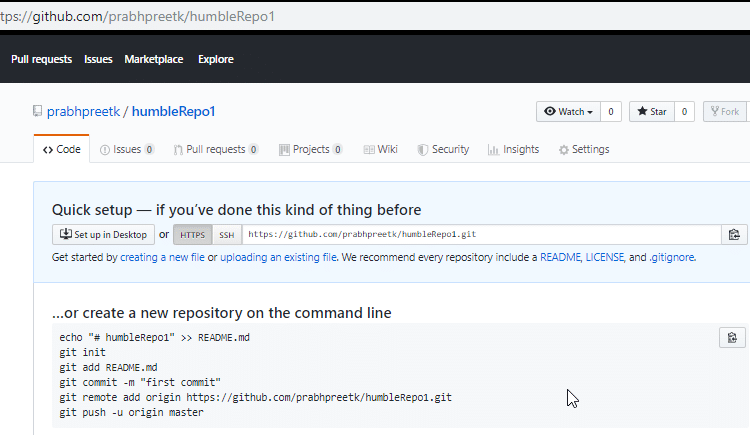
**Usage: git remote add origin “[URL]”**

Once everything is ready on our local system, we can start pushing our code to the remote (central) repository of the project. For that, follow the below steps:

**Step 1:**  
**(1)**Login to the **GitHub account** if the account already exists (If not, sign up on github.com)  
**(2)** Click on**New**

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand6.png)  
**Step 2:**Now, we have to create a new repository. Provide a **name**to our**repository**, select the **privacy**of the repository as**Public**, and then click on **Create repository**  
**[](https://intellipaat.com/mediaFiles/2019/07/GItCommand7.png)**

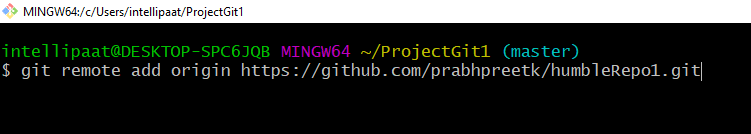
Once we are done with filling up the new repository form, we would land on a page as follows:

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand8.png)

**Step 3:** Click on the Copy icon on the right side of the URL box of the Github repository to copy the link and paste it as shown below:

git remote add origin “URL”

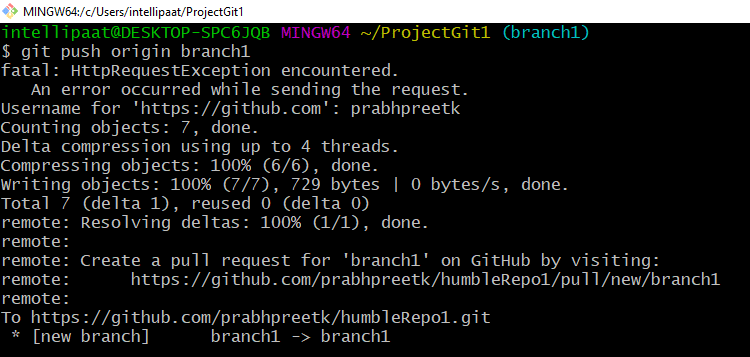
Now, we are ready to operate the remote commands in our repository that we have just created.

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand9.png)

**6. git push**

**Usage: git push origin [branch name]**

Suppose, we have made some changes in the file and want to push the changes to our remote repository on a particular branch. By using the command ‘git push,’ the local repository’s files can be synced with the remote repository on Github.

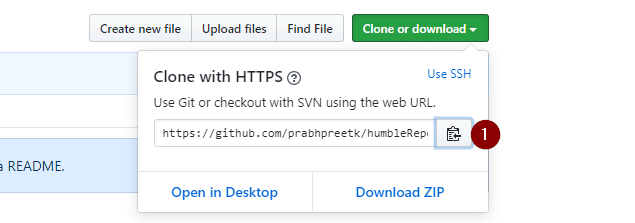
[](https://intellipaat.com/mediaFiles/2019/07/GItCommand10.png)

**7. git clone**

**Usage: git clone [URL]**

Suppose, we want to work on a file that is on a remote Github repository as another developer. How can we do that? We can work on this file by clicking on **Clone or** **Download**and copying the link and pasting it on the terminal with the git clone command. This will import the files of a project from the remote repository to our local system.

**(1)** The below screenshot shows from where to copy the link:

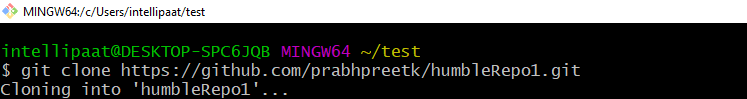
[](https://intellipaat.com/mediaFiles/2019/07/GItCommand11.png)To create a local folder, we have to use the following command:

mkdir [directory- name]

cd [directory- name]

git clone [URL]

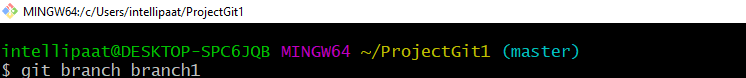
Now, paste the copied link along with the git clone command as shown below:

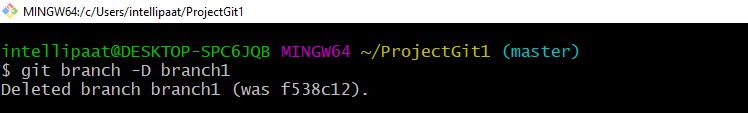
[](https://intellipaat.com/mediaFiles/2019/07/GItCommand12.png)**Note**: Here, we don’t have to use the **git remote add origin** command because we have already cloned the remote repository in the local directory. Now, if we push any new file, it knows where it has to go.

**8. git branch**

**Usage (i): git branch [name-of-the-branch]**

So far, we saw how we can work on Git. Now, imagine, multiple developers working on the same project or repository! To handle the workspace of multiple developers, we can use branches. To create a branch (say, the ‘name-of-the-branch’ is ‘branch1’), we use this command:

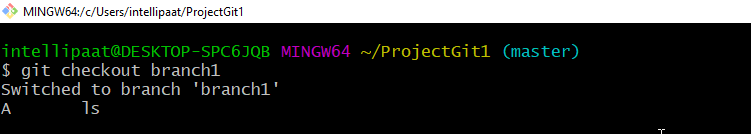
[](https://intellipaat.com/mediaFiles/2019/07/GItCommand13.png)  
**Usage (ii): git branch -D [name -of-the-branch]**

Similarly, to delete a branch, we use the **git branch -D**command:  
**[](https://intellipaat.com/mediaFiles/2019/07/GItCommand14.png)**

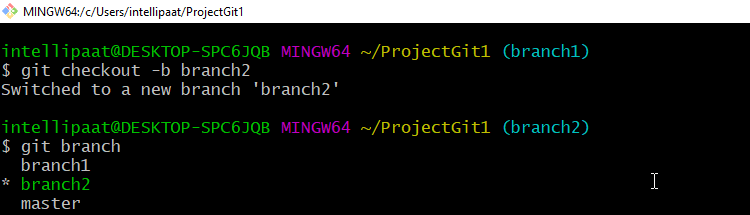
**9. git checkout**

**Usage (i): git checkout [name-of-the-new-branch]**

We use this command to navigate to an existing branch, add new files, and commit the files:

**[](https://intellipaat.com/mediaFiles/2019/07/GItCommand15.png)Usage (ii): git checkout -b [name-of-the-new-branch]**

We use this command to create a branch and navigate to that particular branch (say, the ‘name-of-the-new-branch’ is ‘branch2’) at the same time:

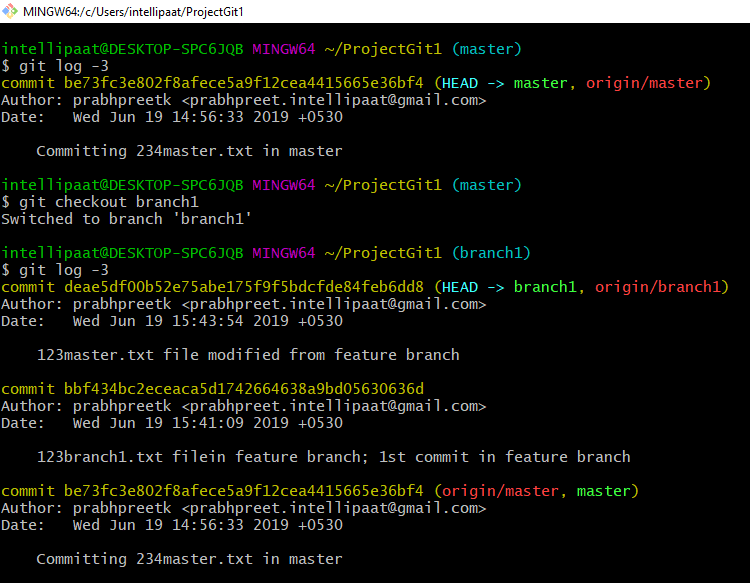
[](https://intellipaat.com/mediaFiles/2019/07/GItCommand16.png)

**10. git log**

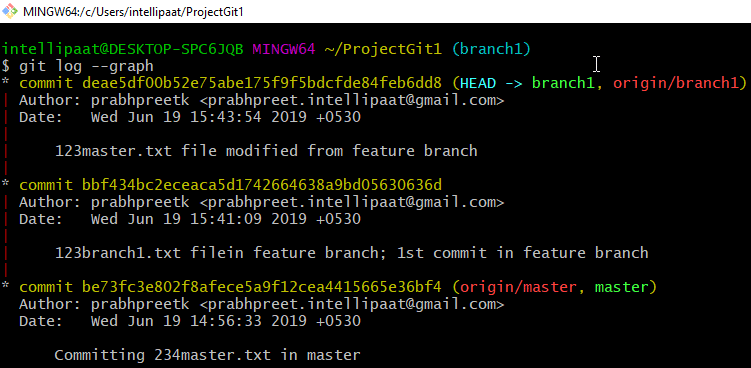
**Usage (i): git log**

This command is used when we want to check the log for every commit in detail in our repository.

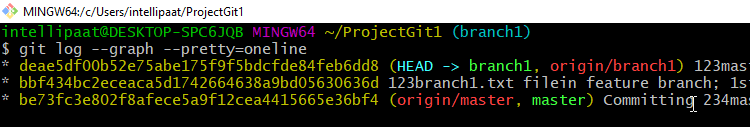
**Note**: It will show the log of the branch we are in. We can check the last three logs by giving the command: **git log -3**

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand17.png)

**Usage (ii): git log –graph**

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand18.png)

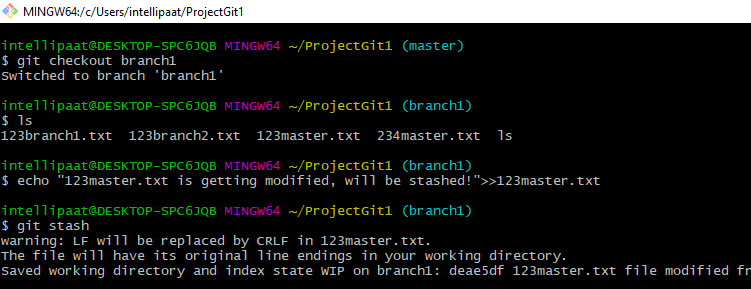
**Usage (iii): git log –graph –pretty=oneline**

**[](https://intellipaat.com/mediaFiles/2019/07/GItCommand19.png)**

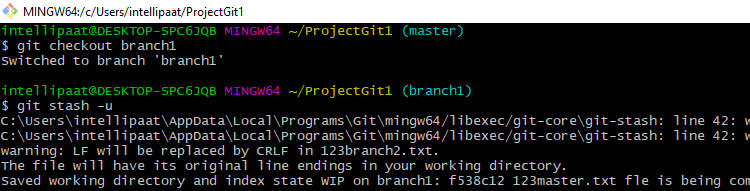
**11. git stash**

**Usage (i): git stash**

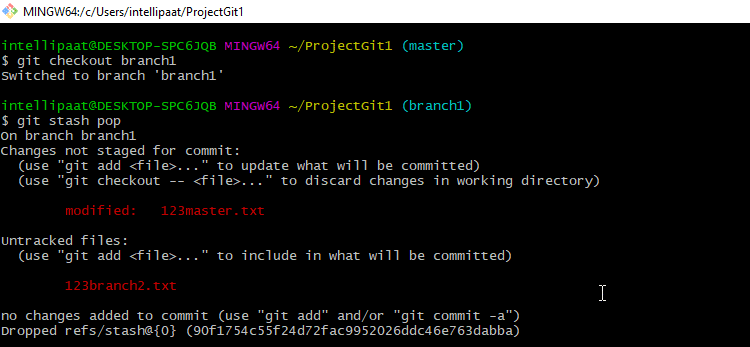
This command can be used when we want to save our work without staging or committing the code to our Git repository and want to switch between branches.

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand20.png)  
**Usage (ii): git stash -u**

This command is used when we want to stash the untracked files.

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand21.png)  
**Usage (iii): git stash pop**

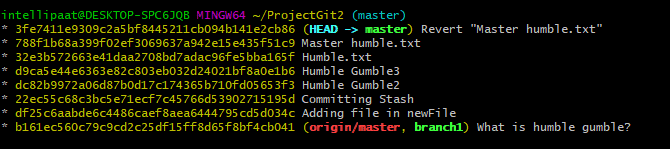
This command is used when we are back on our branch and want to retrieve the code.

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand22.png)

**12. git revert**

**Usage: git revert [commit id]**

The git revert command can be considered as an ‘undo’ command. However, it does not work as the traditional ‘undo’ operation. It figures out how to invert the changes introduced by the commit and appends a new commit with the resulting inverse content.

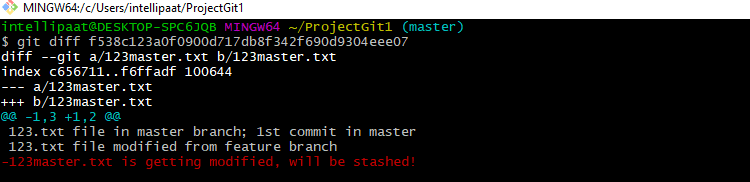
[](https://intellipaat.com/mediaFiles/2019/07/GItCommand23.png)

**13. git diff**

**Usage: git diff [commit-id-of-version-x] [commit-id-of-version-y]**

Diffing is a function that takes two input datasets and outputs the changes between them. The git diff command is a multi-use Git command which, when executed, runs a diff function on Git data sources. These data sources can be commits, branches, files, and more. The git diff command is often used along with the git status and git log commands to analyze the current state of our Git repository. We use **git log** to get the details of commit IDs.

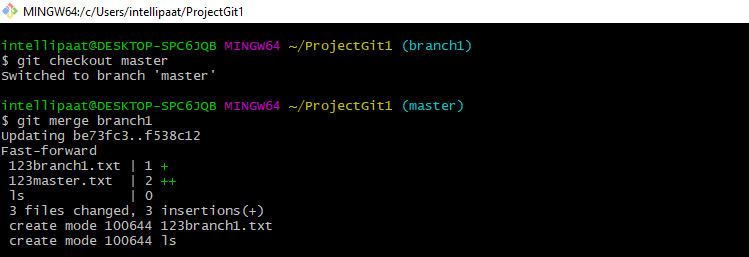
Let’s compare the working directory with the index as shown below:

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand24.png)

**14. git merge**

**Usage: git merge [another-file-name]**

This command will combine multiple sequences of commits into one unified history. In the most frequent use cases, git merge is used to combine two branches. The git merge command takes two commit pointers, usually the branch tips, and finds a common base commit between them. Once it finds a common base commit, it will create a commit sequence.

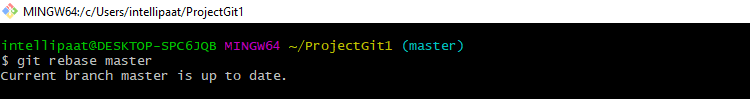
[](https://intellipaat.com/mediaFiles/2019/07/GItCommand25.png)

**15. 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.

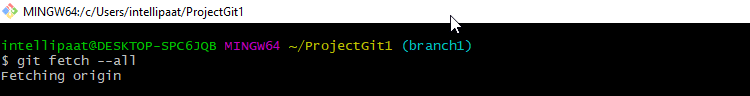
[](https://intellipaat.com/mediaFiles/2019/07/GItCommand26.png)  
Consider a situation where we have branched off from the master and have created a feature branch, but the master branch is still having more commits. We want to get the updated version of the master branch in our feature branch, keeping 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.

**16. git fetch**

**Usage: git fetch**

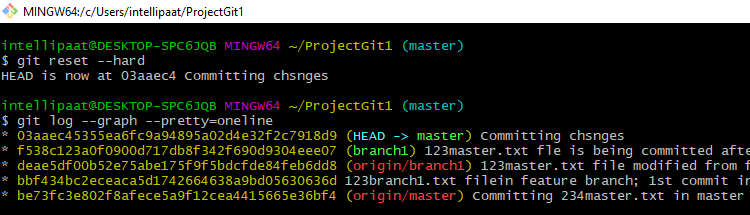
When we use the command git fetch, Git gathers any commit from the target branch that does not exist in our current branch and stores it in our local repository. However, **it does not merge it with our current branch**.

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand27.png)  
This is particularly useful when we need to keep our repository up to date but are working on something that might break if we updated our files. To integrate the commits into our master branch, we use merge. It fetches all of the branches from the repository. It also downloads all the required commits and files from another repository.

**17. git reset**

**Usage: git reset –hard [SOME-COMMIT]**

We use this command to **return** the *entire* working tree to the last committed state.

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand28.png)  
This will discard commits in a private branch or throw away the uncommitted changes!

Here, we have executed a ‘hard reset’ using the **–hard** option. Git displays the output indicating that the HEAD is pointing to the latest commit. Now, when we check the state of the repo with git status, Git will indicate that there are no pending changes (if any prior addition of a new file or modification of an existing file is done before using the ‘git reset –hard’ command). Our modifications to an existing file, if not committed, and the addition of a new file, if not staged, will be destroyed. It is critical to take note that this data loss cannot be undone.

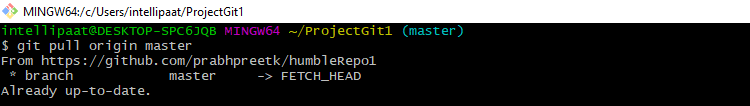
If we do**git reset –hard [SOME-COMMIT]**, then Git will**:**

* Make our current branch (typically master) back to point <SOME-COMMIT>
* Make the files in our working tree and the index (“staging area”) the same as the versions committed at <SOME-COMMIT>

**18. git pull**

**Usage: git pull origin master**

The git pull command first runs ‘git fetch’ which downloads the content from the specified remote repository and then immediately updates the local repo to match the content.

[](https://intellipaat.com/mediaFiles/2019/07/GItCommand30.png)