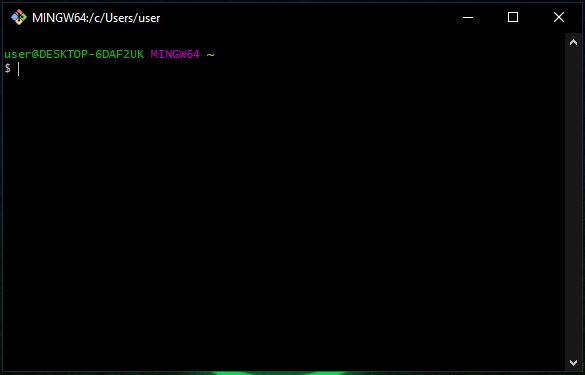
**Git and GitHub Tutorial from CodewithHarry**

Before doing anything we need to download and install Gitbash (for windows users because it doesn’t support Linux type in-built terminals to run git commands).

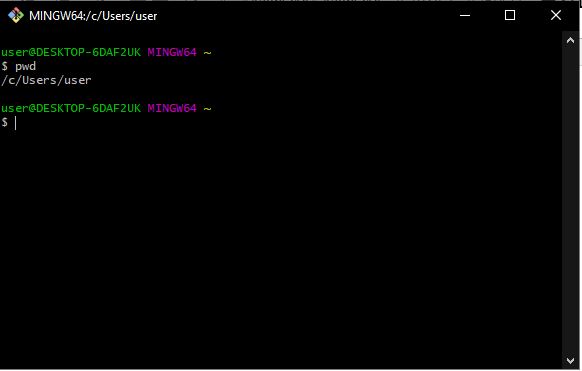
After downloading and installing Gitbash , double click on the Gitbash icon in the screen.

A black cmd type of screen comes up in front .

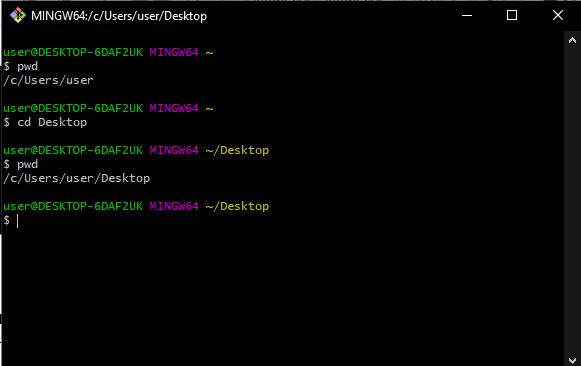
This screen supports all the Linux commands and we can use them In windows now.

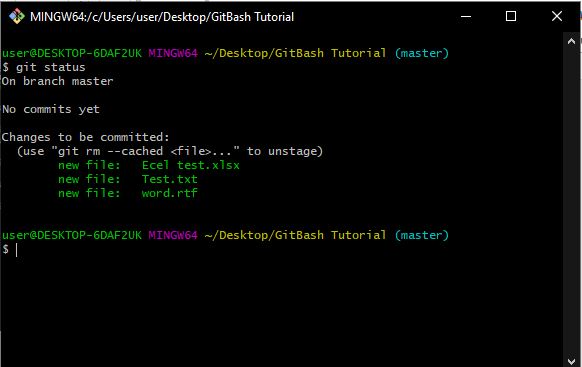
Gitbash is a kind of terminal in which a windows user can use Linux commands .

**pwd –**It stands for **Present Working Directory,** a linux command which shows the status of the current working directory.(The directory we are currently in , it doesnot tell us the contains of the folder like which file it contains)



**cd –** A known and used command in cmd also which is used to change the working directory (if it available). In the image we change the current working directory to Desktop.

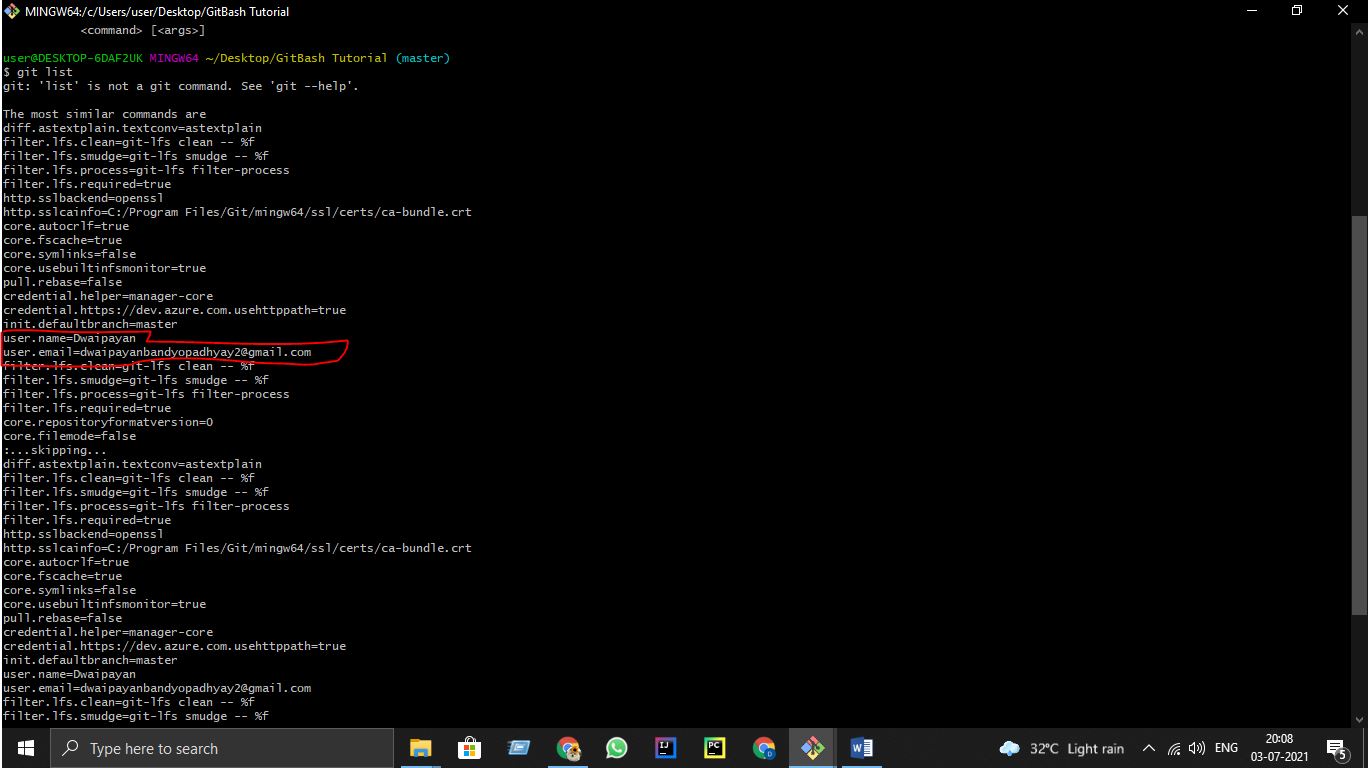


**git status –** This will show the status of that folder i.e the files it contain and their status (are they ready to be committed in the repo or not)

**git config –global user.name “name” –** This is the first thing to do after installing Gitbash, we are getting acquainted with the PC and telling the user details (who is the admin and whose files we will operate).

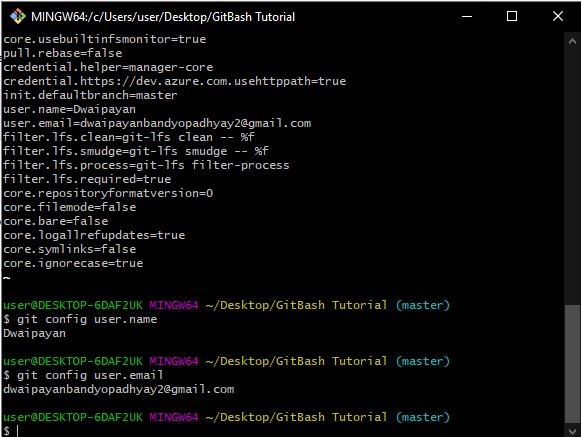
**git config –global user.email “email” –** This is used to tell the PC about the base email of the user.

**git config –list –** This will show us many things (most of them I don’t know about , but it will also show us the user name and email). By using this we will be sure that Git has accepted our name and email and can figure us out.



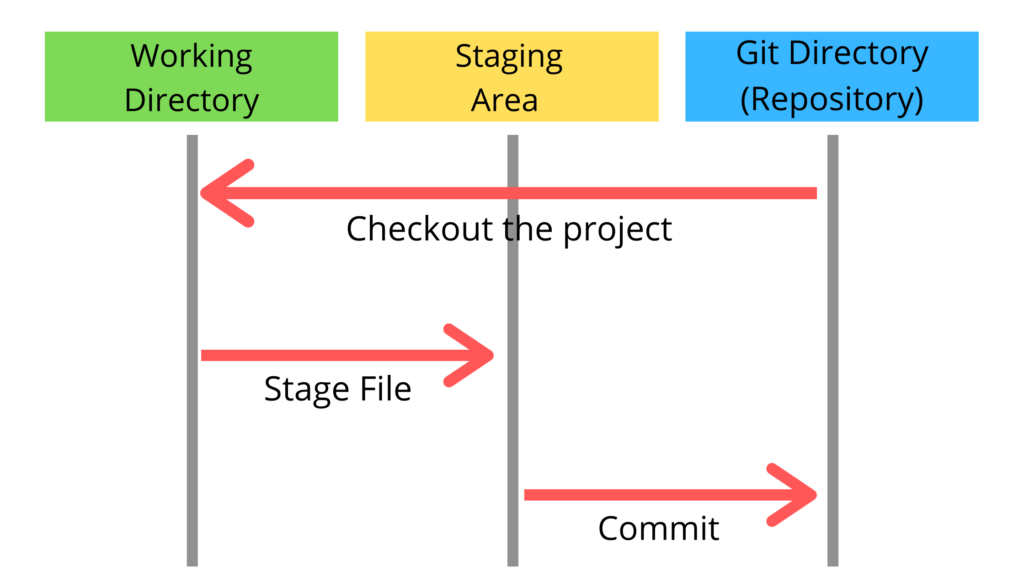
\*If we use “**Up arrow key**” we will get the last command we used into the current working line .

**git config user.name –** To show us the user name saved in the pc.

**git config user.email –** To show us the user email saved in the pc

**Git 3 stage Architecture: -** Git uses a 3-tier or stage architecture

**Working directory, Staging Area, local repository.** The three stages of git can store different (or the same) states of the same code in each stage.



As we can see in above diagram there are three stages in git. When we give any file to the git that file goes from each stage at least once. The three stages of git can store different(or the same) states of the

In above diagram there are three stages those are Working directory,Staging area and Git directory(Repository).

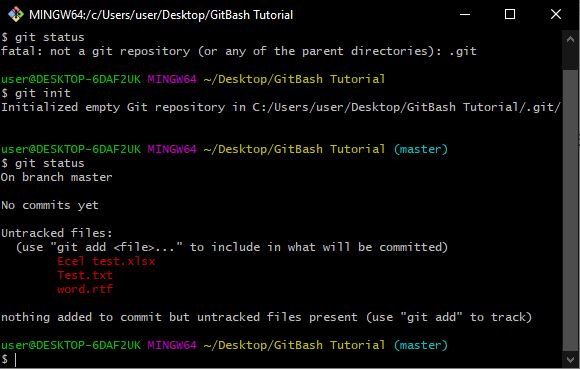
Working directory specifies the file explorer’s folder where your files are stored, Staging area is a area where your those files are present which you want to send to commit(to create snapshot of files), After commit is fired, files which are in staging area will move to Git Repository.

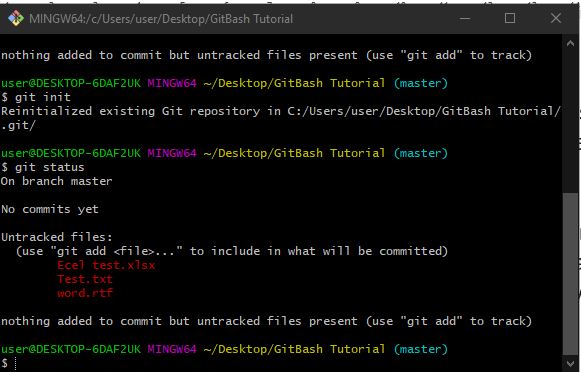
Now if you made any changes in the files which are in Git repository, those files(with changes) will be in Unstaging area. You again have to add them into Staging area and Commit.

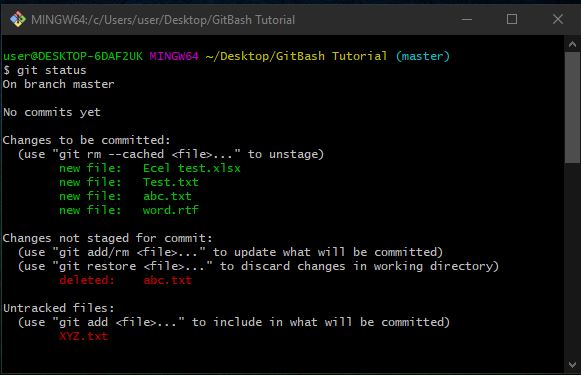
git allows us to first push our files to staging area, and then we can commit those staged files. Continuing our above example, we can only push those two bug-free complete files to staging area and then further commit our changes. The other two files in progress can remain in our working directory. They can be modified later and committed when you need to. Another important advantage of staging area is that files from staging area can again be pulled to our working directory, if we feel they need some more changes before committing them.

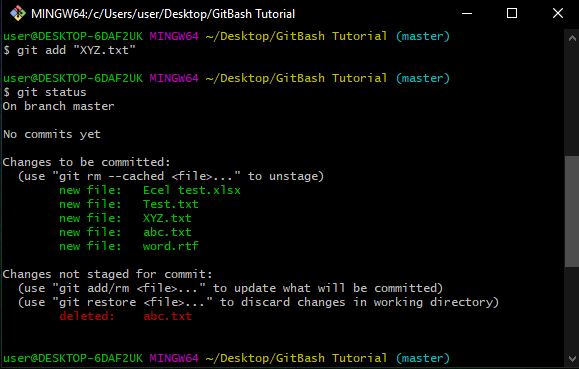
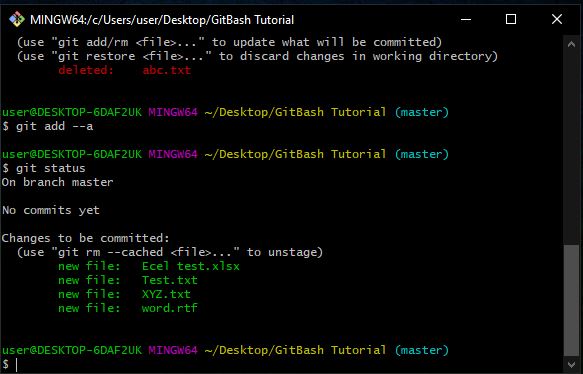
It is to be noted that it is not compulsory in git to first push the files to staging area. The changes can even be directly committed, but it is always a good practice to first stage our files.

**git status :-** This command is used to see the status of the files inside of that git repository. (if any available), if it is not initialized before then it throws an error.

**git init :-** This command is used to initialize a new git repository (if there are not any repo present from earlier), if there are already a git repo then this command will reinitialize that repo but will never create two repos in one folder.



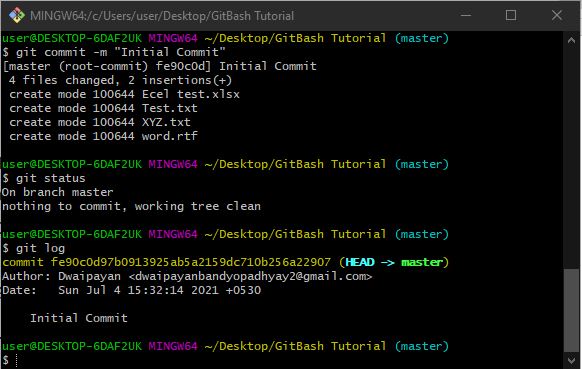
**git add –a or git add . :-** This command will put ALL files available in the staging area. i.e they are ready to be committed now . or we can add files one by one by **git add file-name.extension .**



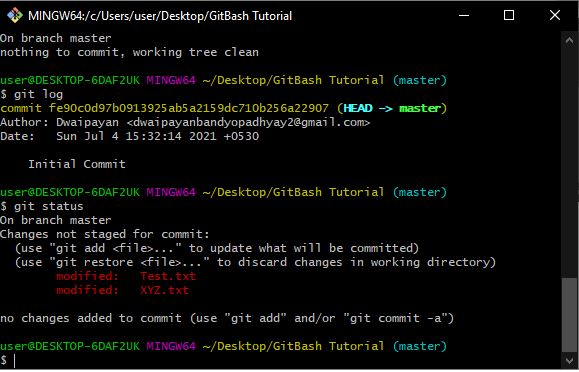
the files in green shows that they are ready to be committed. They are currently in staging area.

**git commit -m “Your message” :-** This command is used to commit all the files that were in staging area with a message using the “**-m”.** Now if we use git status it will show us that “Nothing to commit and the working tree is clean” means there are currently no files in the staging area.

**git log :-** to get the status/ information of the last commit made.

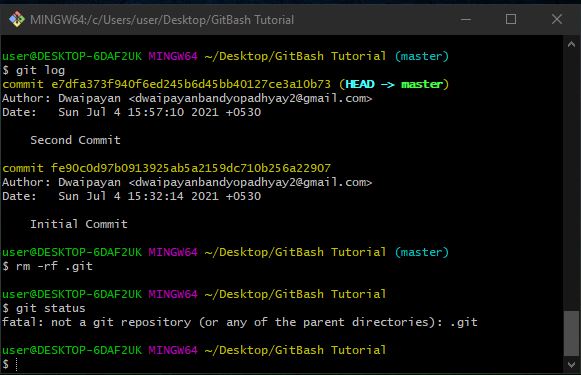


Now if we modify our “committed” files then it will show that the files are being modified and yet to be committed.

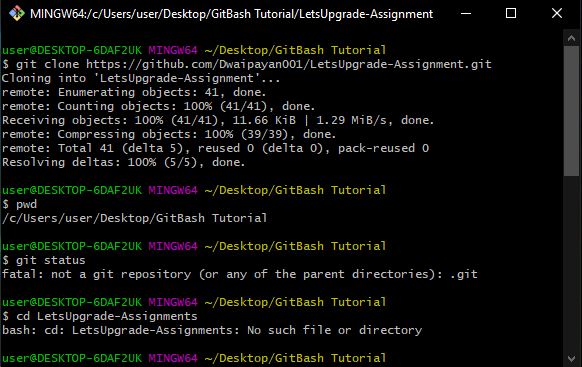


**Cloning a pre-existed repo from GitHub :-**

**rm -rf .git :-** Potentially the most dangerous command available as it will delete the entire .git folder made using git init command earlier and make it blank. If we check the status then it will show that no repo is found.

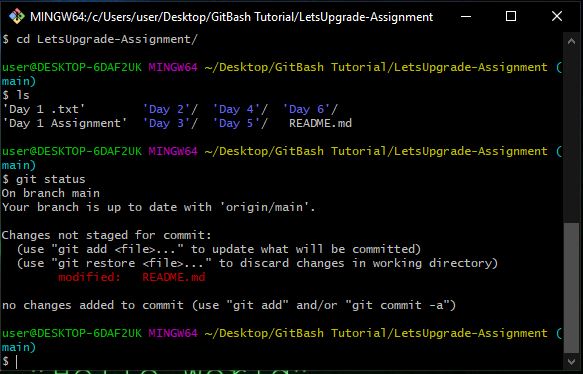


**git clone \*url\* :-** This command helps us to clone a Github repository into our local folder . **To paste the url copied from github / green colored code tab in the gitbash we can use “RightClick -> paste” or use the shortcut key “shift + insert”.**

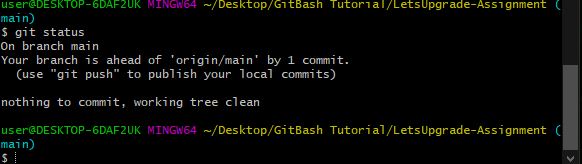
****

**ls –** This command is used to list the files present in the current working directory. (no git is required in front)

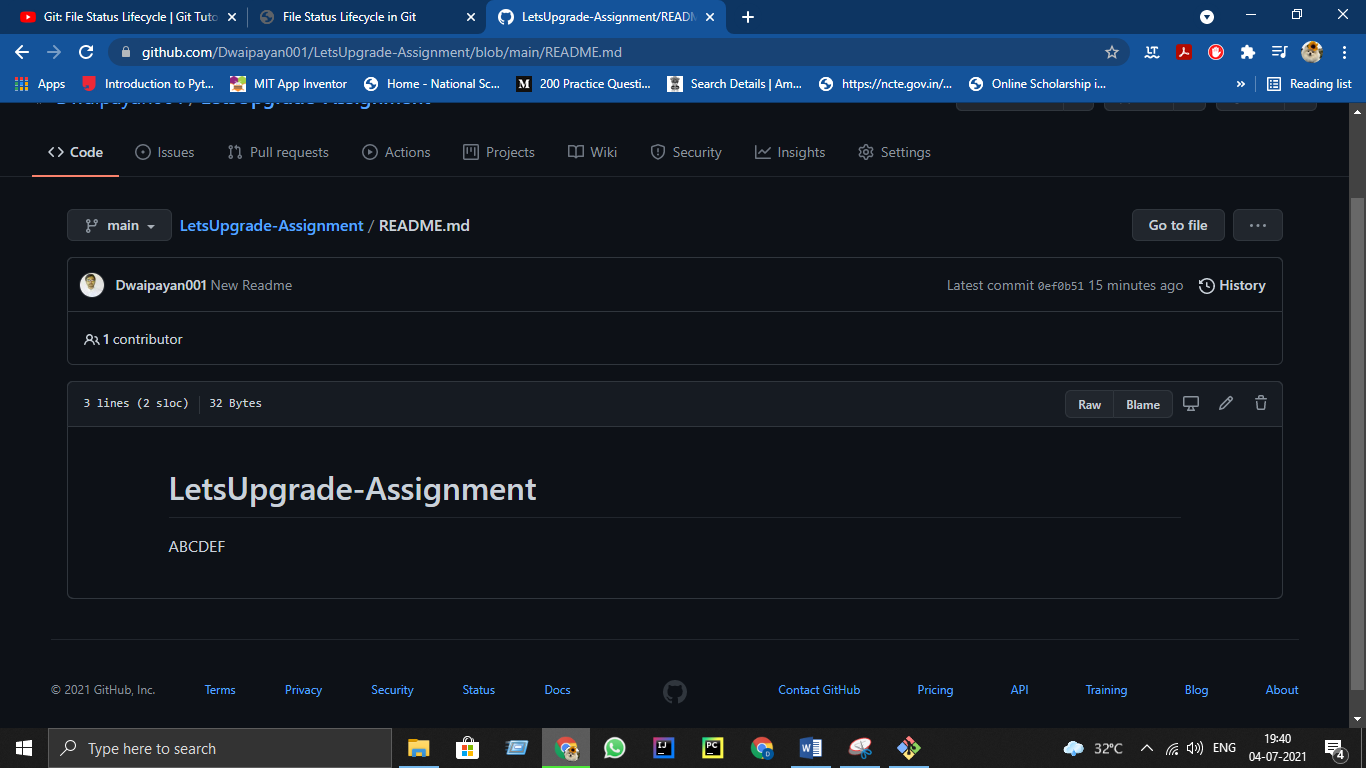
After changing the directory if we make some changes and see the status it will show us the change we have made and that change is still yet to be in the staging area.

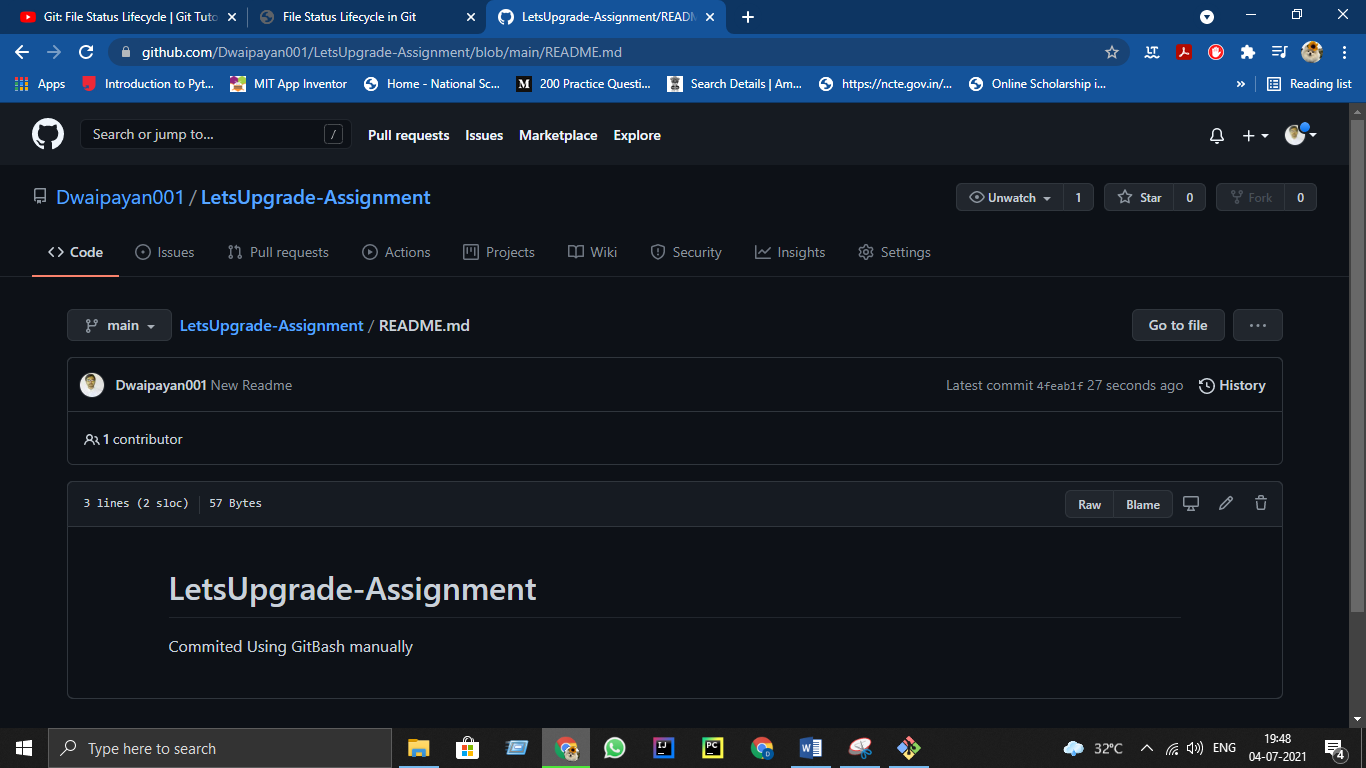


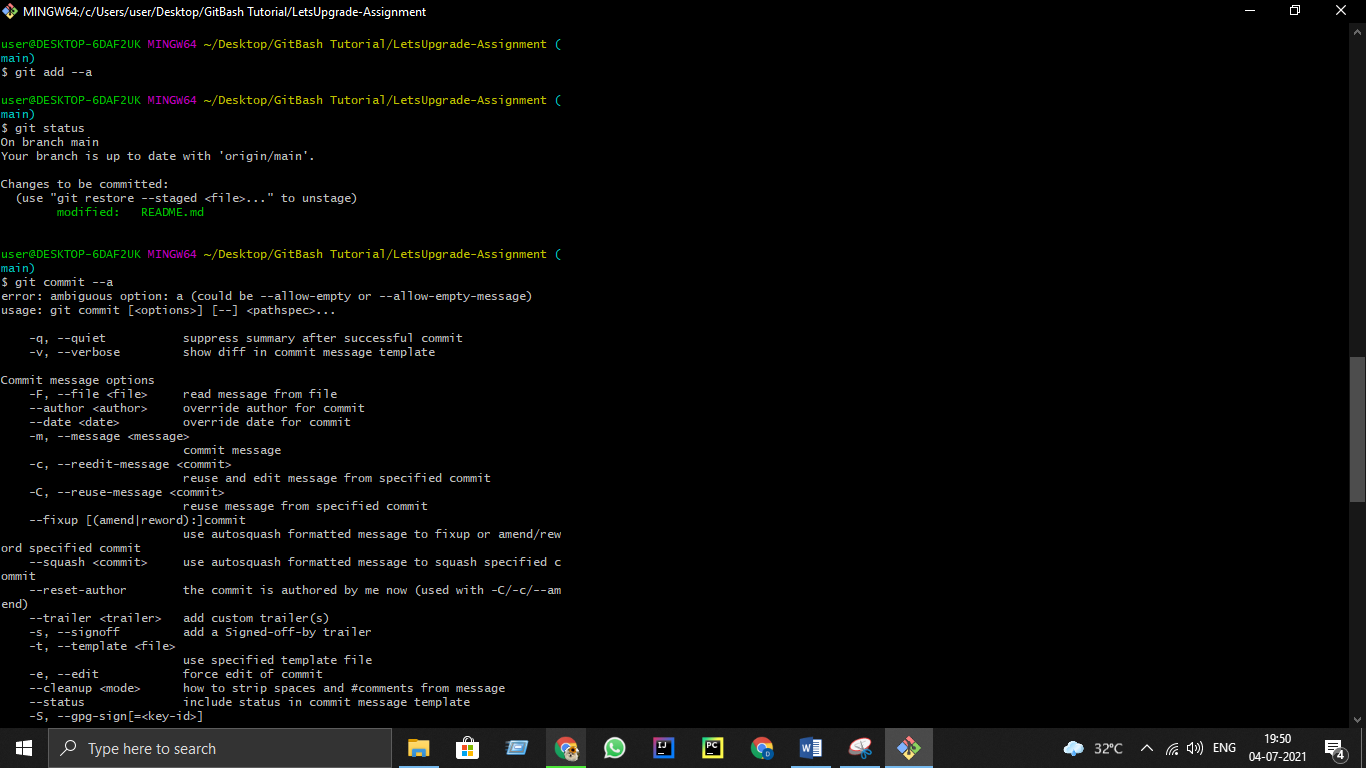
Now if we commit the new changes in the git repo we will be able to do that of course but there will a message there that **“use \*git push\* to publish your local commit”** which means that the commit we just made has been saved in our local repo or pc but if we want to make some change in the real one (the one we cloned from github) then we have to **“push**” instead of only “commit”



**Pushing a file into a GitHub Repo using GItbash** –

Screenshot before push -

Screenshot after push (with the message provided in place of ABCDE-

**Behind the Scenes** -

**File Status Life Cycle :-**

Numbers are according to the numbers made in the images after definitions.

**1 & 2- untracked –** when we create an empty .git repository(using init) in a folder containing some files , those files become untracked files because I have initialized a blank .git folder but the mother folder contains some files from already which are not included in the .git folder (till now), so they are untracked (shown by git status command)

**3 and some 4 - unmodified :-**  using the git add command we put some files or all the untracked files in to the staging area, means they are ready to commit or pushed , they are unmodified files.

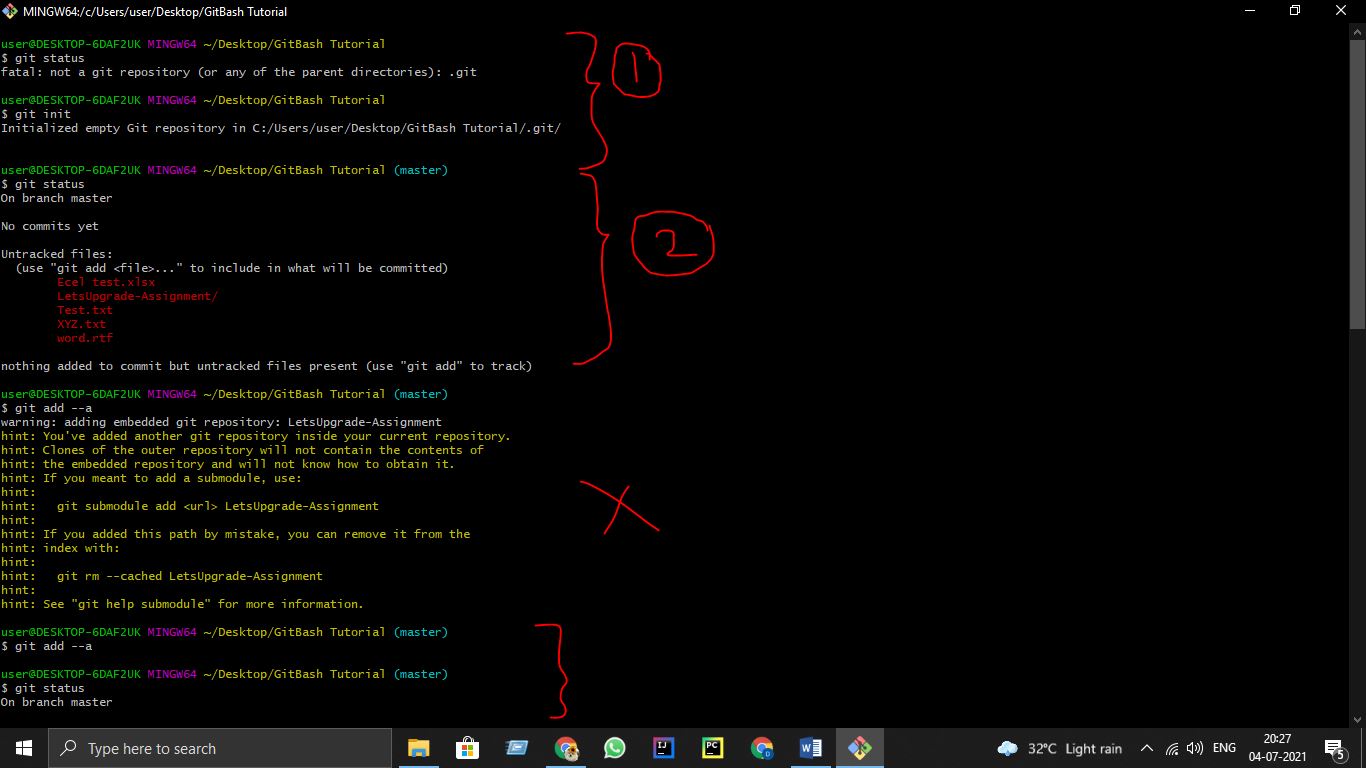
**rest 4 - modified :-**  After adding the files into the stage area if we change any of the files from them then that file is known as modified file and that is now become an untracked file too. We need to add it again to the stage area.

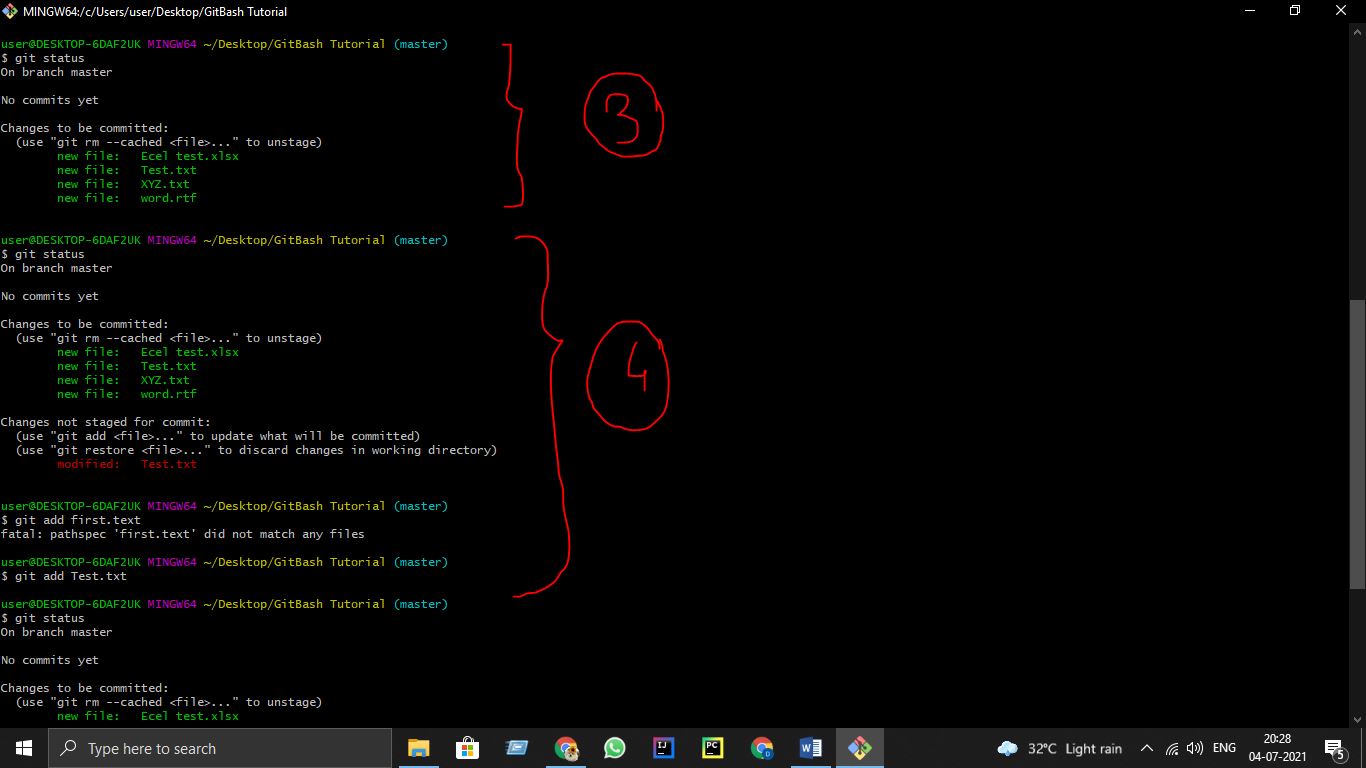
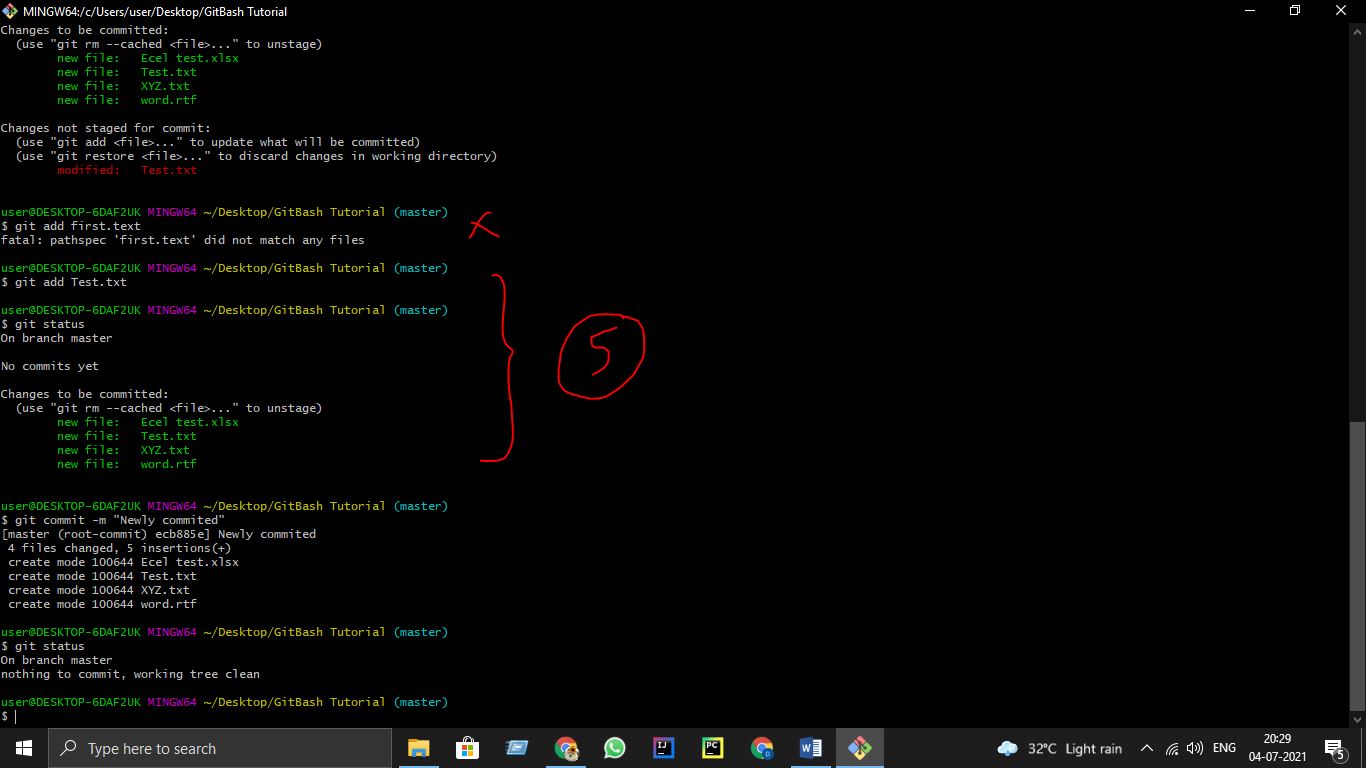
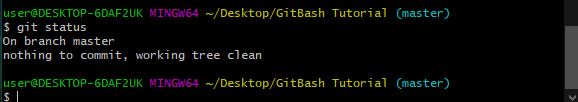
**5 and rest - staged –** when we use the git add command for the second time (if we have made any changes after the first add command) then It goes to the staging area means it is ready to commit and after commit it will return back to unmodified state as there will be no more files to be modified until we add some new files manually .

\***git add is a multipurpose command i.e when we are in untracked state , we have just initialized the .git repo , if we use add then , the files which are available in the mother folder will start to get tracked and become unmodified , then if we change any of the files while in the unmodified state it becomes modified , and then again if we use the add command it will now be staged and ready to commit.**

**If we doesn’t make any change i.e modify the files after using the first add command and use commit after it , it will directly put those files into the staging area. That’s why it is called a multipurpose command.**

**Images –**

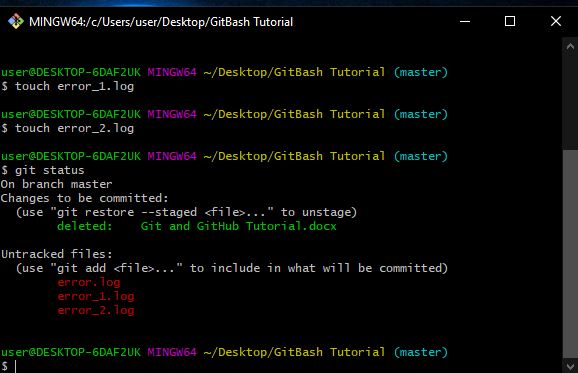
****



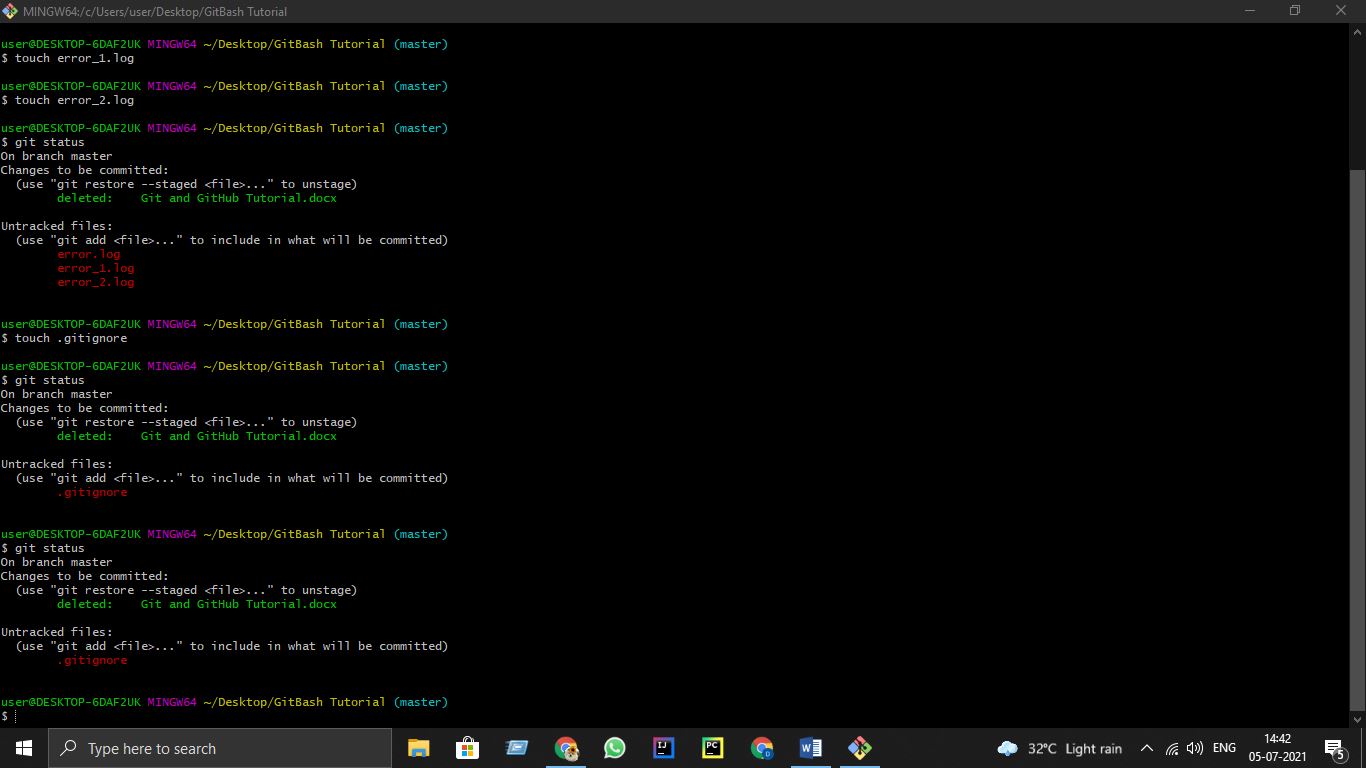
The last image shows that all the staged files are committed now and no more files left to be staged or track ,returned to unmodified state

**git touch filename.extension -** This command is used to make new empty files inside a repo .

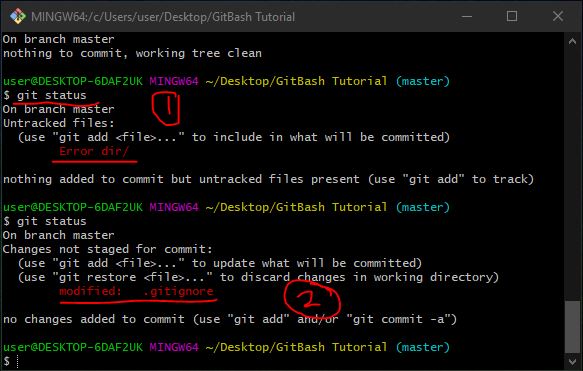
**touch .gitignore :- .gitignore** is an extension of a file which is used to tell git that I want to ignore the files written inside it. i.e during software developments software generate some log files which are of no use to us , but git will show all of them as untracked file. To ignore this we use .gitignore file. inside the .gitignore file we will list up all the unnecessary file names with their extension and save the file , after that if we run the **git status** git will ignore those files.

 **Created some unnecessary log files and git is pointing them as untracked file (.gitignore hasn’t been used)**

Now we will make a .gitignore file using touch command and after creating that we will type the name of these log files inside the gitignore file.

We created a gitignore file and inside that file we typed **\*.log** which will find and ignore all files whose extension is .log.

Now what if we want to ignore an entire folder / directory ?

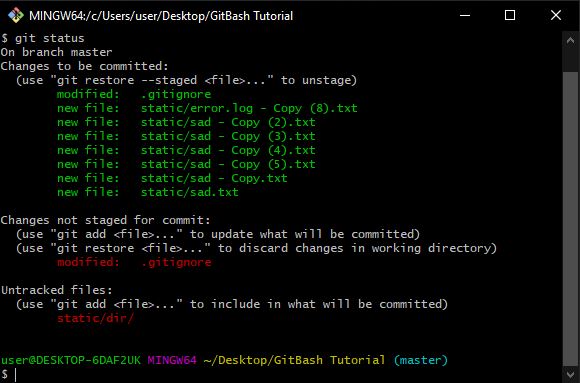
in the previously created .gitignore file we will write the directory name with a suffix “/” (because it is a directory), now if we check git status it will show that the gitignore file has been modified.

**1 –** Shows that there is a new untracked directory in the repository .

**2 –** Added the **Error dir/** into the gitignore file now it only shows that the gitignore file has been modified, but no sign of **Error dir/.** Even if we change something inside the Error dir/ it will not consider it as untracked file/folder .

\*\* **git will always skip or ignore any blank folder , it will not show it as untracked one \*\***

**\*If I want to only ignore the outer directory named “dir” and have a same directory inside of another folder in the repo then I will write “/dir/” instead of “dir/”**

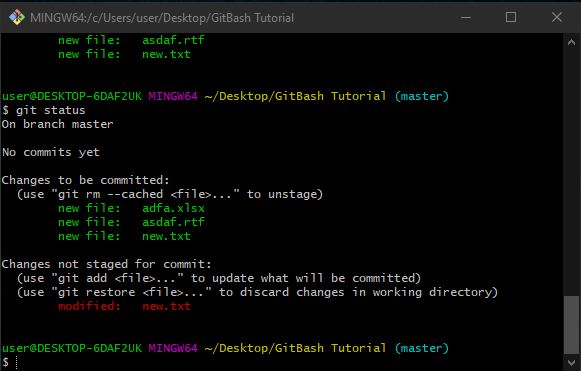
****

here I have a folder named **dir** outside of the folder static , and another **dir** inside the folder static, using **/dir/(in the gitignore file),** instead of **dir/** ignores the **dir** outside but says untracked in the the static folder.

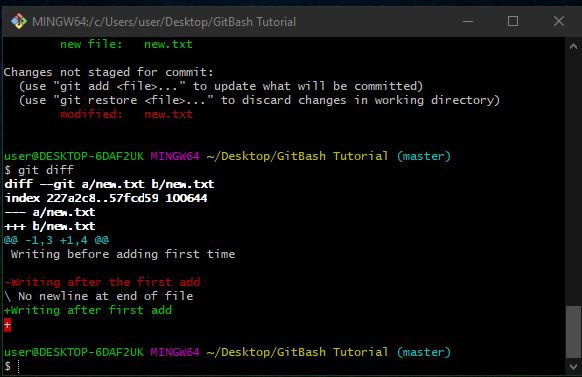
We can also manually do that, if we want to ignore a specific folder inside a folder we will write (in the .gitignore file).

**outside\_folder\_name / the\_foldername\_which I want to ignore.**

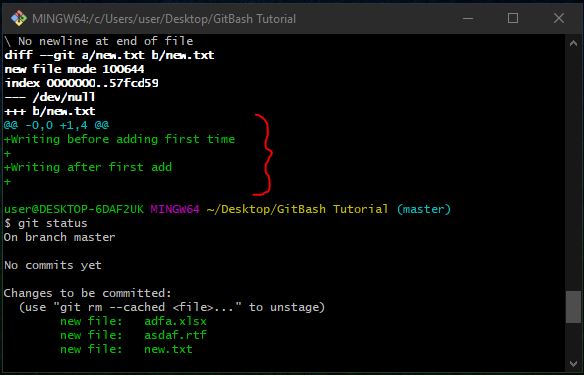
**git diff –** If we change a file while it is in staging area and then run the git status then we see 2 copy of the same file , one is already added and ready to be committed(staged) one is still in the modified zone . This command helps us to differentiate between those two files i.e the changes we have made in the file after putting it in the staging area.

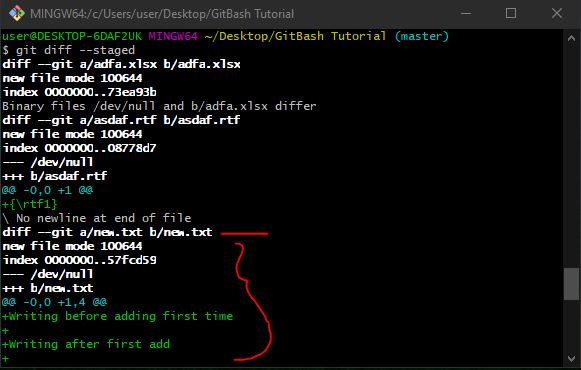


It shows that we have two files of same name , one in staging area and another in modified area , we have modified the file after putting it in staging area.



Now this shows that what changes we have made after putting the file in the staging area.

**git diff - - staged :-** This command compares the files of the last commit with the new files in the staging area.



The images show the changed we have made after the last commit .

**Diff between git diff and git diff - - staged :-**

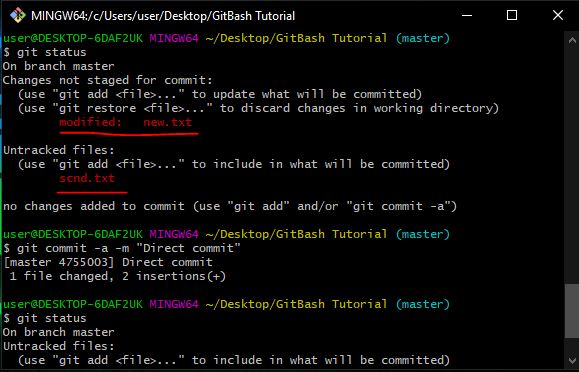
**git diff –** compares the file in the staging area with the modified file in the current working directory.

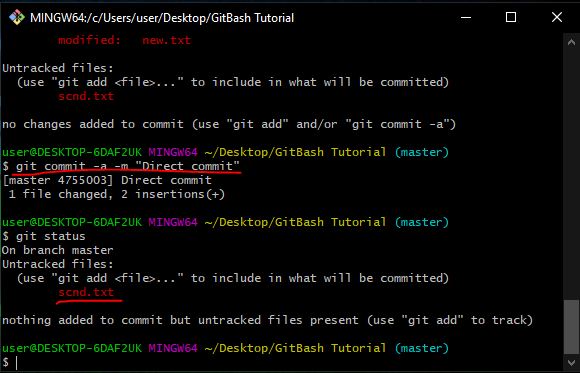
**git diff - - staged** – compares the file of last commit with the file in the current staging area.

**Skipping the staging area (direct commit) :-**

Though it is a very bad practice to directly commit any file without staging it, but for the sake of learning we are doing it.

**git commit -a -m “Your message” –** This command is used to skip the staging area and directly commit a modified/normal file. If the folder contains any untracked file then that file will not be committed, it will remain untracked until we add it. This will only work when the files are been already tracked.

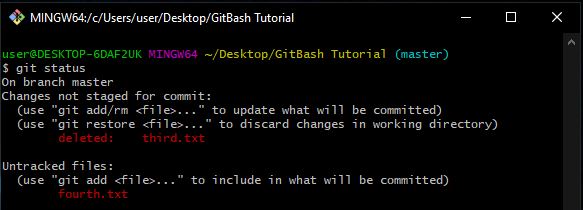


It shows that I have modified a file and there is an untracked file in the repo.

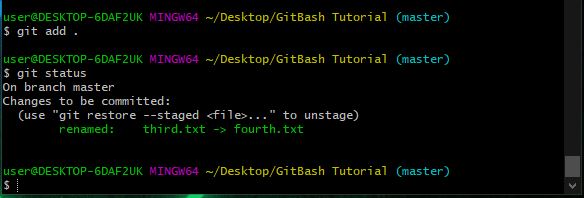
This one shows that after directly committing the file (skipping staging area) we still get the scnd.txt file as it is an untracked file.

**Moving and Renaming files :-**

If we rename a file manually then git will show us that we have deleted the old file and created a new file (until it has been added or staged). After adding the file into the staging area if we check the status then it will show that we haven’t created a new file we just renamed it.

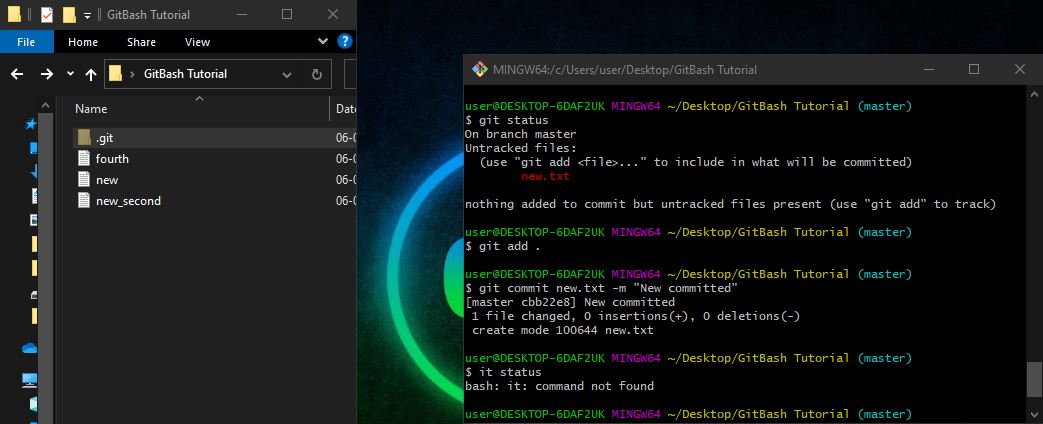


Here I only renamed the file third from fourth manually, but git considers it as a new untracked file.



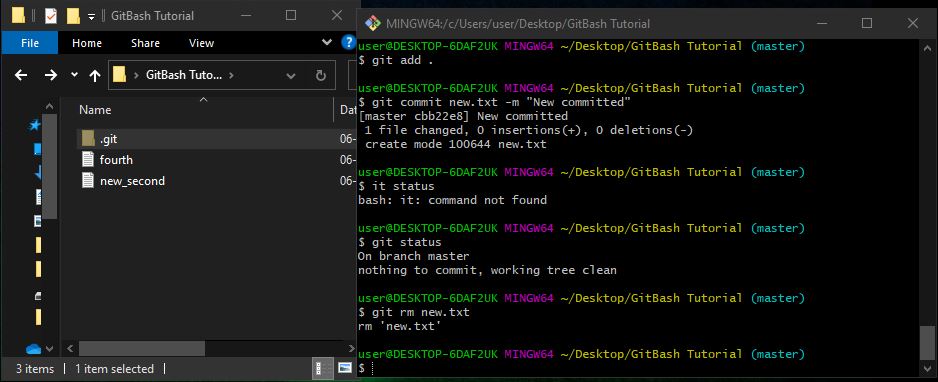
Here after adding the file into the staging area if we check the status then it shows that we have just renamed the file and nothing else.

**git rm filename.extension –** This command is used to delete a file which has been committed earlier . The file we want to delete has to be committed , it will not work until the file has been committed.



here I have created a blank file using the touch command and named it as new.txt and committed it too.

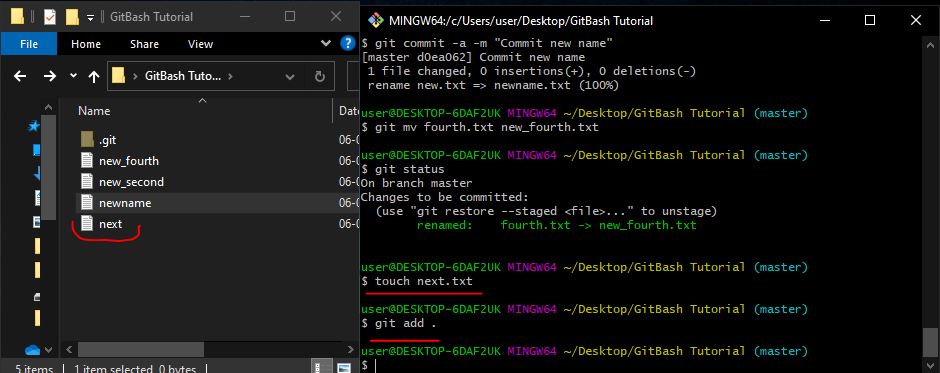
**Now I want to delete that file –**

****

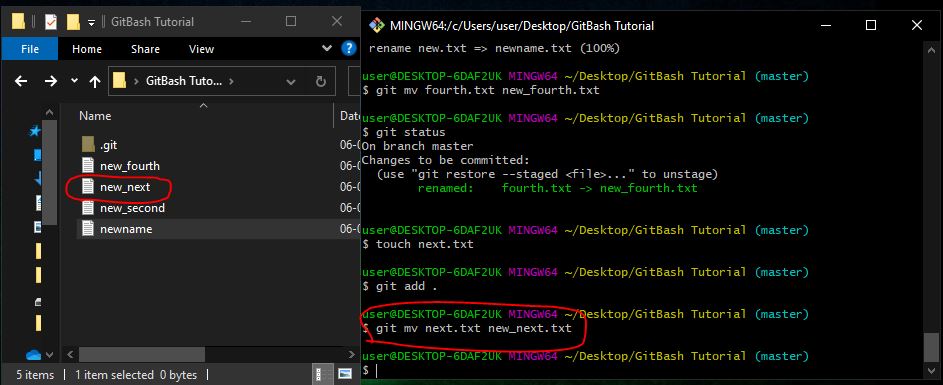
Here using the **git rm** command deleted the new.txt file which was been committed earlier.

**git mv oldfilename.extension <space> newfilename.extension :-**

This command is used to rename a file using git. The file must be tracked or added in the staging area (It will work if it is not committed yet). It will also work if it has been committed.



Created a new file named next using touch command and added it in staging area.



Now renamed the file using git mv command. The file was already staged but not committed (It works both ways), but it would become a problem when we have a large number of files , we will not be able to identify the file we committed if we rename it after the commit.