Git Commands

Git is a free, open-source distributed version control system tool designed to handle small to very large projects with speed and efficiency. It has steadily grown from just being a preferred skill to a must-have skill for multiple job roles today. Git has become an essential part of our everyday development process.

In this Git commands tutorial, let’s talk about the top 18 Git commands that are useful for working with Git.

Here are the top 18 Git commands list discussed in this tutorial:

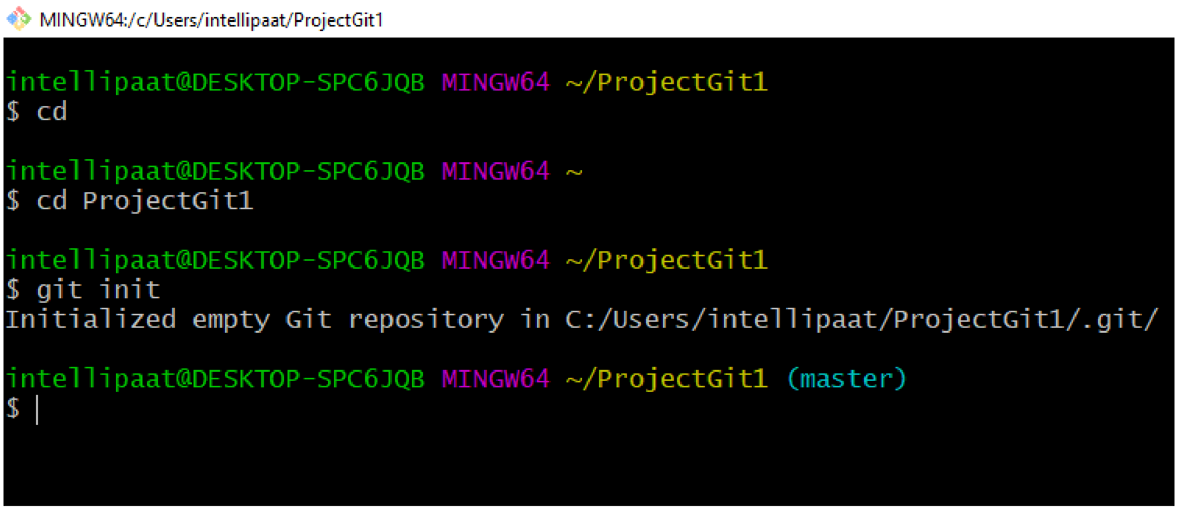
- git init
- git add
- git commit
- git status
- git remote
- git push
- git clone
- git branch
- git checkout
- git log
- git stash
- git revert
- git diff
- git merge
- git rebase
- git fetch
- git reset
- git pull

So, let’s get started!

1. git init

Usage: git init [repository name]

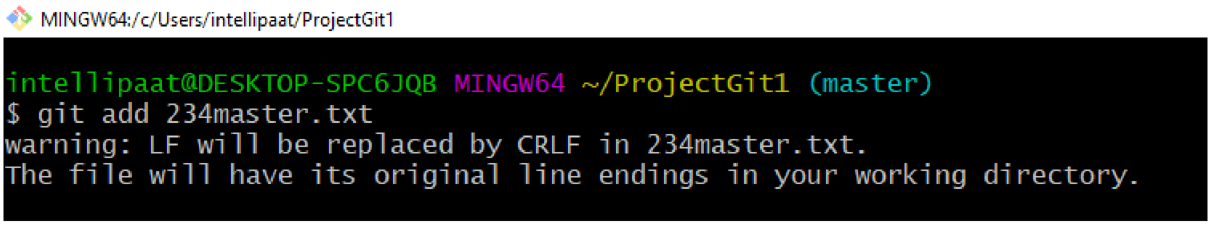
We have to navigate to our project directory and type the command **git init** to initialize a Git repository for our local project folder. Git will create a hidden **.git** directory and use it for keeping its files organized in other subdirectories.


```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1
$ cd

intellipaat@DESKTOP-SPC6JQB MINGW64 ~
$ cd ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1
$ git init
Initialized empty Git repository in C:/Users/intellipaat/ProjectGit1/.git/

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ |
```

2. git add

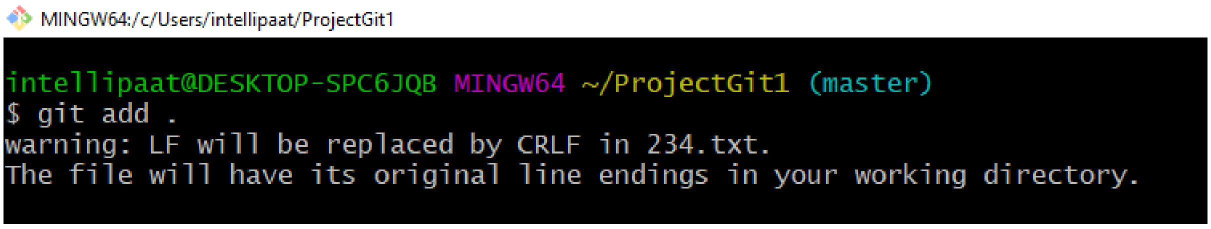
Usage (i): git add [file(s) name]

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.


```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git add 234master.txt
warning: LF will be replaced by CRLF in 234master.txt.
The file will have its original line endings in your working directory.
```

Usage (ii): git add . or git add *

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


```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git add .
warning: LF will be replaced by CRLF in 234.txt.
The file will have its original line endings in your working directory.
```

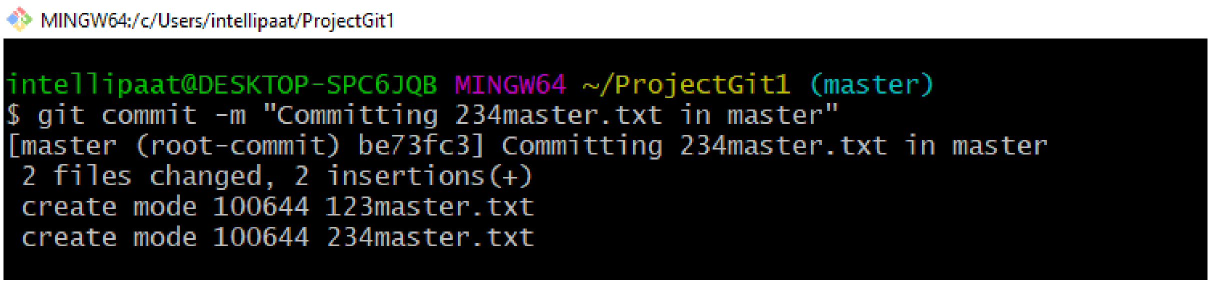
We can use this command as **git add -A** as well.

Note: We will have to commit our files after we add them to the staging area.

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.



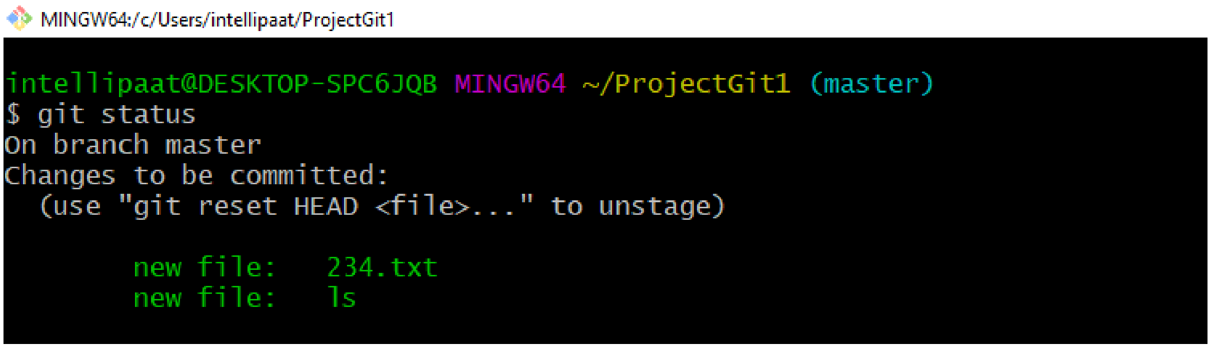
MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git commit -m "Committing 234master.txt in master"
[master (root-commit) be73fc3] Committing 234master.txt in master
 2 files changed, 2 insertions(+)
 create mode 100644 123master.txt
 create mode 100644 234master.txt
```

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.



MINGW64:/c/Users/intellipaat/ProjectGit1

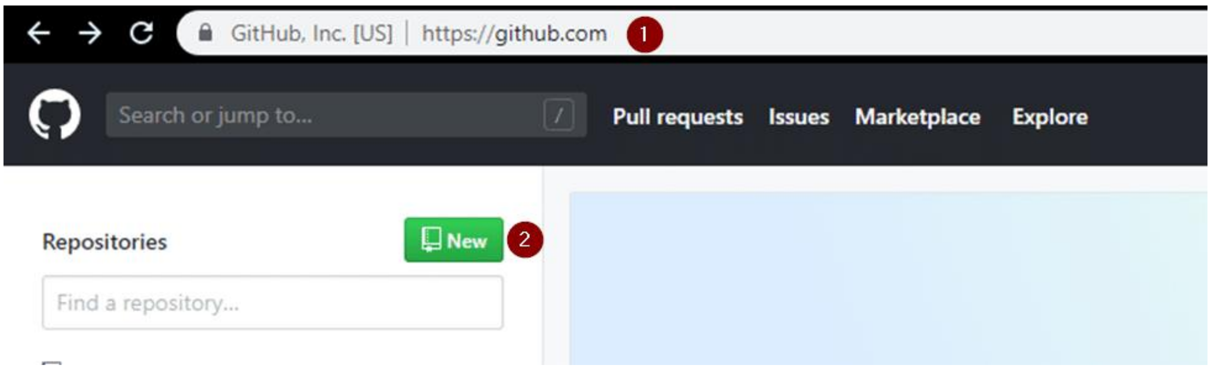
```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   234.txt
        new file:   ls
```

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**


GitHub, Inc. [US] | https://github.com 1

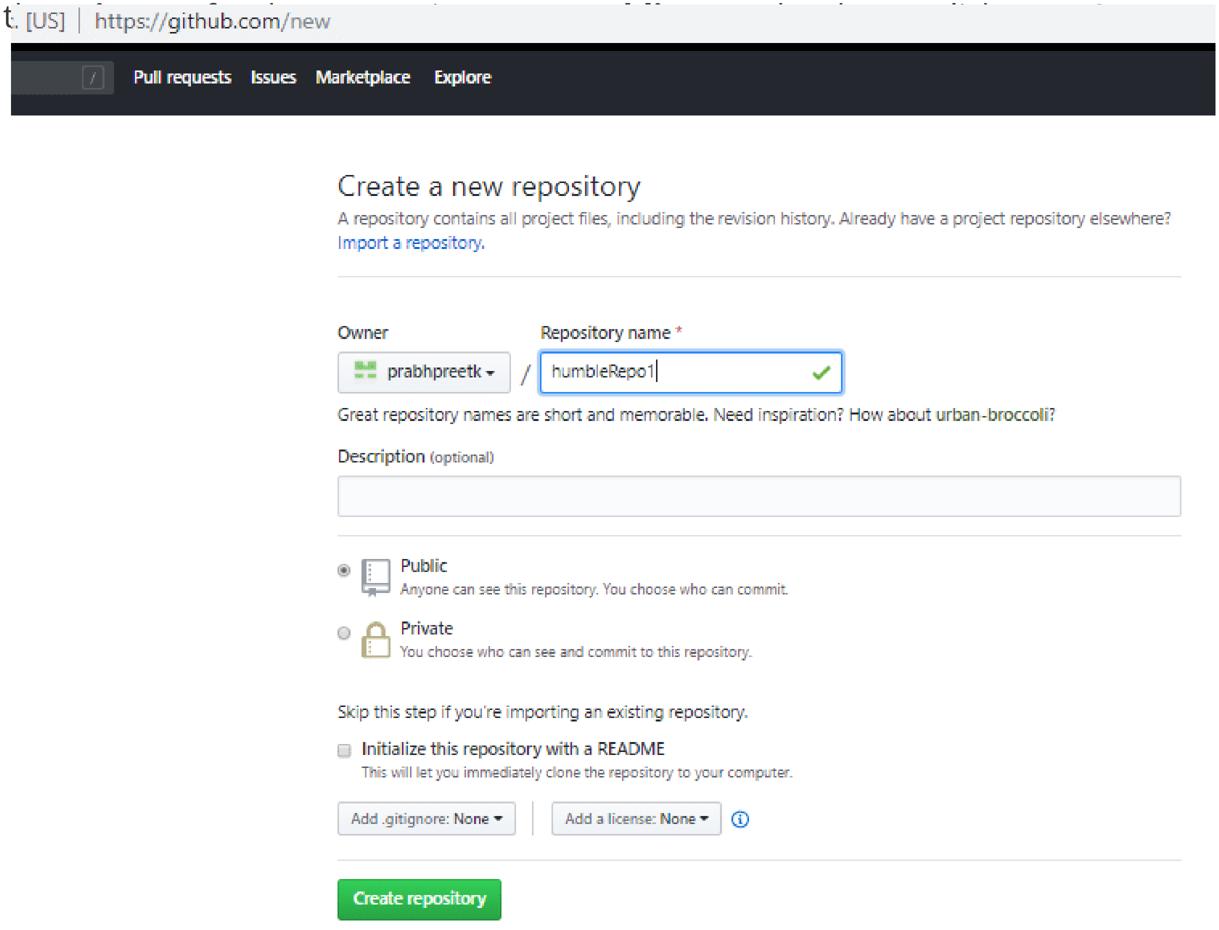
Search or jump to... / Pull requests Issues Marketplace Explore

Repositories New 2

Find a repository...

Step 2: Now, we have to create a new repository. Provide a **name** to our **repository**, select

t **repository**


[US] | https://github.com/new

Pull requests Issues Marketplace Explore

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner: prabhpreetk / Repository name *: humbleRepo1

Great repository names are short and memorable. Need inspiration? How about urban-broccoli?

Description (optional)

● Public — Anyone can see this repository. You choose who can commit.

○ Private — You choose who can see and commit to this repository.

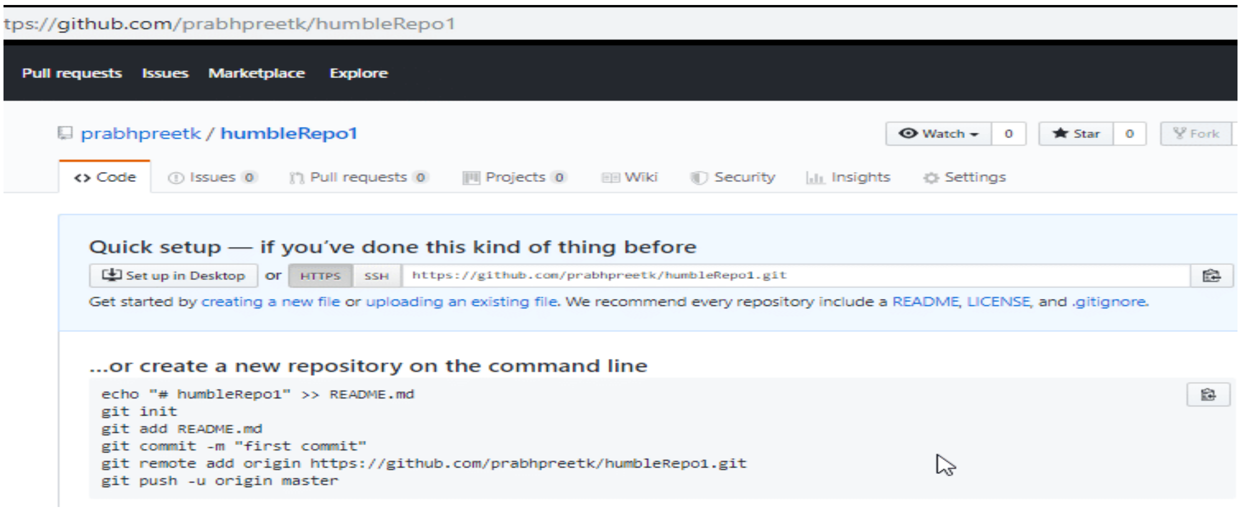
Skip this step if you're importing an existing repository.

☐ Initialize this repository with a README — This will let you immediately clone the repository to your computer.

Add .gitignore: None | Add a license: None

Create repository

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


tps://github.com/prabhpreetk/humbleRepo1

Pull requests Issues Marketplace Explore

prabhpreetk / humbleRepo1 — Watch 0 | Star 0 | Fork

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Security | Insights | Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/prabhpreetk/humbleRepo1.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

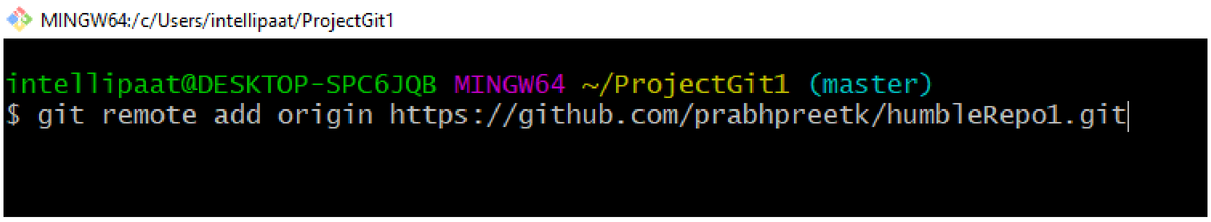
```
echo "# humbleRepo1" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/prabhpreetk/humbleRepo1.git
git push -u origin master
```

•

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.

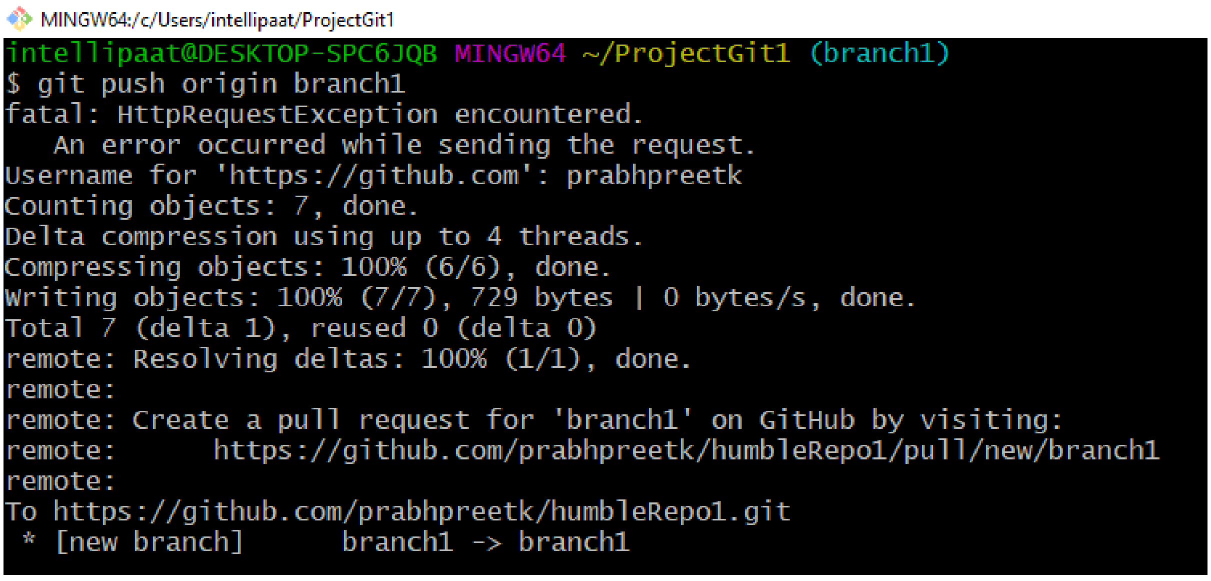


```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git remote add origin https://github.com/prabhpreetk/humbleRepo1.git
```

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.



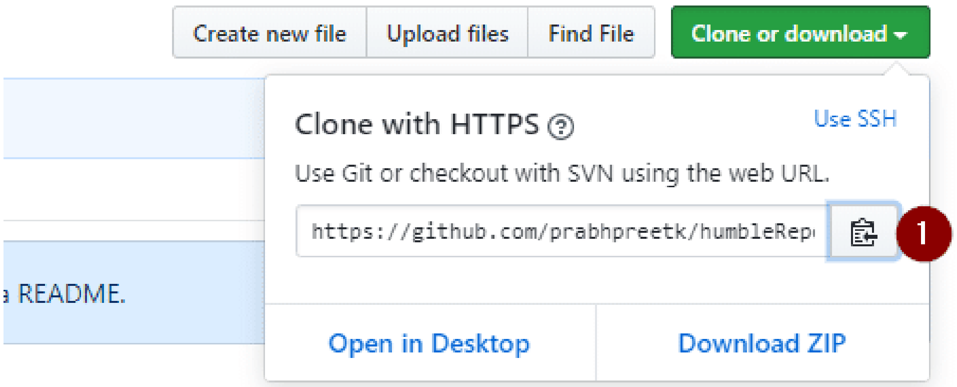
```
MINGW64:/c/Users/intellipaat/ProjectGit1
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git push origin branch1
fatal: HttpRequestException encountered.
   An error occurred while sending the request.
Username for 'https://github.com': prabhpreetk
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 729 bytes | 0 bytes/s, done.
Total 7 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'branch1' on GitHub by visiting:
remote:      https://github.com/prabhpreetk/humbleRepo1/pull/new/branch1
remote:
To https://github.com/prabhpreetk/humbleRepo1.git
 * [new branch]      branch1 -> branch1
```

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.

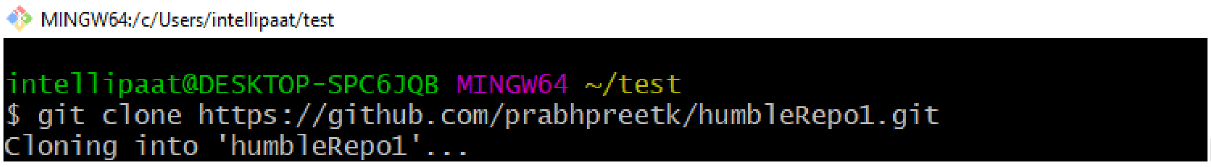
(1) The below screenshot shows from where to copy the link:



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:



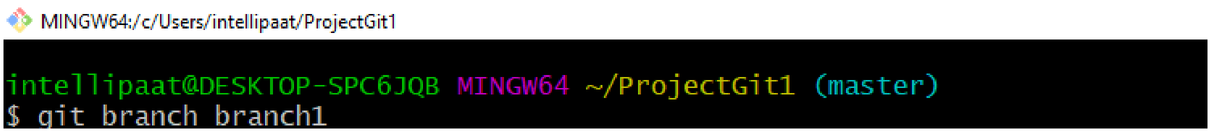
```
MINGW64:/c/Users/intellipaat/test

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/test
$ git clone https://github.com/prabhpreetk/humbleRepo1.git
Cloning into 'humbleRepo1'...
```

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

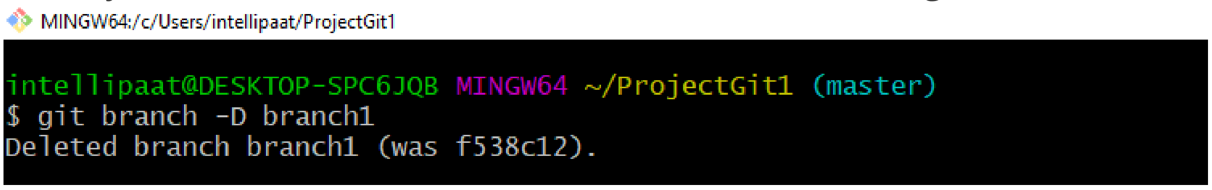
Usage (i): git branch [name-of-the-branch]

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:


```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git branch branch1
```

Usage (ii): git branch -D [name -of-the-branch]

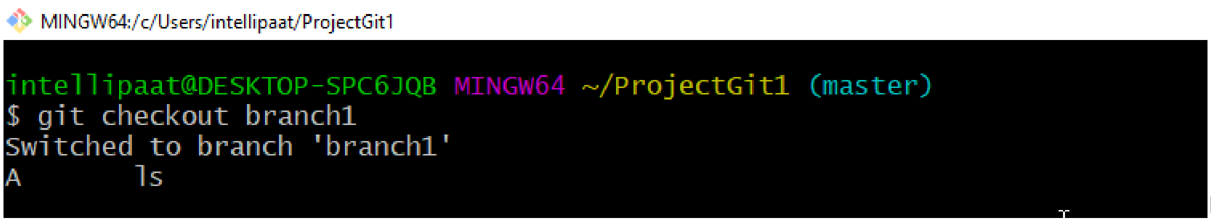
Similarly, to delete a branch, we use the **git branch -D** command:


```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git branch -D branch1
Deleted branch branch1 (was f538c12).
```

9. git checkout

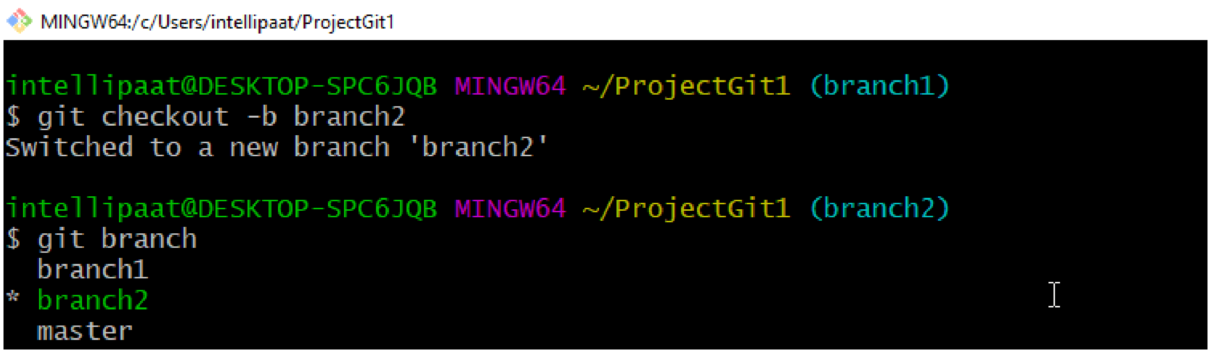
Usage (i): git checkout [name-of-the-new-branch]

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


```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git checkout branch1
Switched to branch 'branch1'
A       ls
```

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:

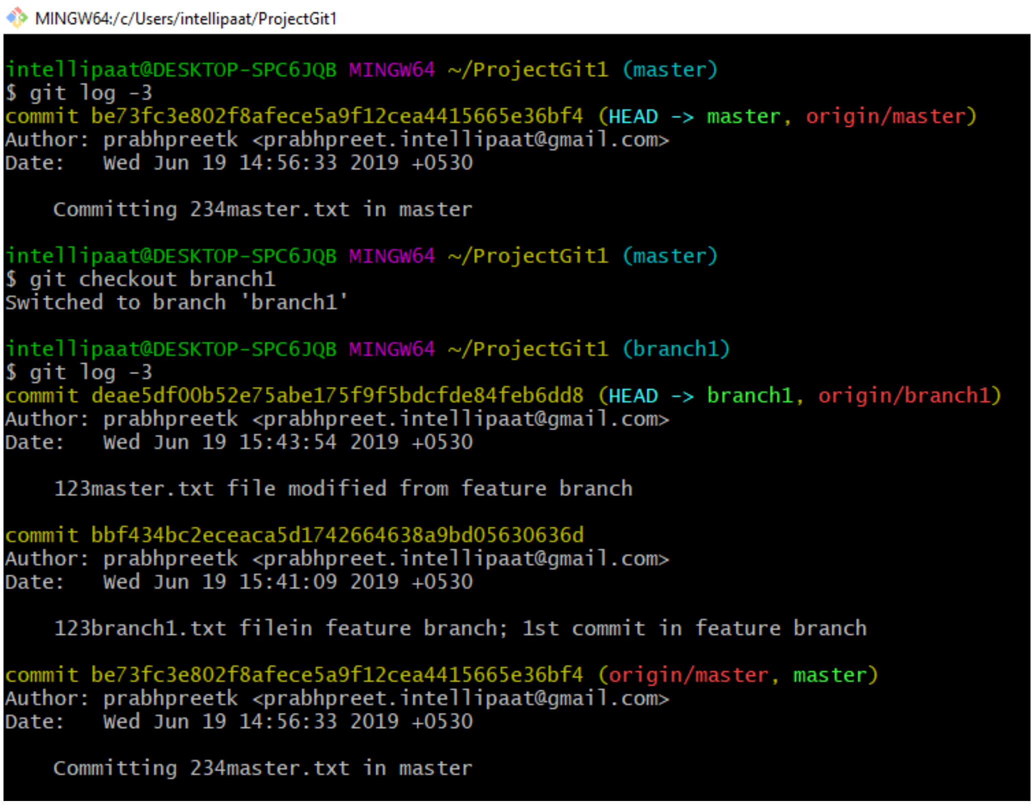

```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git checkout -b branch2
Switched to a new branch 'branch2'

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch2)
$ git branch
  branch1
* branch2
  master
```

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.

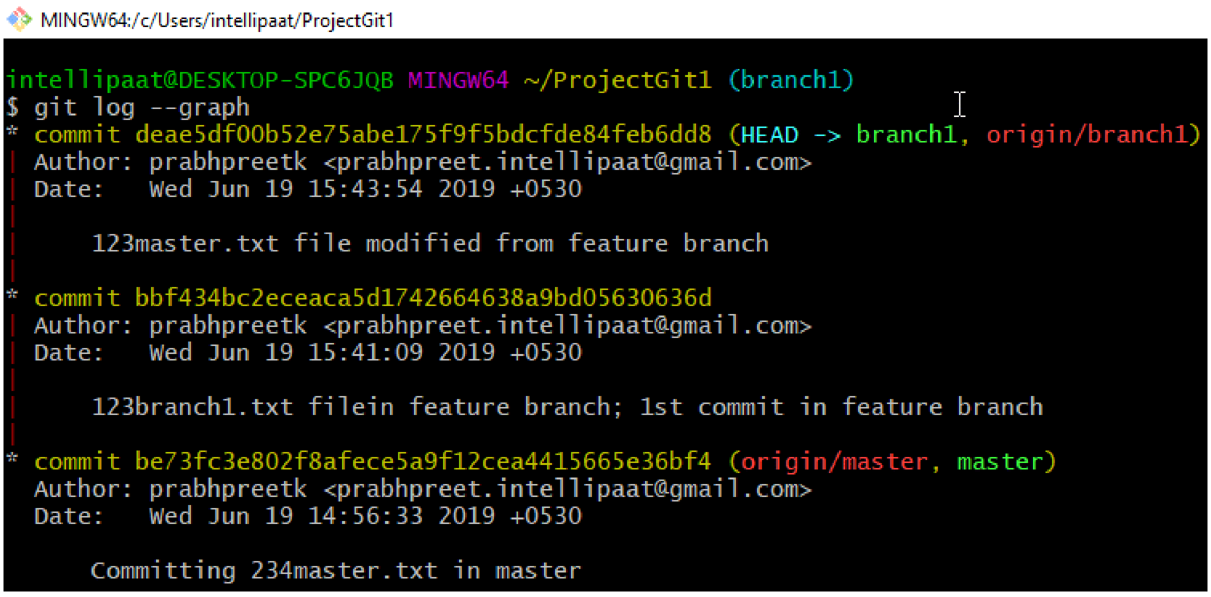
Note: It will show the log of the branch we are in. We can check the last three logs by giving the command: **git log -3**



MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git log -3
commit be73fc3e802f8afece5a9f12cea4415665e36bf4 (HEAD -> master, origin/master)
Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
Date:   Wed Jun 19 14:56:33 2019 +0530

    Committing 234master.txt in master

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git checkout branch1
Switched to branch 'branch1'

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git log -3
commit deae5df00b52e75abe175f9f5bdcfde84feb6dd8 (HEAD -> branch1, origin/branch1)
Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
Date:   Wed Jun 19 15:43:54 2019 +0530

    123master.txt file modified from feature branch

commit bbf434bc2eceaca5d1742664638a9bd05630636d
Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
Date:   Wed Jun 19 15:41:09 2019 +0530

    123branch1.txt filein feature branch; 1st commit in feature branch

commit be73fc3e802f8afece5a9f12cea4415665e36bf4 (origin/master, master)
Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
Date:   Wed Jun 19 14:56:33 2019 +0530

    Committing 234master.txt in master
```

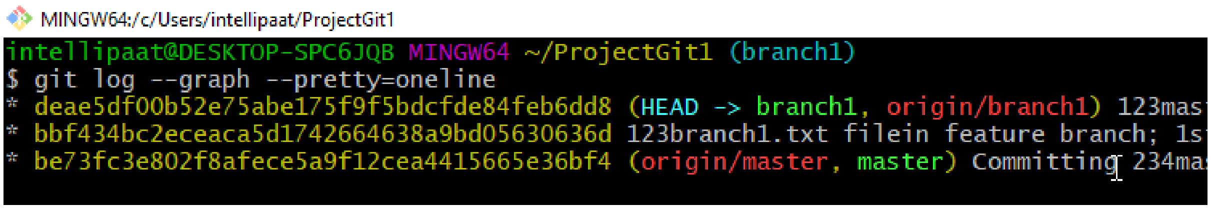
Usage (ii): git log –graph



MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git log --graph
* commit deae5df00b52e75abe175f9f5bdcfde84feb6dd8 (HEAD -> branch1, origin/branch1)
| Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
| Date:   Wed Jun 19 15:43:54 2019 +0530
|
|     123master.txt file modified from feature branch
|
* commit bbf434bc2eceaca5d1742664638a9bd05630636d
| Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
| Date:   Wed Jun 19 15:41:09 2019 +0530
|
|     123branch1.txt filein feature branch; 1st commit in feature branch
|
* commit be73fc3e802f8afece5a9f12cea4415665e36bf4 (origin/master, master)
  Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
  Date:   Wed Jun 19 14:56:33 2019 +0530

      Committing 234master.txt in master
```

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

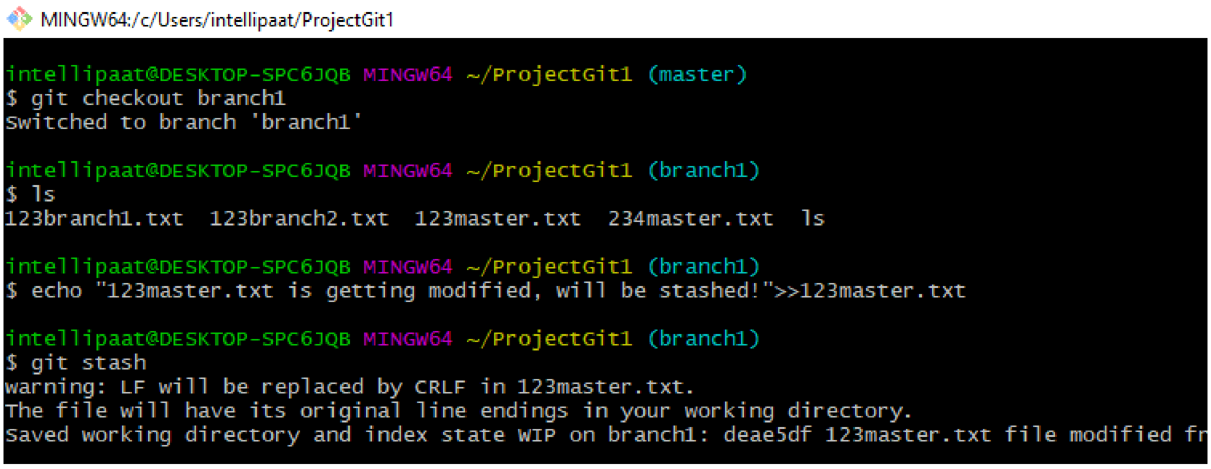


```
MINGW64:/c/Users/intellipaat/ProjectGit1
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git log --graph --pretty=oneline
* deae5df00b52e75abe175f9f5bdcfde84feb6dd8 (HEAD -> branch1, origin/branch1) 123mas
* bbf434bc2eceaca5d1742664638a9bd05630636d 123branch1.txt filein feature branch; 1s
* be73fc3e802f8afece5a9f12cea4415665e36bf4 (origin/master, master) Committing 234ma
```

11. git stash

Usage (i): git stash

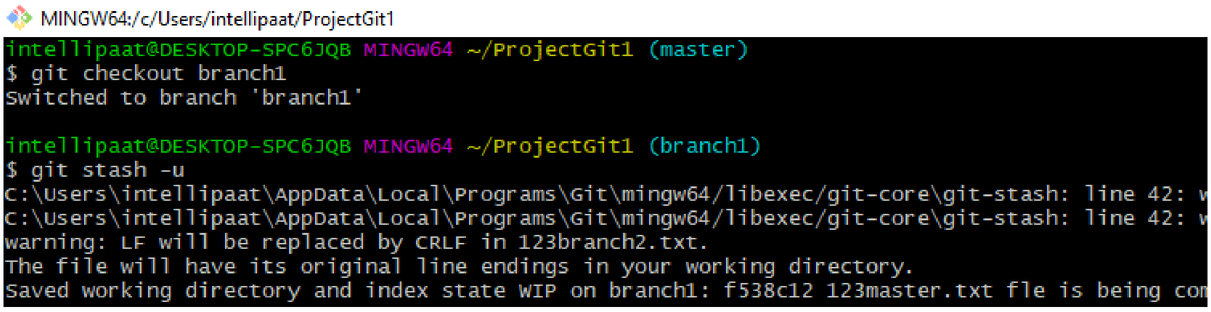
This command can be used when we want to save our work without staging or committing the code to our Git repository and want to switch between branches.



```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git checkout branch1
Switched to branch 'branch1'

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ ls
123branch1.txt  123branch2.txt  123master.txt  234master.txt  ls

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ echo "123master.txt is getting modified, will be stashed!">>123master.txt

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git stash
warning: LF will be replaced by CRLF in 123master.txt.
The file will have its original line endings in your working directory.
Saved working directory and index state WIP on branch1: deae5df 123master.txt file modified fr
```

Usage (ii): git stash -u

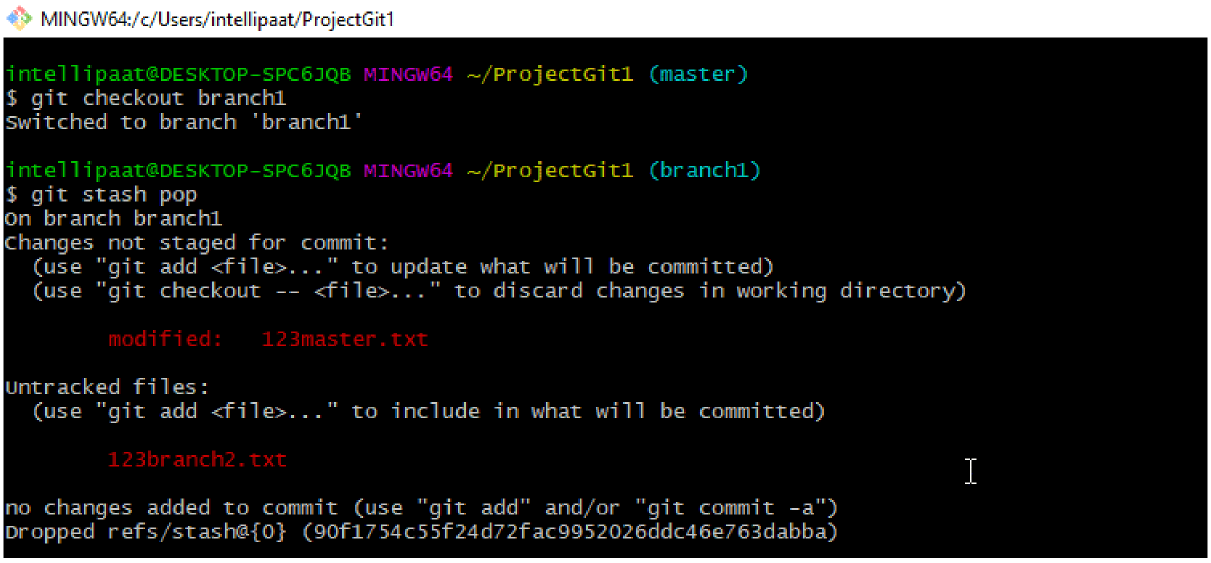
This command is used when we want to stash the untracked files.



```
MINGW64:/c/Users/intellipaat/ProjectGit1
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git checkout branch1
Switched to branch 'branch1'

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git stash -u
C:\Users\intellipaat\AppData\Local\Programs\Git\mingw64/libexec/git-core\git-stash: line 42: w
C:\Users\intellipaat\AppData\Local\Programs\Git\mingw64/libexec/git-core\git-stash: line 42: w
warning: LF will be replaced by CRLF in 123branch2.txt.
The file will have its original line endings in your working directory.
Saved working directory and index state WIP on branch1: f538c12 123master.txt fle is being com
```

Usage (iii): git stash pop

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

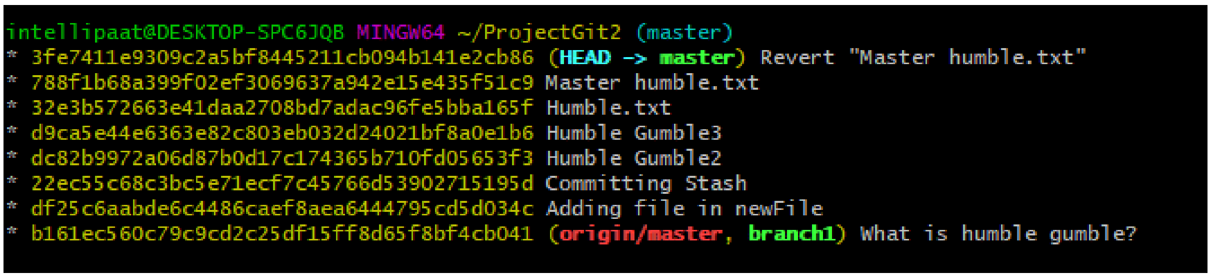

MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git checkout branch1
Switched to branch 'branch1'

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git stash pop
On branch branch1
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   123master.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        123branch2.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (90f1754c55f24d72fac9952026ddc46e763dabba)
```

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.



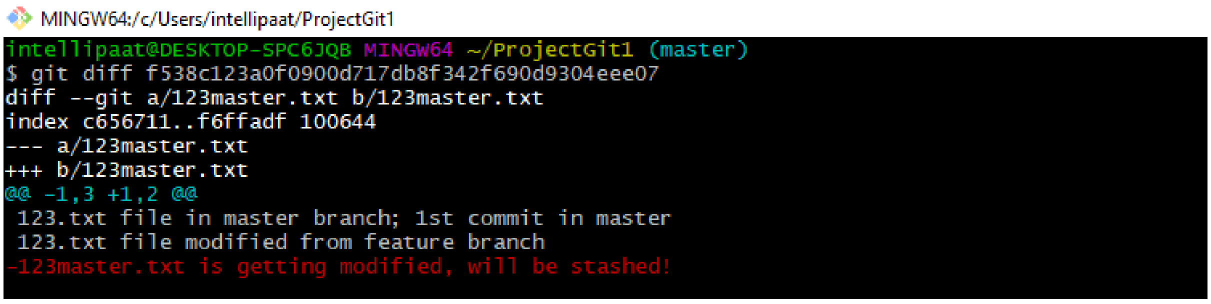
```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit2 (master)
* 3fe7411e9309c2a5bf8445211cb094b141e2cb86 (HEAD -> master) Revert "Master humble.txt"
* 788f1b68a399f02ef3069637a942e15e435f51c9 Master humble.txt
* 32e3b572663e41daa2708bd7adac96fe5bba165f Humble.txt
* d9ca5e44e6363e82c803eb032d24021bf8a0e1b6 Humble Gumble3
* dc82b9972a06d87b0d17c174365b710fd05653f3 Humble Gumble2
* 22ec55c68c3bc5e71ecf7c45766d53902715195d Committing Stash
* df25c6aabde6c4486caef8aea6444795cd5d034c Adding file in newFile
* b161ec560c79c9cd2c25df15ff8d65f8bf4cb041 (origin/master, branch1) What is humble gumble?
```

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:

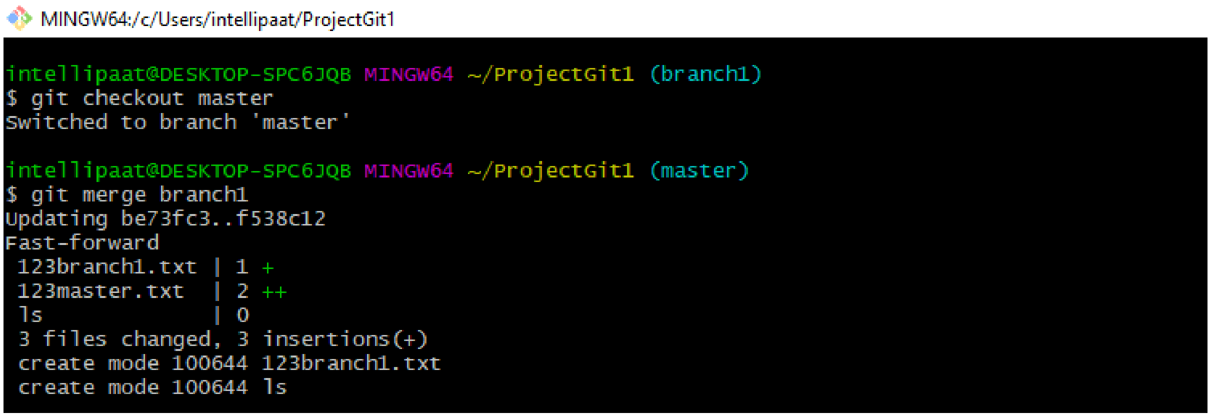

MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git diff f538c123a0f0900d717db8f342f690d9304eee07
diff --git a/123master.txt b/123master.txt
index c656711..f6ffadf 100644
--- a/123master.txt
+++ b/123master.txt
@@ -1,3 +1,2 @@
 123.txt file in master branch; 1st commit in master
 123.txt file modified from feature branch
-123master.txt is getting modified, will be stashed!
```

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.


MINGW64:/c/Users/intellipaat/ProjectGit1

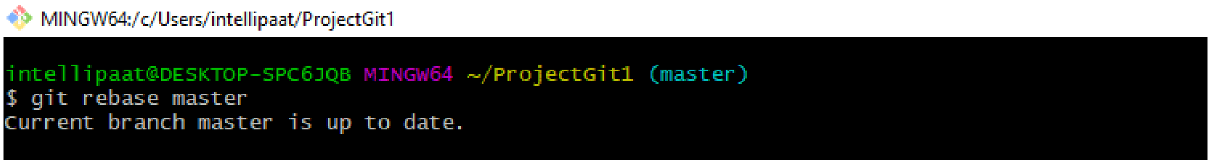
```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git checkout master
Switched to branch 'master'

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git merge branch1
Updating be73fc3..f538c12
Fast-forward
 123branch1.txt | 1 +
 123master.txt  | 2 ++
 ls             | 0
 3 files changed, 3 insertions(+)
 create mode 100644 123branch1.txt
 create mode 100644 ls
```

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.



MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git rebase master
Current branch master is up to date.
```

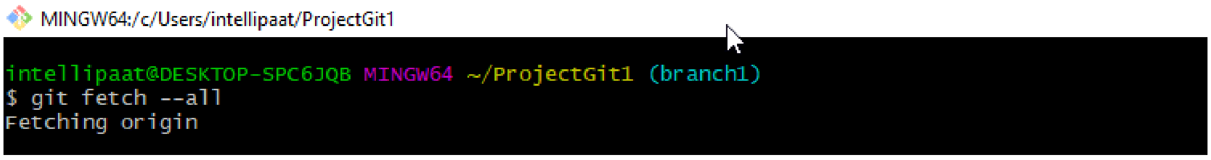
Consider a situation where we have branched off from the master and have created a feature branch, but the master branch is still having more commits. We want to get the updated version of the master branch in our feature branch, keeping our branch's history clean, so that it appears as if we are working 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**.



MINGW64:/c/Users/intellipaat/ProjectGit1

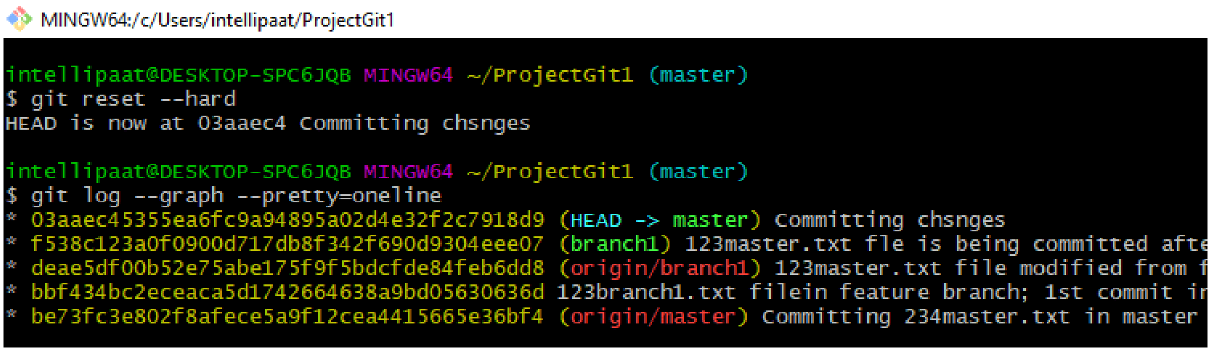
```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git fetch --all
Fetching origin
```

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.



MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git reset --hard
HEAD is now at 03aaec4 Committing chsnges

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git log --graph --pretty=oneline
* 03aaec45355ea6fc9a94895a02d4e32f2c7918d9 (HEAD -> master) Committing chsnges
* f538c123a0f0900d717db8f342f690d9304eee07 (branch1) 123master.txt fle is being committed afte
* deae5df00b52e75abe175f9f5bdcfde84feb6dd8 (origin/branch1) 123master.txt file modified from f
* bbf434bc2eceaca5d1742664638a9bd05630636d 123branch1.txt filein feature branch; 1st commit ir
* be73fc3e802f8afece5a9f12cea4415665e36bf4 (origin/master) Committing 234master.txt in master
```

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.

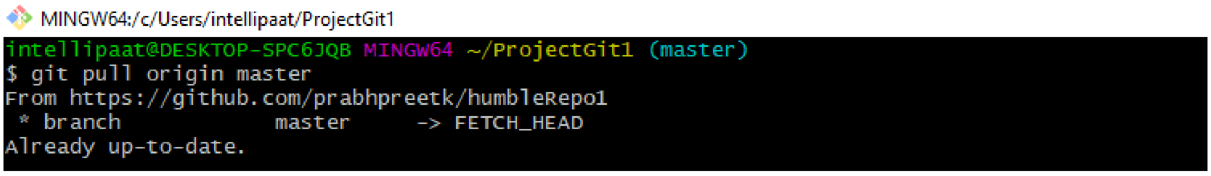
If we do **git reset –hard [SOME-COMMIT]**, then Git will**:**

- 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.

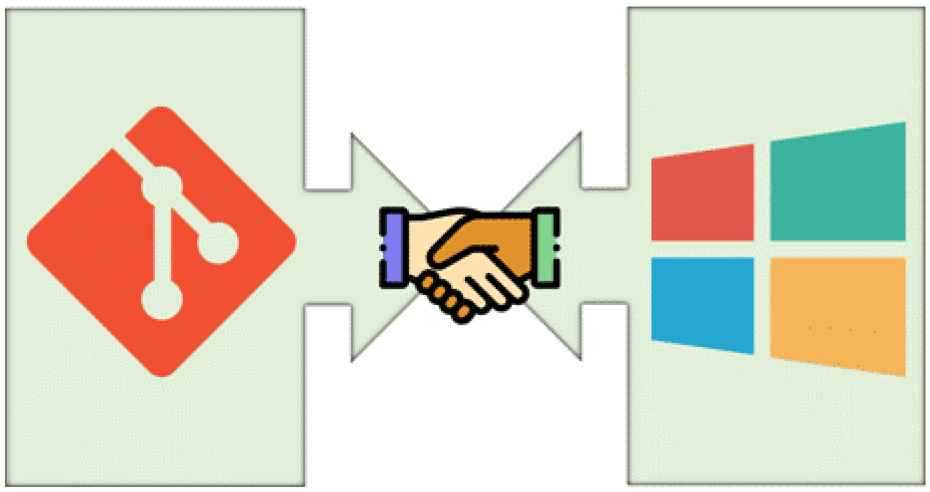


MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git pull origin master
From https://github.com/prabhpreetk/humbleRepo1
 * branch            master     -> FETCH_HEAD
Already up-to-date.
```

Git and Its Popularity

Git is one of the most important version control systems that has experienced a significant growth rate and popularity over the years. Git plays a key role in both developers' and non-developers world. If we are part of the software developers' world, it is expected that we know Git. So, having a good grasp of common Git commands is very beneficial for us to get into a lucrative career.



Even Microsoft has started leveraging Git very recently. This opens up more opportunities for the developers' community. Hence, it is the right time to learn Git.

Hopefully, you have understood Git and got answers to the questions: 'What are Git commands?' and 'What do Git commands do?