GIT COMMANDS

GIT INIT

The git init command is the first command that you will run on Git. The git init command is used to create a new blank repository. It is used to make an existing project as a Git project. Several Git commands run inside the repository, but init command can be run outside of the repository.

The git init command creates a .git subdirectory in the current working directory. This newly created subdirectory contains all of the necessary metadata. These metadata can be categorized into objects, refs, and temp files. It also initializes a HEAD pointer for the master branch of the repository.

Creating the first repository

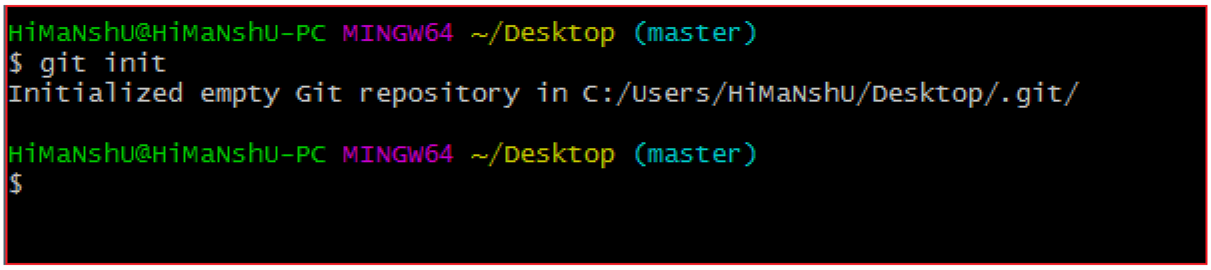
Git version control system allows you to share projects among developers. For learning Git, it is essential to understand that how can we create a project on Git. A repository is a directory that contains all the project-related data. There can also be more than one project on a single repository.

We can create a repository for blank and existing projects. Let's understand how to create a repository.

To create a blank repository, open command line on your desired directory and run the init command as follows:

```
$ git init
```

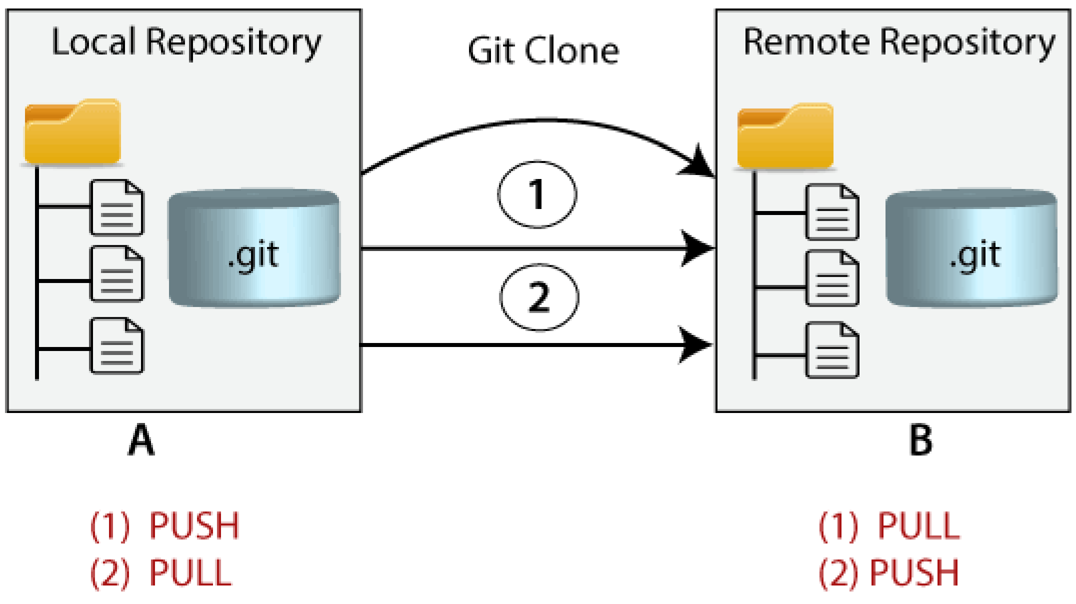
The above command will create an empty .git repository. Suppose we want to make a git repository on our desktop. To do so, open Git Bash on the desktop and run the above command. Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop (master)
$ git init
Initialized empty Git repository in C:/Users/HiMaNshU/Desktop/.git/

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop (master)
$
```

GIT CLONE

In Git, cloning is the act of making a copy of any target repository. The target repository can be remote or local. You can clone your repository from the remote repository to create a local copy on your system. Also, you can sync between the two locations.



Git Clone Command

The **git clone** is a command-line utility which is used to make a local copy of a remote repository. It accesses the repository through a remote URL.

Usually, the original repository is located on a remote server, often from a Git service like GitHub, Bitbucket, or GitLab. The remote repository URL is referred to the **origin**.

```
$ git clone <repository URL>
```

Git Clone Repository

Suppose, you want to clone a repository from GitHub, or have an existing repository owned by any other user you would like to contribute. Steps to clone a repository are as follows:

Step 1:

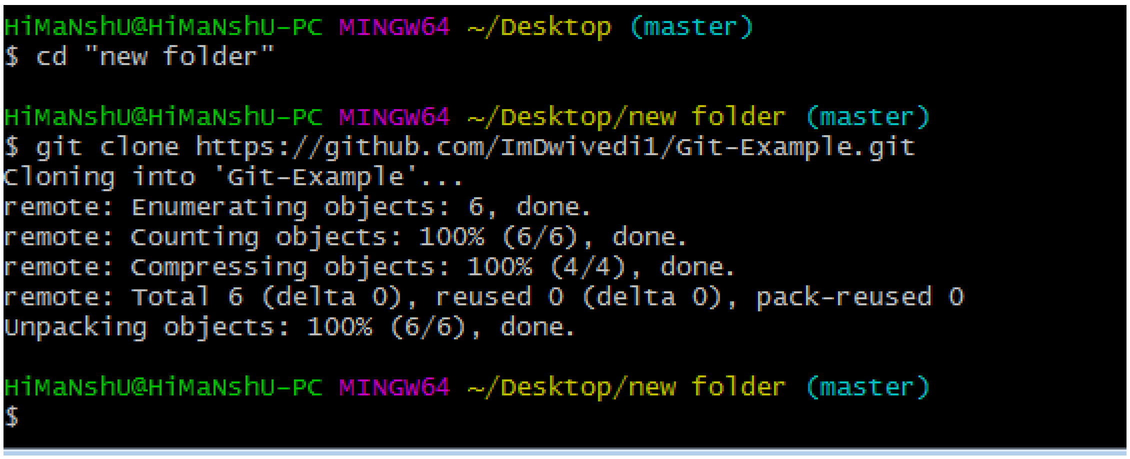
Open GitHub and navigate to the main page of the repository.

Step 2:

Under the repository name, click on **Clone or download**.

Step 3:

Select the **Clone with HTTPs section** and **copy the clone URL** for the repository. For the empty repository, you can copy the repository page URL from your browser and skip to next step.



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop (master)
$ cd "new folder"

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/new folder (master)
$ git clone https://github.com/ImDwivedi1/Git-Example.git
Cloning into 'Git-Example'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/new folder (master)
$
```

GIT ADD FILE NAME

The git add command is used to add file contents to the Index (Staging Area).This command updates the current content of the working tree to the staging area. It also prepares the staged content for the next commit. Every time we add or update any file in our project, it is required to forward updates to the staging area.

The git add command is a core part of Git technology. It typically adds one file at a time, but there some options are available that can add more than one file at once.

The "index" contains a snapshot of the working tree data. This snapshot will be forwarded for the next commit.

The git add command can be run many times before making a commit. These all add operations can be put under one commit. The add command adds the files that are specified on command line.

The git add command does not add the .gitignore file by default. In fact, we can ignore the files by this command.

Let's understand how to add files on Git?

Git add files

Git add command is a straight forward command. It adds files to the staging area. We can add single or multiple files at once in the staging area. It will be run as:

$ git add <File name>

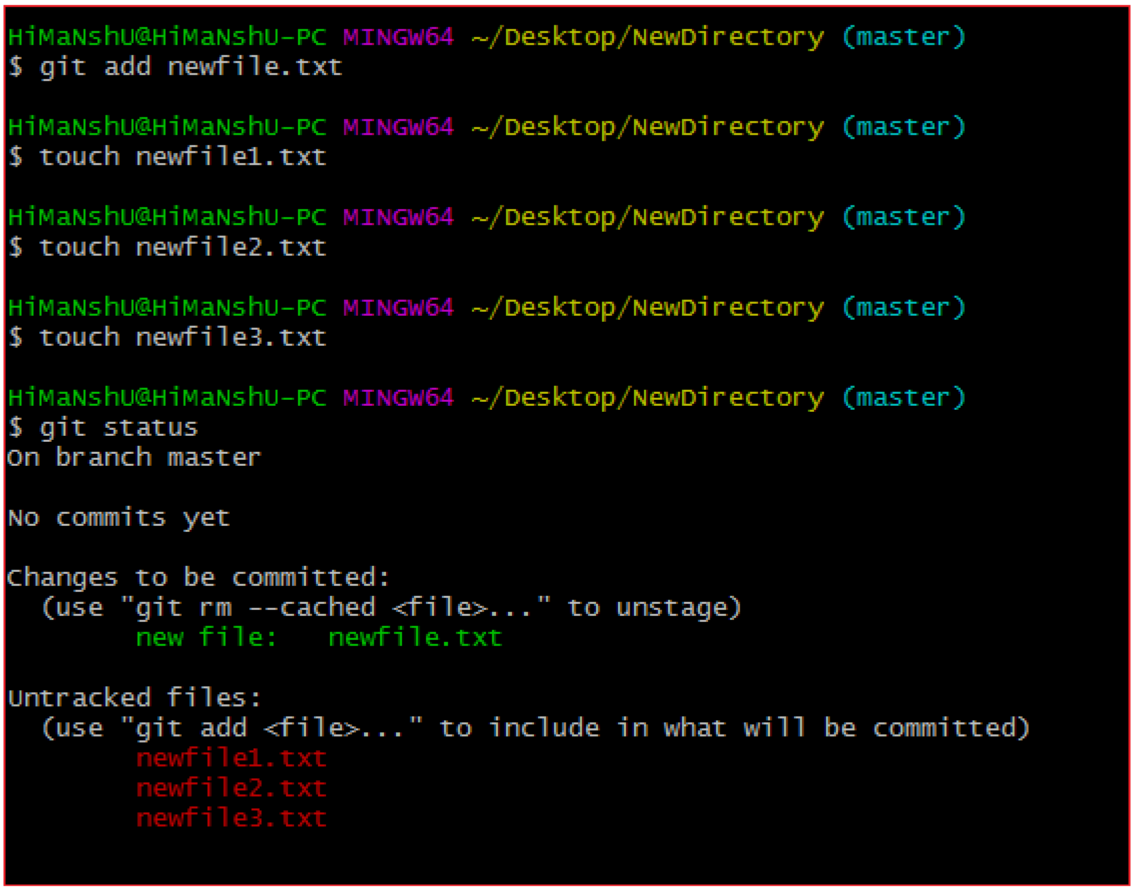
Git Add All

We can add more than one files in Git, but we have to run the add command repeatedly. Git facilitates us with a unique option of the add command by which we can add all the available files at once. To add all the files from the repository, run the add command with **-A** option. We can use '.' Instead of **-A** option. This command will stage all the files at a time. It will run as follows:

$ git add -A Or $ git add .

The above command will add all the files available in the repository. Consider the below scenario:

We can either create four new files, or we can copy it, and then we add all these files at once. Consider the below output:

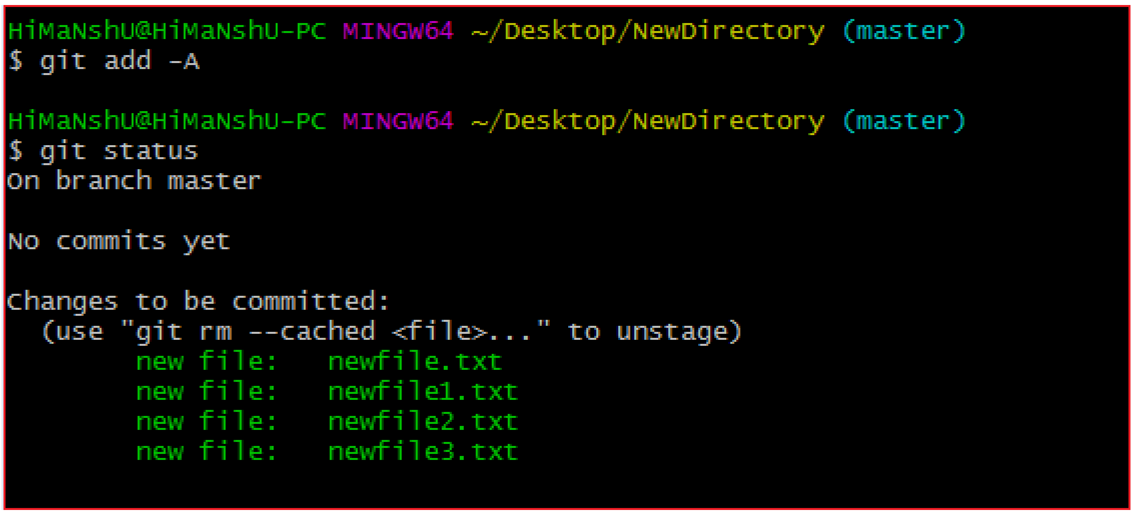


```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git add newfile.txt

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ touch newfile1.txt

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ touch newfile2.txt

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ touch newfile3.txt

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   newfile.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        newfile1.txt
        newfile2.txt
        newfile3.txt
```

In the above output, all the files are displaying as untracked files by Git. To track all of these files at once, run the below command:

$ git add -A

The above command will add all the files to the staging area. Remember, the **-A** option is case sensitive. Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git add -A

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   newfile.txt
        new file:   newfile1.txt
        new file:   newfile2.txt
        new file:   newfile3.txt
```

In the above output, all the files have been added. The status of all files is displaying as staged.

GIT STATUS

The git status command is used to display the state of the repository and staging area. It allows us to see the tracked, untracked files and changes. This command will not show any commit records or information.

Mostly, it is used to display the state between **Git Add** and **Git commit** command. We can check whether the changes and files are tracked or not.

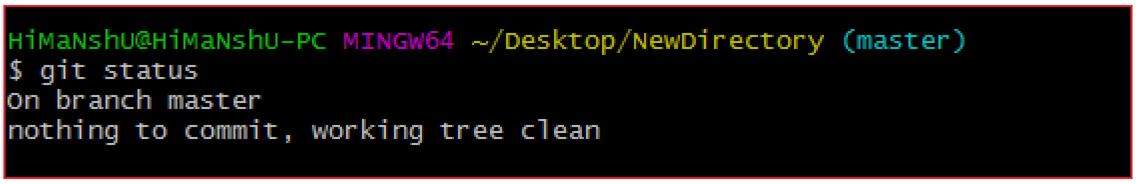
Let's understand the different states of status command.

Status when Working Tree is cleaned

Before starting with git status command, let's see how the git status looks like when there are no changes made. To check the status, open the git bash, and run the status command on your desired directory. It will run as follows:

```
$ git status
```

Output:

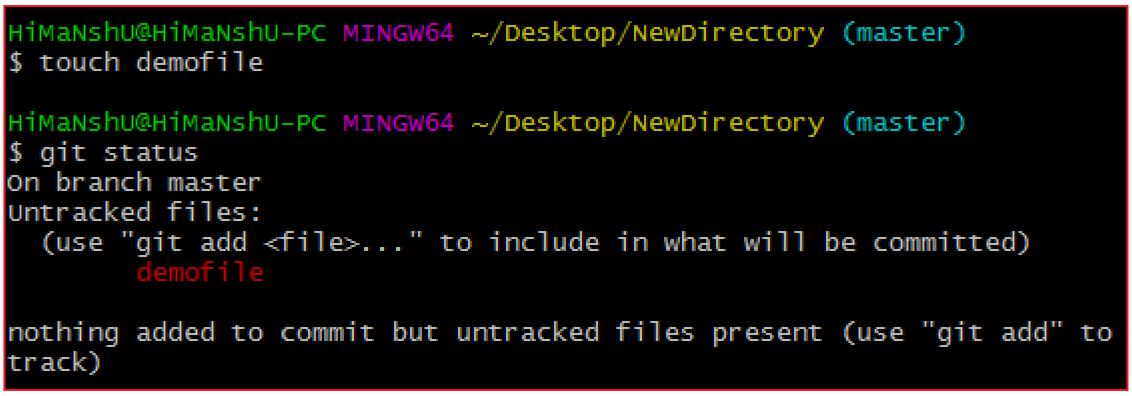


```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Since there is nothing to track or untrack in the working tree, so the output is showing as the **working tree is clean**.

Status when a new file is created

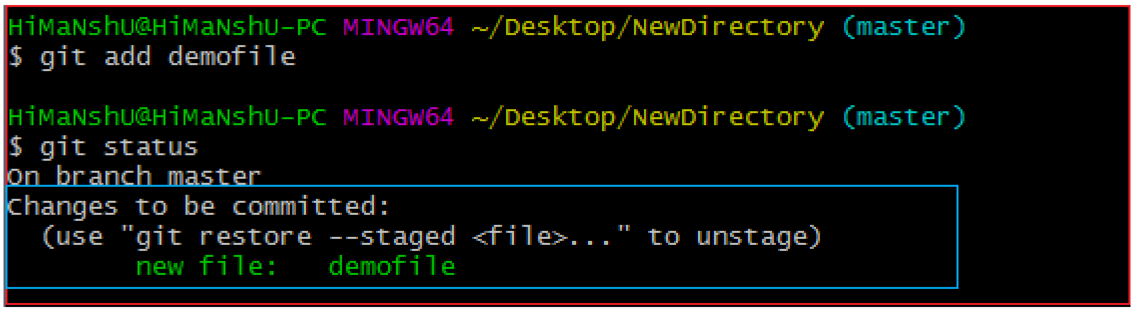
When we create a file in the repository, the state of the repository changes. Let's create a file using the **touch** command. Now, check the status using the status command. Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ touch demofile

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        demofile

nothing added to commit but untracked files present (use "git add" to
track)
```

As we can see from the above output, the status is showing as "**nothing added to commit but untracked files present (use "git add" to track)**". The status command also displays the suggestions. As in the above output, it is suggesting to use the add command to track the file.

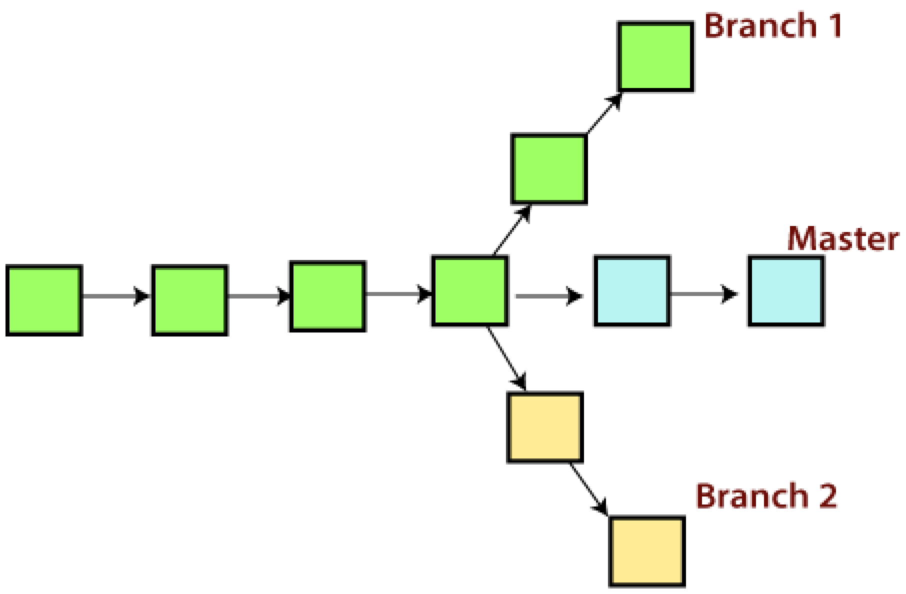
Let's track the file and will see the status after adding a file to the repository. To track the file, run the add command. Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git add demofile

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   demofile
```

GIT BRACH

A branch is a version of the repository that diverges from the main working project. It is a feature available in most modern version control systems. A Git project can have more than one branch. These branches are a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug, you spawn a new branch to summarize your changes. So, it is complex to merge the unstable code with the main code base and also facilitates you to clean up your future history before merging with the main branch.



Git Master Branch

The master branch is a default branch in Git. It is instantiated when first commit made on the project. When you make the first commit, you're given a master branch to the starting commit point. When you start making a commit, then master branch pointer automatically moves forward. A repository can have only one master branch.

Master branch is the branch in which all the changes eventually get merged back. It can be called as an official working version of your project.

Operations on Branches

We can perform various operations on Git branches. The **git branch command** allows you to **create**, **list**, **rename** and **delete** branches. Many operations on branches are applied by git checkout and git merge command. So, the git branch is tightly integrated with the **git checkout** and **git merge commands**.

The Operations that can be performed on a branch:

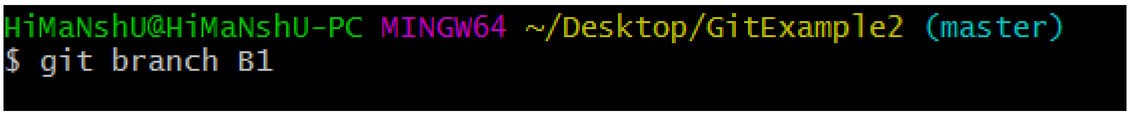
Create Branch

You can create a new branch with the help of the **git branch** command. This command will be used as:

Syntax:

```
1. $ git branch <branch name>
```

Output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git branch B1
```

This command will create the **branch B1** locally in Git directory.

List Branch

You can List all of the available branches in your repository by using the following command.

Either we can use **git branch - list** or **git branch** command to list the available branches in the repository.

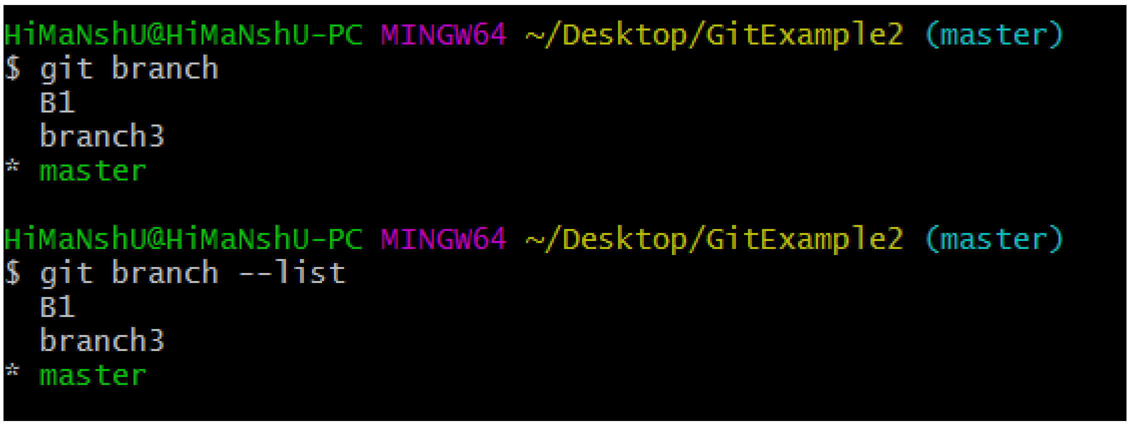
Syntax:

1. $ git branch --list

or

1. $ git branch

Output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git branch
  B1
  branch3
* master

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git branch --list
  B1
  branch3
* master
```

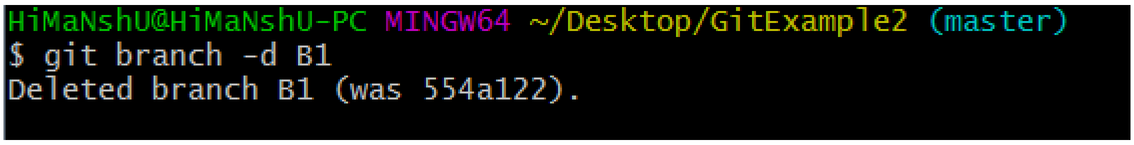
Here, both commands are listing the available branches in the repository. The symbol * is representing currently active branch.

Delete Branch

You can delete the specified branch. It is a safe operation. In this command, Git prevents you from deleting the branch if it has unmerged changes. Below is the command to do this.

1. $ git branch -d<**branch** name>

Output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git branch -d B1
Deleted branch B1 (was 554a122).
```

This command will delete the existing branch B1 from the repository.

The **git branch d** command can be used in two formats. Another format of this command is **git branch D**. The '**git branch D**' command is used to delete the specified branch.

$ git branch -D <**branch** name>

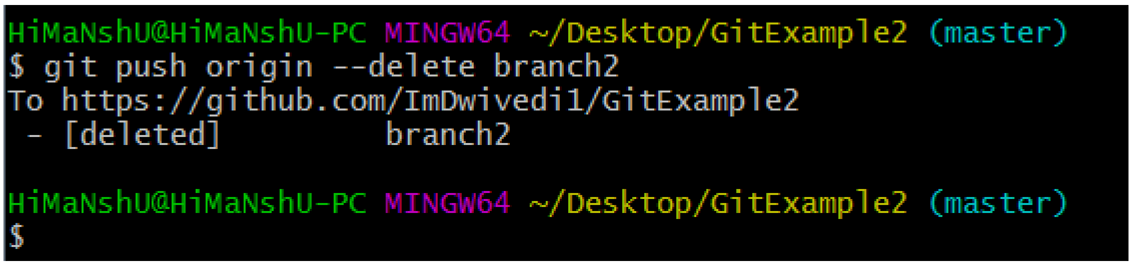
Delete a Remote Branch

You can delete a remote branch from Git desktop application. Below command is used to delete a remote branch:

Syntax:

$ git push origin -delete <**branch** name>

Output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git push origin --delete branch2
To https://github.com/ImDwivedi1/GitExample2
 - [deleted]         branch2

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

As you can see in the above output, the remote branch named **branch2** from my GitHub account is deleted.

GIT COMMIT -M”msg”

It is used to record the changes in the repository. It is the next command after the git add. Every commit contains the index data and the commit message. Every commit forms a parent-child relationship. When we add a file in Git, it will take place in the staging area. A commit command is used to fetch updates from the staging area to the repository.

The staging and committing are co-related to each other. Staging allows us to continue in making changes to the repository, and when we want to share these changes to the version control system, committing allows us to record these changes.

Commits are the snapshots of the project. Every commit is recorded in the master branch of the repository. We can recall the commits or revert it to the older version. Two different commits will never overwrite because each commit has its own commit-id. This commit-id is a cryptographic number created by **SHA (Secure Hash Algorithm)** algorithm.

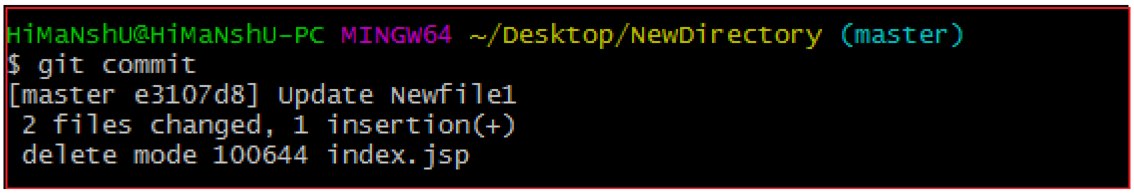
The git commit command

The commit command will commit the changes and generate a commit-id. The commit command without any argument will open the default text editor and ask for the commit message. We can specify our commit message in this text editor. It will run as follows:

$ git commit

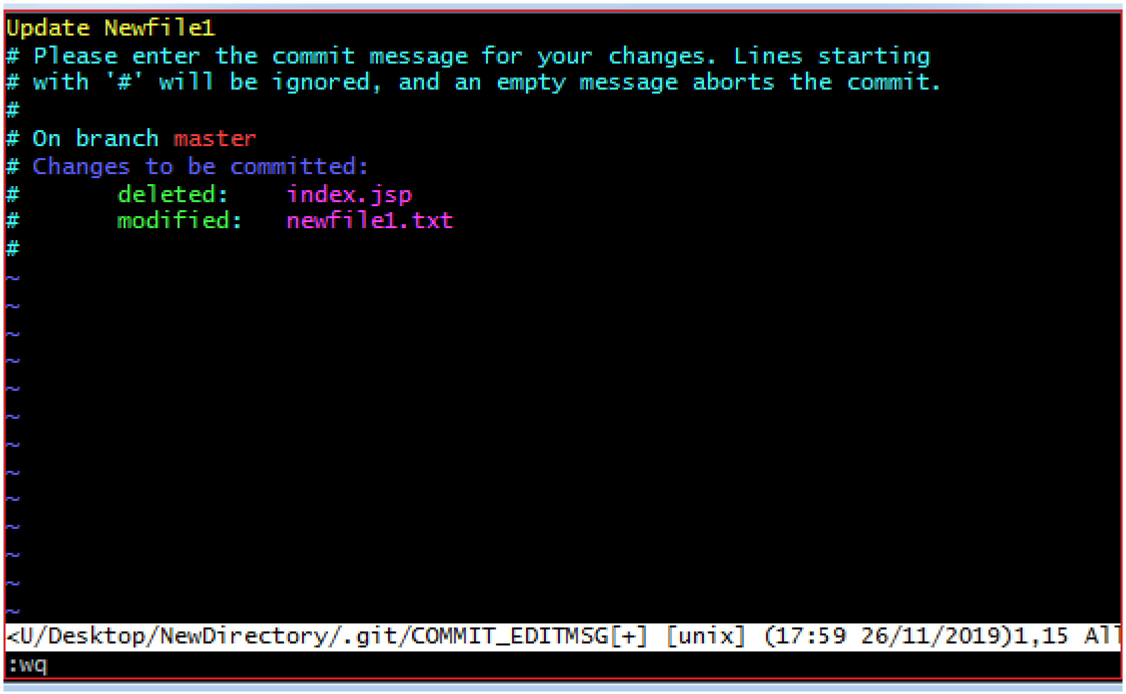
The above command will prompt a default editor and ask for a commit message. We have made a change to **newfile1.txt** and want it to commit it. It can be done as follows:

Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git commit
[master e3107d8] Update Newfile1
 2 files changed, 1 insertion(+)
 delete mode 100644 index.jsp
```

As we run the command, it will prompt a default text editor and ask for a commit message. The text editor will look like as follows:

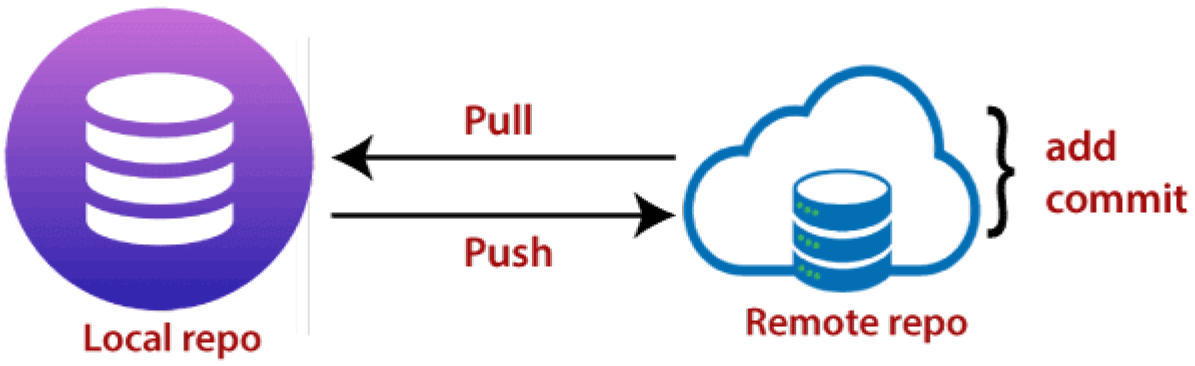


```
Update Newfile1
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       deleted:    index.jsp
#       modified:   newfile1.txt
#
~
~
~
~
~
~
~
~
~
~
~
~
~
<U/Desktop/NewDirectory/.git/COMMIT_EDITMSG[+] [unix] (17:59 26/11/2019)1,15 All
:wq
```

Press the **Esc** key and after that '**I**' for insert mode. Type a commit message whatever you want. Press **Esc** after that '**:wq**' to save and exit from the editor. Hence, we have successfully made a commit.

GIT PUSH

The push term refers to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repository. Pushing is capable of overwriting changes; caution should be taken when pushing.



Moreover, we can say the push updates the remote refs with local refs. Every time you push into the repository, it is updated with some interesting changes that you made. If we do not specify the location of a repository, then it will push to default location at **origin master**.

The "git push" command is used to push into the repository. The push command can be considered as a tool to transfer commits between local and remote repositories. The basic syntax is given below:

```
$ git push <option> [<Remote URL><branch name><refspec>...]
```

Push command supports many additional options. Some options are as follows under push tags.

Git Push Tags

<repository>: The repository is the destination of a push operation. It can be either a URL or the name of a remote repository.

<refspec>: It specifies the destination ref to update source object.

--all: The word "all" stands for all branches. It pushes all branches.

--prune: It removes the remote branches that do not have a local counterpart. Means, if you have a remote branch say demo, if this branch does not exist locally, then it will be removed.

--mirror: It is used to mirror the repository to the remote. Updated or Newly created local refs will be pushed to the remote end. It can be force updated on the remote end. The deleted refs will be removed from the remote end.

--dry-run: Dry run tests the commands. It does all this except originally update the repository.

--tags: It pushes all local tags.

--delete: It deletes the specified branch.

-u: It creates an upstream tracking connection. It is very useful if you are going to push the branch for the first time.

Git Push Origin Master

Git push origin master is a special command-line utility that specifies the remote branch and directory. When you have multiple branches and directory, then this command assists you in determining your main branch and repository.

Generally, the term **origin stands** for the remote repository, and master is considered as the main branch. So, the entire statement "**git push origin master**" pushed the local content on the master branch of the remote location.

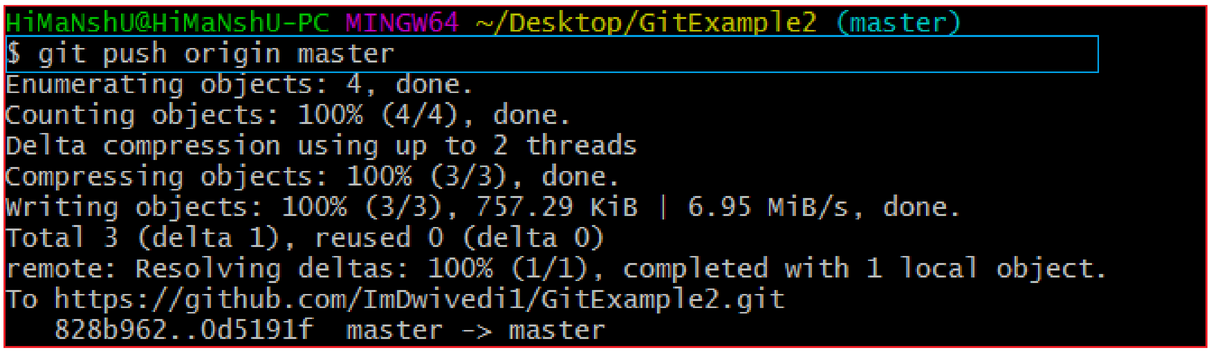
Syntax:

```
$ git push origin master
```

The image is wholly tracked in the local repository. Now, we can push it to origin master as:

1. $ git push origin master

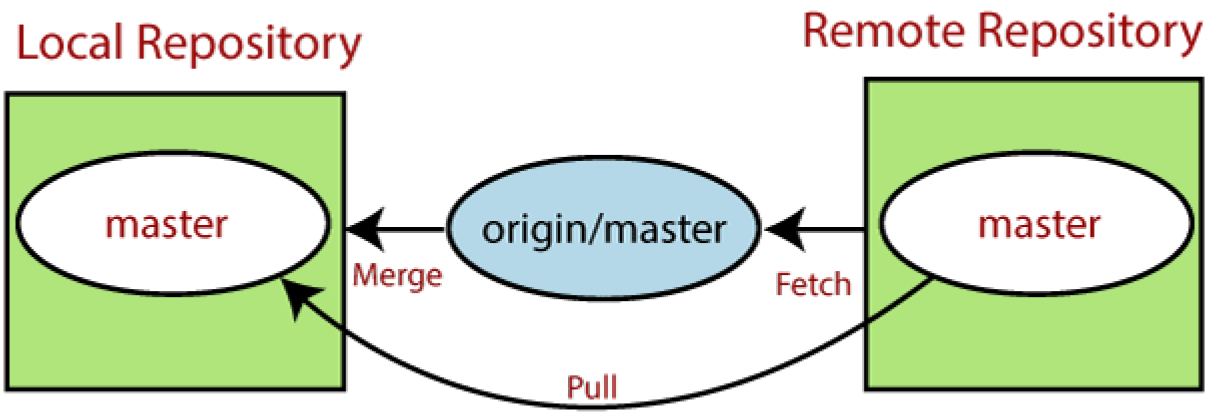
Output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 757.29 KiB | 6.95 MiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ImDwivedi1/GitExample2.git
   828b962..0d5191f  master -> master
```

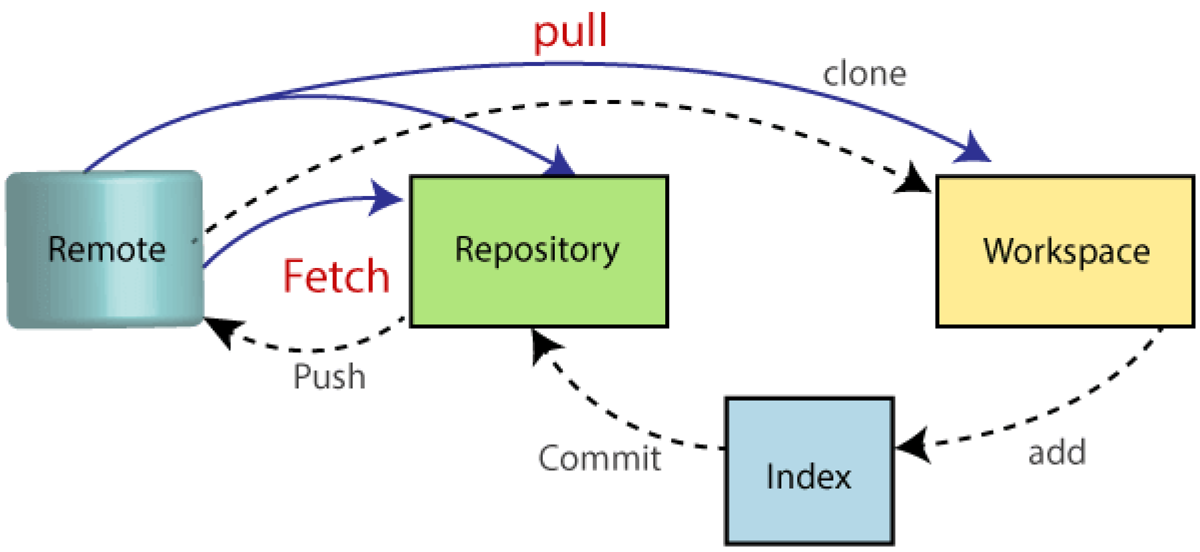
GIT PULL

The term pull is used to receive data from GitHub. It fetches and merges changes from the remote server to your working directory. The **git pull command** is used to pull a repository.



Pull request is a process for a developer to notify team members that they have completed a feature. Once their feature branch is ready, the developer files a pull request via their remote server account. Pull request announces all the team members that they need to review the code and merge it into the master branch.

The below figure demonstrates how pull acts between different locations and how it is similar or dissimilar to other related commands.



The git pull command

The pull command is used to access the changes (commits)from a remote repository to the local repository. It updates the local branches with the remote-tracking branches. Remote tracking branches are branches that have been set up to push and pull from the remote repository. Generally, it is a collection of the fetch and merges command. First, it fetches the changes from remote and combined them with the local repository.

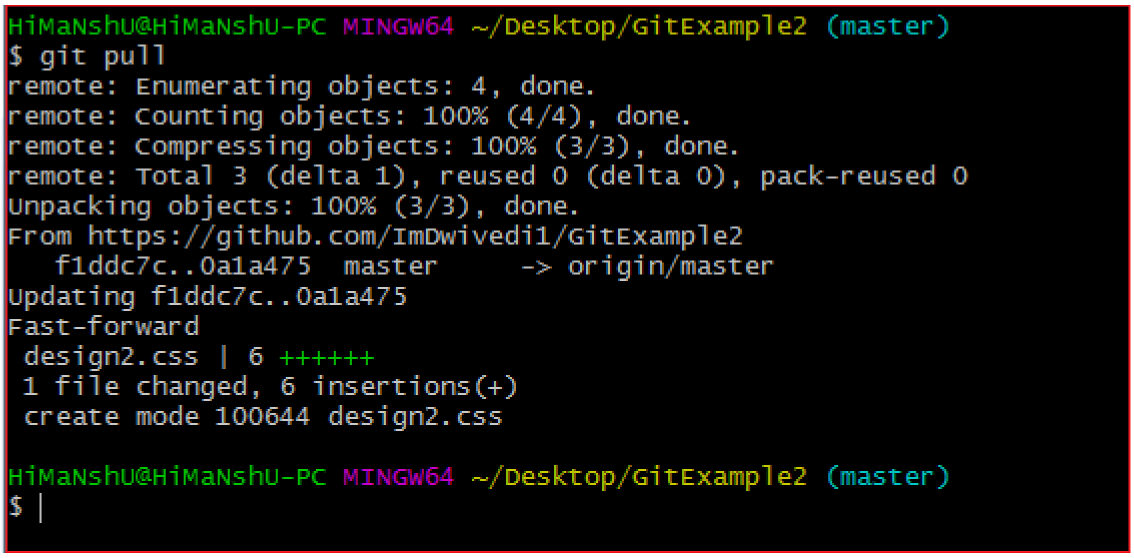
Default git pull:

We can pull a remote repository by just using the git pull command. It's a default option. Syntax of git pull is given below:

Syntax:

1. $ git pull

Output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ImDwivedi1/GitExample2
   f1ddc7c..0a1a475  master     -> origin/master
Updating f1ddc7c..0a1a475
Fast-forward
 design2.css | 6 ++++++
 1 file changed, 6 insertions(+)
 create mode 100644 design2.css

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

Thank You