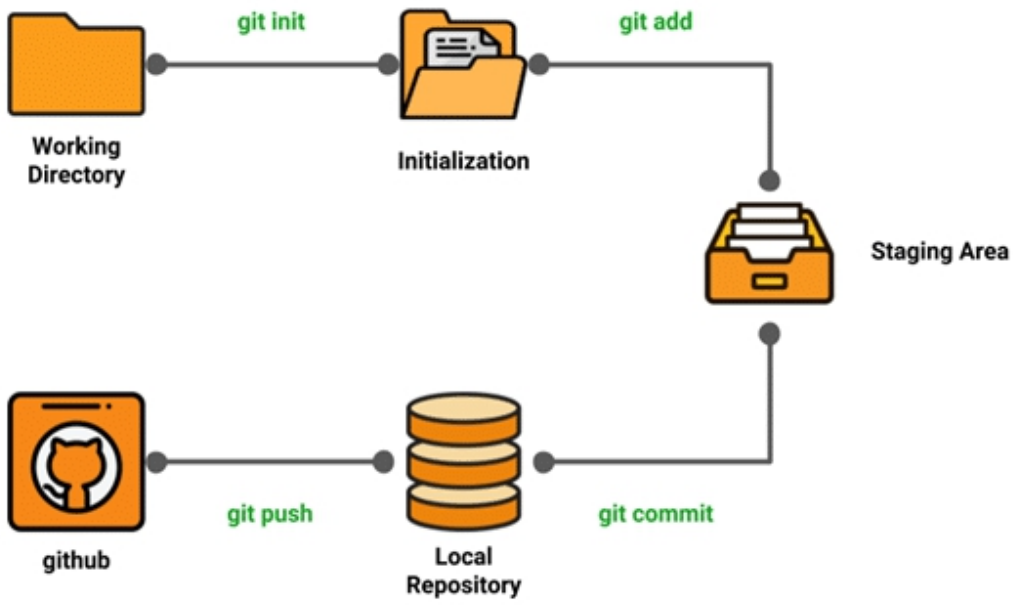
What is Git?

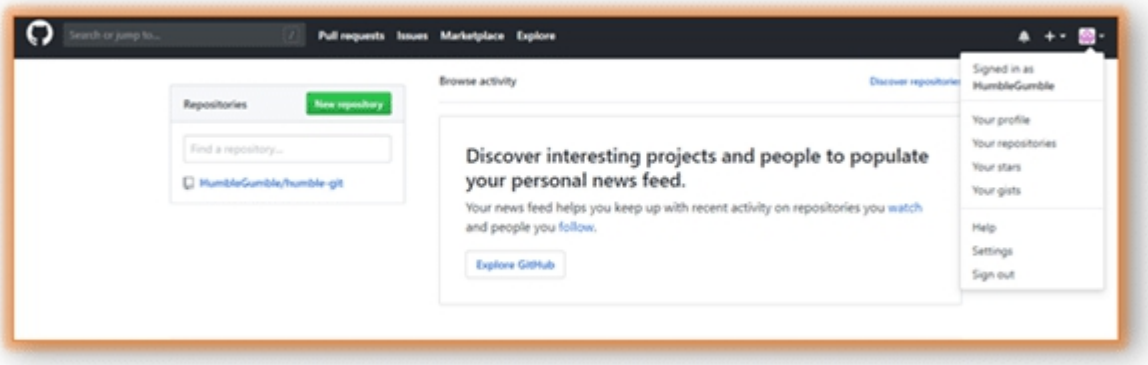
Git helps developers keep track of the history of their code files by storing them in different versions on its own server repository, i.e., GitHub.



Creating a GitHub account:

- Go to https://Github.com
- Create a GitHub account
- Login

After accomplishing these steps, our GitHub page should look like this:

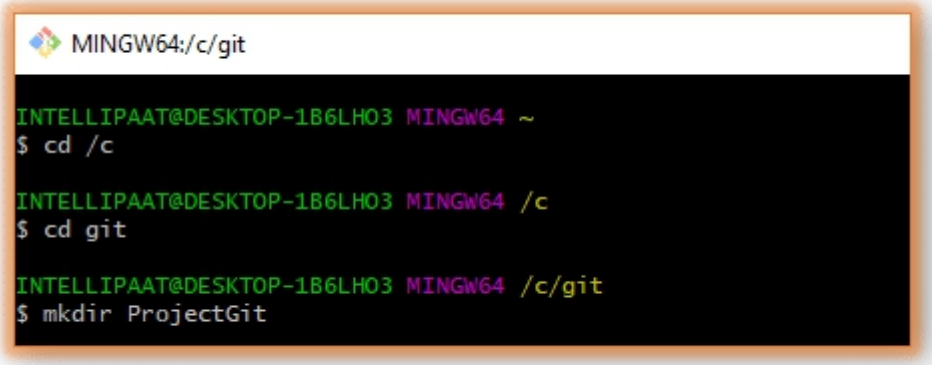


Configuring Git:

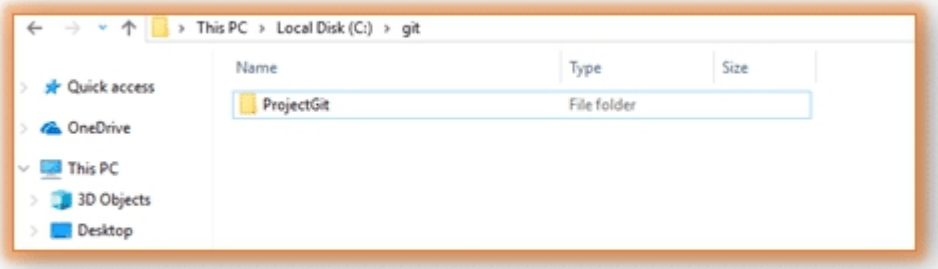
To tell Git who we are, run the following two commands:

Creating a local repository:

To begin with, open a terminal and move to where we want to place the project on our local machine using the cd (change directory) command. For example, if we have a 'projects' folder on our desktop, we would do something like below:



We can go to the Git folder and see the new directory that we have just created.



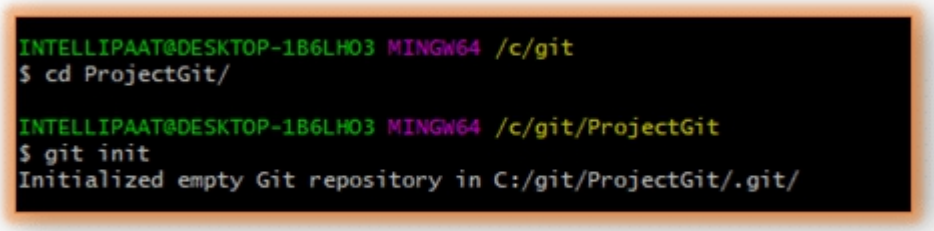
Common Git Commands

We will divide the common Git commands into two primary categories:

- **Local**: git init, git touch, git status, git add, git commit, and git rm
- **Remote**: git remote, git pull, and git push

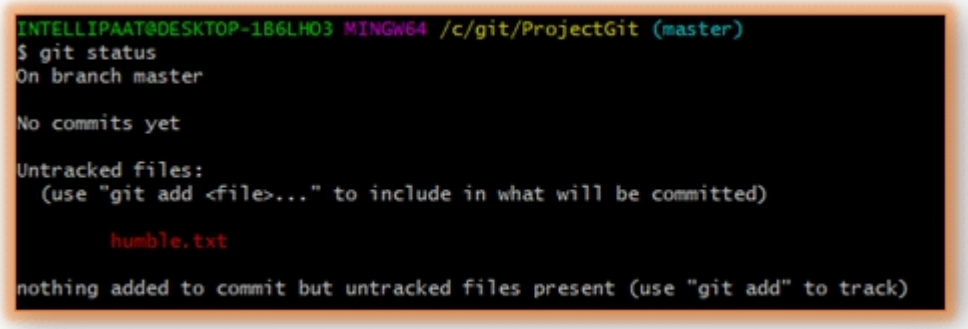
Local Git Commands

- ***git init***: We use the git init command to initialize a Git repository in the root of a folder.



- ***git touch***: To add files to a project, we can use git touch. Let us see how we can add a file to the Git repository we have just created
  Step 1: Create a new file with the command touch
  Step 2: See the files present in our master branch

- ***git status***: After creating a new file, we can use the git status command and see the status of the files in the master branch.



- ***git status***: Now, we will add the humble.txt file to the staging environment by using git add, before going ahead to the staging to see the change using git status.

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (master)
$ git status
On branch master

No commits yet

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

        new file:   humble.txt
```

- ***git commit***: Now we will use the git commit command as shown below:

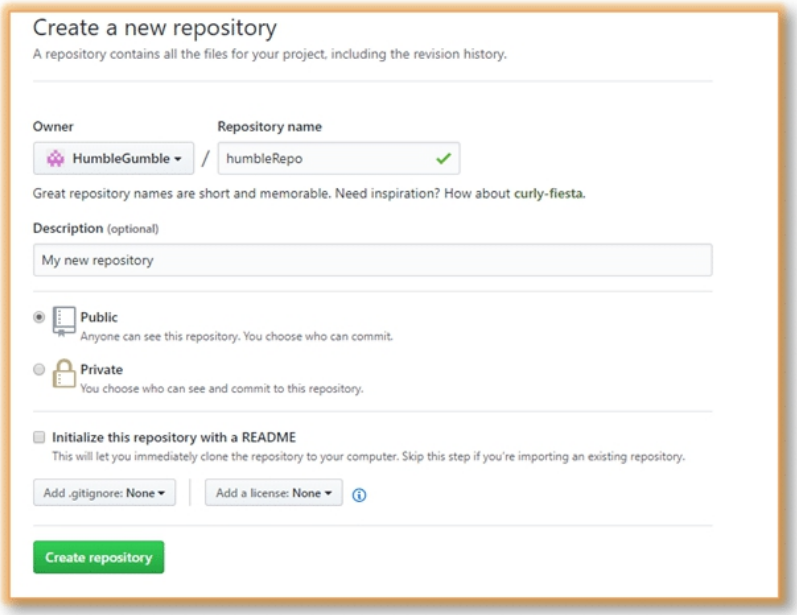
```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (master)
$ git commit -m "What is HumbleGumble?"
[master (root-commit) 169cfba] What is HumbleGumble?
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 humble.txt
```

Thus, we have successfully created our first commit.

Remote Commands

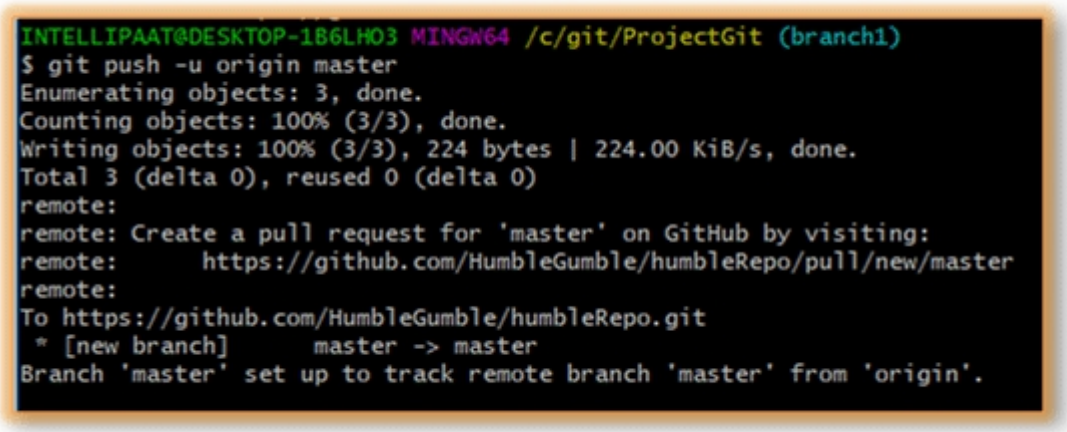
Let us now discuss the remote commands such as remote, push, and pull in Git. For that, we will create a new repository in our GitHub account. How do we do that? Follow the below steps:

Step 1: Go to the GitHub account
Step 2: Create a new repository



To push an existing repository to a remote repository from a command line, follow the commands given below.

- ***Git remote:***

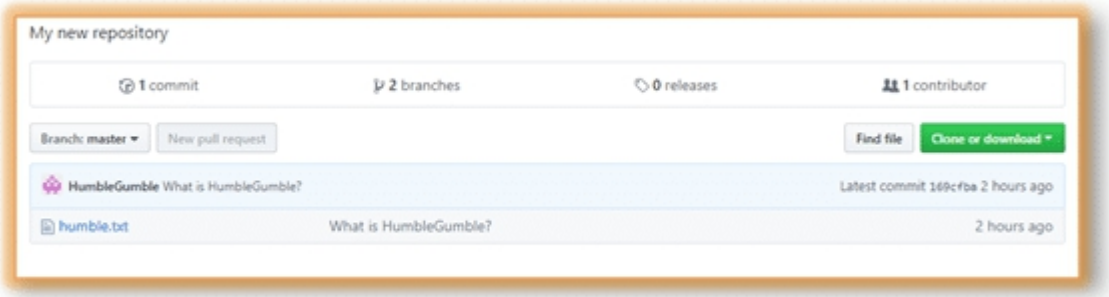


- ***git push***

we can just use the following command, assuming that our brach name is 'branch1':

```
git push origin branch1
```

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (branch1)
$ git push origin branch1
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'branch1' on GitHub by visiting:
remote:      https://github.com/HumbleGumble/humbleRepo/pull/new/branch1
remote:
To https://github.com/HumbleGumble/humbleRepo.git
 * [new branch]      branch1 -> branch1
```



- ***git pull***

When it comes to syncing a remote repository, the pull command comes in handy. Let us take a look at the commands that can leads to the pulling operation in Git.

- *remote origin*
- *pull master*

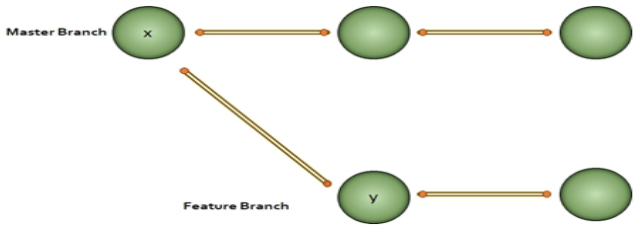
We will use these as shown below:

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (master)
$ git remote set-url origin "https://github.com/HumbleGumble/humbleRepo.git"

INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (master)
$ git pull origin master
From https://github.com/HumbleGumble/humbleRepo
 * branch            master     -> FETCH_HEAD
 + 5ad410f...169cfba master     -> origin/master  (forced update)
Already up to date.
```

Branching in Git

In branching, we mainly make use of the ***git checkout*** *and* ***git branch*** *commands.*



Step 1: We will run **git checkout -b <new branch name>**
Step 2: Then, we will use the git branch command to confirm that our branch is created

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (master)
$ git checkout -b branch1
Switched to a new branch 'branch1'

INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (branch1)
$ git branch
* branch1
  master
```

Now, how to change a branch to the master branch? For that, we use the following command:

```
git checkout master
```

Merging in Git (git checkout, git add, git log, git merge, merging conflicts, and rebasing)

The merge commit represents every change that has occurred on the feature branch since it got branched out from the master.

Step 1: Create a new branch called 'branch2'

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (branch1)
$ git checkout -b branch2
Switched to a new branch 'branch2'
```

Step 2: Create a new file in the branch

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (branch2)
$ touch newFile.txt
```

Step 3: Add changes from all tracked and untracked files

Note: Refer to the following **git add** attributes

```
 -n, --dry-run         dry run
 -v, --verbose         be verbose

 -i, --interactive     interactive picking
 -p, --patch           select hunks interactively
 -e, --edit            edit current diff and apply
 -f, --force           allow adding otherwise ignored files
 -u, --update          update tracked files
 --renormalize         renormalize EOL of tracked files (implies -u)
 -N, --intent-to-add   record only the fact that the path will be added later
 -A, --all             add changes from all tracked and untracked files
 --ignore-removal      ignore paths removed in the working tree (same as --no
ll)
 --refresh             don't add, only refresh the index
 --ignore-errors       just skip files which cannot be added because of error

 --ignore-missing      check if - even missing - files are ignored in dry run
 --chmod (+|-)x        override the executable bit of the listed files
```

In our case, we have given the command as **git add -A** and after that, we will commit one sentence as shown below:

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (branch2)
$ git add -A

INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (branch2)
$ git commit -m "Added a file in branch2"
[branch2 59ce99d] Added a file in branch2
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newFile.txt
```

Step 4: Check the last three logs by running the command: **git log -3**

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (branch2)
$ git commit -m "Added a file in branch2"
[branch2 59ce99d] Added a file in branch2
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newFile.txt

INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (branch2)
$ git log -3
commit 59ce99d0690aacba8c4ce9aaf62bcac158821f9d (HEAD -> branch2)
Author: HumbleGumble <debashis.intellipaat@gmail.com>
Date:   Thu Nov 29 16:17:48 2018 +0530

    Added a file in branch2

commit 169cfba94dbbc394c895353fe6b1bfaa38890e9f (origin/master, origin/branch1,
master, branch1)
Author: HumbleGumble <debashis.intellipaat@gmail.com>
Date:   Thu Nov 29 13:15:35 2018 +0530

    What is HumbleGumble?
```

We have created another branch on our master branch. Now, we will see how to perform **merging**.

Let us get inside the master branch using the following command:

```
git checkout master
```

After that, we will perform merging with the help of the below command:

```
git merge branch2
```

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (branch2)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (master)
$ git merge branch2
Updating 169cfba..59ce99d
Fast-forward
 newFile.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newFile.txt
```

Now, we have successfully merged the feature branch into the master branch. Let us take a look at how it looks inside the master branch using the following command:

```
git log --graph
```

```
INTELLIPAAT@DESKTOP-1B6LHO3 MINGW64 /c/git/ProjectGit (master)
$ git log --graph
* commit 59ce99d0690aacba8c4ce9aaf62bcac158821f9d (HEAD -> master, branch2)
| Author: HumbleGumble <debashis.intellipaat@gmail.com>
| Date:   Thu Nov 29 16:17:48 2018 +0530
|
|     Added a file in branch2
|
* commit 169cfba94dbbc394c895353fe6b1bfaa38890e9f (origin/master, origin/branch1
, branch1)
  Author: HumbleGumble <debashis.intellipaat@gmail.com>
  Date:   Thu Nov 29 13:15:35 2018 +0530

      What is HumbleGumble?
```

Here, we can see the graph with commits on both master and feature branches. The red part of the graph indicates the merging operation.

Advantages of merging:

- Merging allows parallel working in collaborative projects.
- It saves the time that is consumed by manual merging.

The disadvantage of merging:

- Merging conflicts may occur while merging branches.

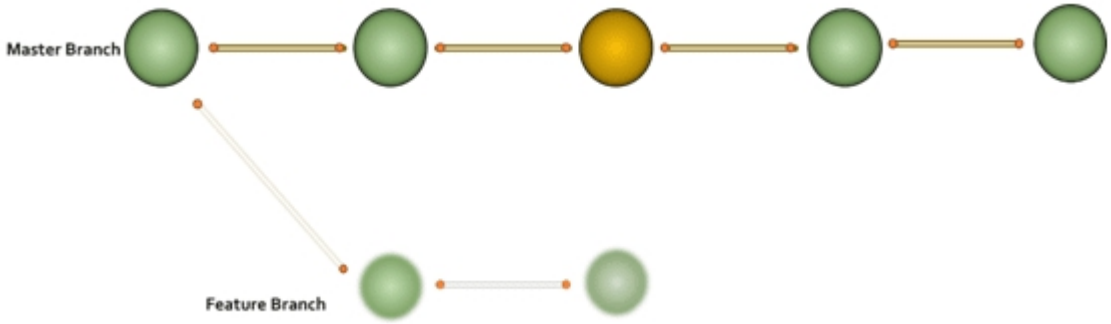
Now that we have successfully learned to branch and merge with Git and GitHub, further in this Git tutorial, let us look at yet another important Git operation, i.e., rebasing.

Git Rebase

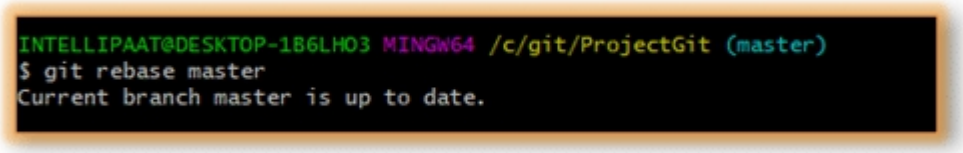
When our project becomes relatively large, the commit log and the history of the repository become messy. Here, we use rebasing. Rebasing will take a set of commits, copy them, and store them outside our repository. This helps us maintain a linear sequence of commits in our repository.

Let us take the same example. Here, we will rebase the master branch and see what happens.

Note: In rebasing, the base of the feature branch gets changed and the last commit of the master branch becomes the new base of the feature branch.



Now, let us perform rebasing in Git Bash.



The advantage of rebasing:

- Rebasing provides a cleaner project history.

The disadvantage of rebasing:

- In a collaborative workflow, re-writing the project history can be potentially catastrophic.