**Case Study Git**

* Local Operations :-

1] Create a new local git repository on to your machine and configure it for

user.name and user.email attribute.

🡪 $ git config --global user.name "Anirban"

🡪 $ git config --global user.email "anirban\_tapadar@persistent.com"

2] Create a files in directory (File1, IgnoreFile and FileToDelete provided), say directory contains – File1.txt, IgnoreFile.txt and FileToDelete.txt, now start tracking only the File1.txt and FileToDelete.txt, and ignore the IgnoreFile.txt by using .gitignore.

🡪In .gitignore file--- /ignoreFile.txt

🡪 $ git add File1.txt FiletoDelete.txt .gitignore

3] Commit these changes to repository.

Use commit message as “This is First Commit, Tracking File1 and FileToDelete”

Note: - only File1.txt and FileToDelete.txt should be tracked and not the IgnoreFile.txt

🡪 $ git commit -m " This is First Commit, Tracking File1 and FileToDelete "

4] Make 2 more commits by making following changes in the File1.txt

Add changes to file as:-

Change1 – commit with message as “Change1 in File1”

🡪 $ git commit -am "Change 1 in File1"

Change2 – commit with message as “Change2 in File1”

🡪 $ git commit -am "Change 2 in File1"

And if you check the log it should display in following manner: -

Change2 in File1

Change1 in File1

This is First Commit, Tracking File1 and FileToDelete

🡪$ git log –oneline

5] Delete the FileToDelete.txt file from directory and commit with message as “FileToDelete is deleted”

🡪 $ git rm FiletoDelete.txt

🡪 $ git commit -m "FiletoDelete is deleted"

6] Display logs:-

- Show all the commit logs.

- Show last three logs in one line

- Show only commits that occur between from date and to date

- Show only those commit message that contains “File1” as in commits.

🡪 $ git log –oneline

🡪 $ git log -n 3 –oneline

🡪 $ git log --after="2022-03-02" --until="2022-03-03"

🡪 $ git log --grep "File1" –oneline

7] Permanently undo a “FileToDelete is deleted” commit snapshot.

🡪 $ git reset 7b54a73

8] Rename File1.txt to TestFile.txt

🡪 $ git mv File1.txt TestFile.txt

* Branching and Rebasing

1] Create a branch ‘bug’, checkout to bug branch to solve the bug in file1.

Bug is – Put a “ ”(space) between numerical and letters

e.g.:- Change1 à Change 1

🡪 $ git branch bug

🡪 $ git checkout bug

2] Commit this change with message as “Bug Resolved – Space added between numerical and letters”

🡪 $ git commit -am "Bug Resolved – Space added between numerical and letters"

3] Merge the changes to master assuming bug is resolved.

🡪 $ git checkout master

* $ git merge bug

4] Delete branch ‘bug’

🡪 $ git branch -d bug

5] Recover the deleted branch ‘bug’ and rename it to ‘bug123’

🡪$ git reflog bug

🡪 $ git checkout -b bug 48094V4

6] Consider the following scenario and achieve the same to Rebase:-

- Create new branch ‘idea’ and make a new commit on ‘idea’ branch

- Now checkout to master and make a new commit on master

🡪 $ git branch idea

🡪 $ git checkout idea

🡪 $ git commit -am "New Commit on idea branch"

🡪 $ git checkout master

🡪 $ git commit -am "New Commit on master branch"

7] Refer the below image:-

master

A

idea

B

D

C

E

***Now use Rebase to get the linear story line as follows:-***

master

A

D

E

B

C

🡪$ git rebase master

Successfully rebased and updated refs/heads/idea.

* $ git checkout master
* $ git merge idea

8] Now say bug is resolved and you are good to release this stable version, tag this release as ‘v1.0’ with message as “Stable version 1.0 released”.

🡪 $ git tag -l "v1.0"

🡪 $ git tag v1.0 -m "Stable version 1.0 released"

9] Consider the following scenario to work with Stashing:-

Checkout to ‘bug123’ branch first and add text as “Change 3” in File1.txt, stage this change and don’t commit the same so that you can get a dirty state of working directory.

🡪 $ git checkout bug123

🡪 $ git add File1.txt

Now assume that you have a priority task to be made on master branch, checkout to master and store this dirty state (on ‘bug123’) to stash so that you get back to this state later.

🡪 $ git checkout master

🡪 $ git stash

Saved working directory and index state WIP on master: 9cef0f5 New commit in master branch

10] Display the list of all stash.

🡪 $ git stash list

11] Checkout to ‘bug123’ branch, reapply the stash which is stored recently and finally commit this change with the message as “Change 3 in File1”

🡪 $ git checkout bug 123

🡪 $ git stash pop

🡪$ git commit -am “Change 3 is done in File1”

* **Git Remote:-**

1] Create (Signup) GitHub account by visiting <https://github.com/>

🡪Done!!

2] Create new repository as “TestRepo”.

🡪Done!!

3] Add remote on to your local repository and name it as ‘Origin’.

🡪 $ git remote add Origin <https://github.com/I-ownthispage/TestRepo.git>

4] Display all remote names with their details.

🡪 $ git remote -v

5] Rename remote branch ‘Origin’ to ‘RemoteBranch’.

🡪 $ git remote rename Origin RemoteBranch

6] Take a copy (Clone) of remote repository to your local machine.

🡪 $ git clone <https://github.com/I-ownthispageTestRepo.git>

7] Now create a new file as ‘RemoteFile.txt’ on ***remote*** repository, make 1 commit by making following change in the RemoteFile.txt

Add changes to file as:-

Change1 – commit with message as “Change1 in RemoteFile”

And if you check the log on ***remote*** it should display in following manner:-

Change1 in RemoteFile

Note: - If you check log on local there will be no commits which are made on remote.

🡪 Done!!

8] Fetch remote ‘Origin’ to synch up with remote.

🡪 $ git fetch

🡪 $ git status

9] Finally merge the remote changes to local master.

🡪 $ git pull

10] Now, make 1 more commit on ***remote (GitHub)*** by making following change in the RemoteFile.txt

Add changes to file as:-

Change2 – commit with message as “Change2 in RemoteFile”

And if you check the log on ***remote*** it should display in following manner:-

Change2 in RemoteFile

Change1 in RemoteFile

🡪 Done!!

11] Use Pull to take these recent changes to local.

🡪 $ git pull

12] Now, make 2 new commits on ***local*** by making following change in the RemoteFile.txt

Add changes to file as:-

Change3 – commit with message as “Change3 in RemoteFile”

Change4 – commit with message as “Change4 in RemoteFile”

And if you check the log on ***local*** it should display in following manner:-

Change4 in RemoteFile

Change3 in RemoteFile

Change2 in RemoteFile

Change1 in RemoteFile

🡪 $ git commit -am “Change 3 in RemoteFile”

🡪 $ git commit -am “Change 4 in RemoteFile”

🡪 $ git log --oneline

13] Push these recent changes to remote.

🡪 $ git push