GIT

Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

Listed below are the functions of a VCS:

Allows developers to work simultaneously.

Does not allow overwriting each other’s changes.

Maintains a history of every version.

Following are the types of VCS:

Centralized version control system (CVCS). (Not good since 'single point of failure')

Distributed/Decentralized version control system (DVCS) (DVCS clients not only check out the latest snapshot of the directory but they also fully mirror the repository. If the server goes down, then the repository from any client can be copied back to the server to restore it. Every checkout is a full backup of the repository.)

Git does not rely on the central server and that is why you can perform many operations when you are offline. You can commit changes, create branches, view logs, and perform other operations when you are offline.

**Advantages**:-

* Free and Open Source- Source code can be downloaded and changes can be made accordingly
* Fast and Small – Speed since most actions are performed locally. Though Git mirrors entire repository, the size of the data on the client side is small. This illustrates the efficiency of Git at compressing and storing data on the client side
* Security- Git uses a common cryptographic hash function called secure hash function (SHA1), to name and identify objects within its database.
* No need for powerful hardware- The Devs don’t interact with server unless for pushing/pulling code. The actions are generally on the client side system hence the server hardware can be simple.
* Easier Branching- Takes less time compared to CVCS (uses cheap copy mechanism i.e. any new branch created from and older one copies all the existing content to the new branch). Creating, deleting and merging branches takes less time in Git.

After installing GIT we use the **‘git init’** command in the source location. This cmd is used to initiate tracking folders and files. This creates a **‘.git’** folder at the location.

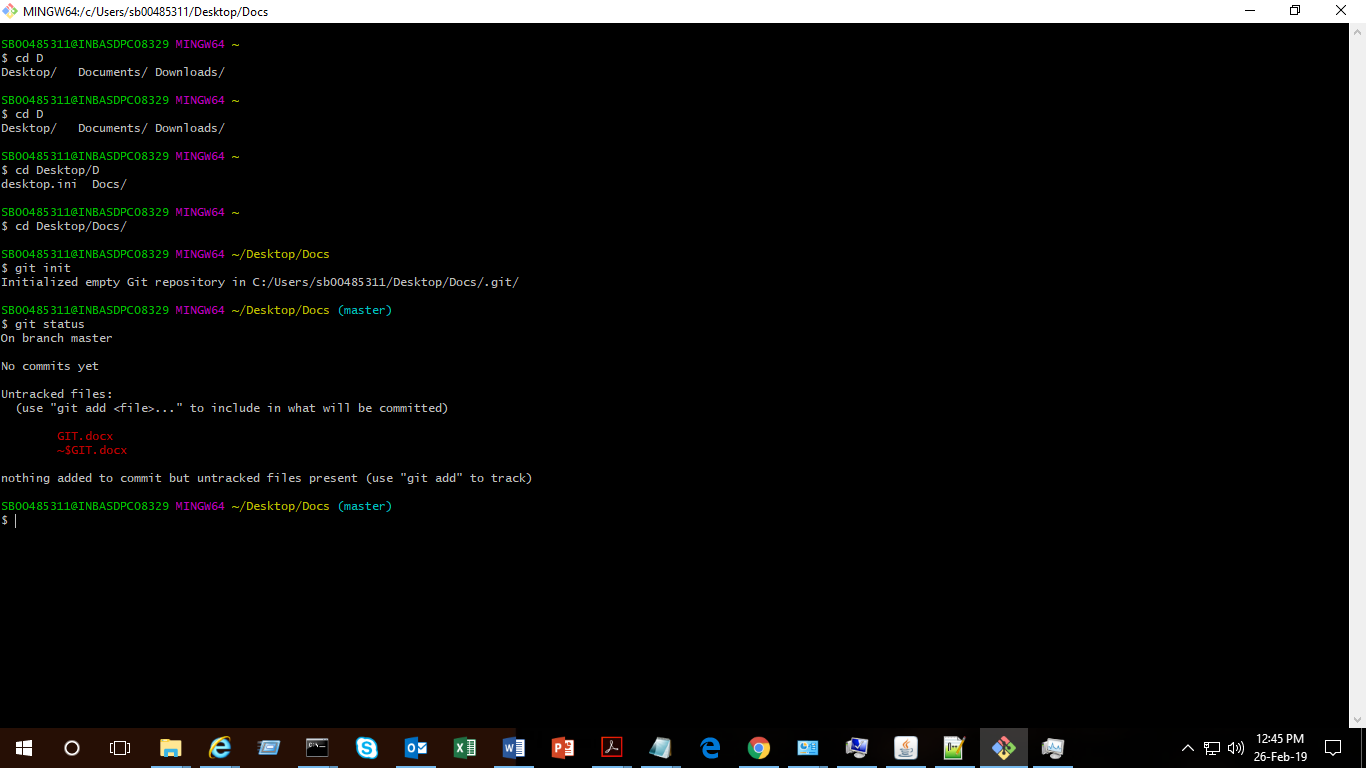


Fig1: gitbash UI for initializing and checking the status of new changes

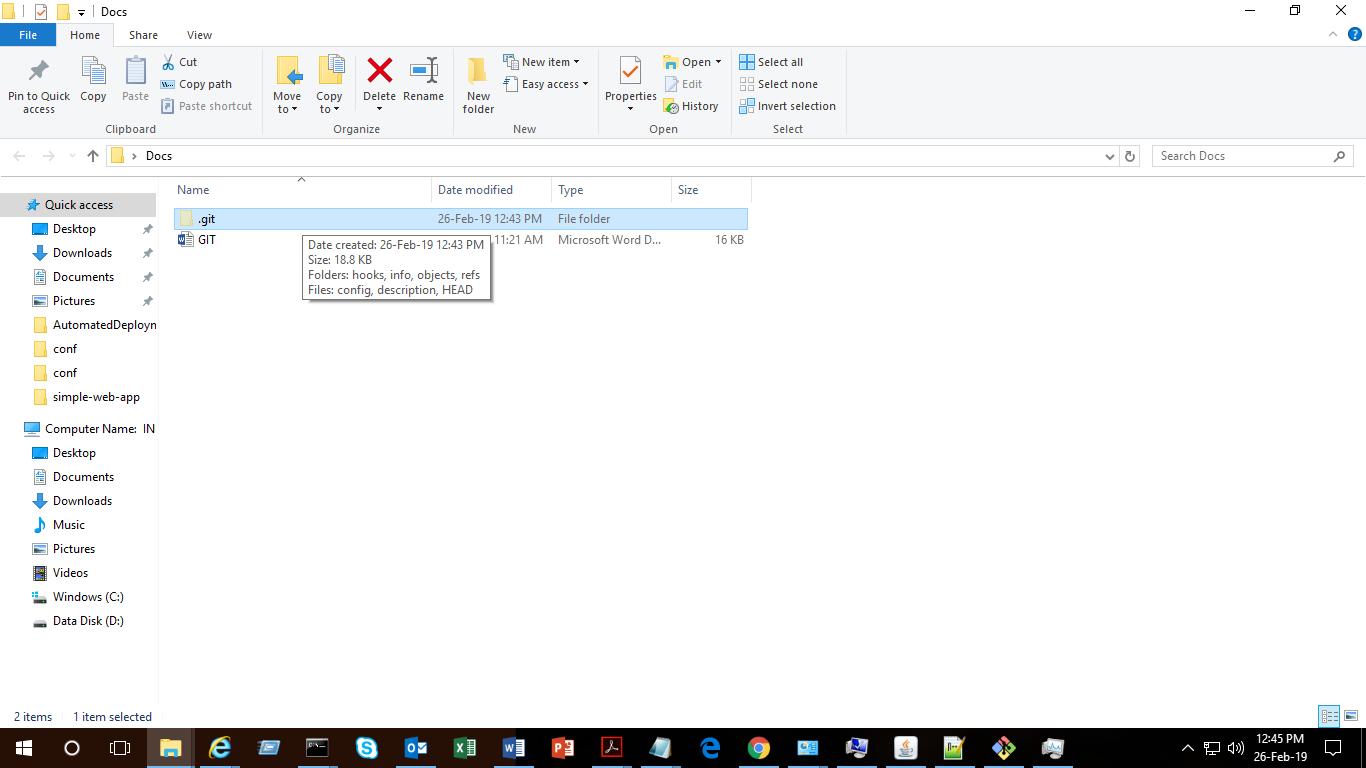


Fig2: .git folder is formed at the source location which contains the info on the repo

Using the **‘git status’** cmd we can see the status of the files at the location. It tells us about the changes that haven’t been added to staging yet. (Can be seen in the Fig1)

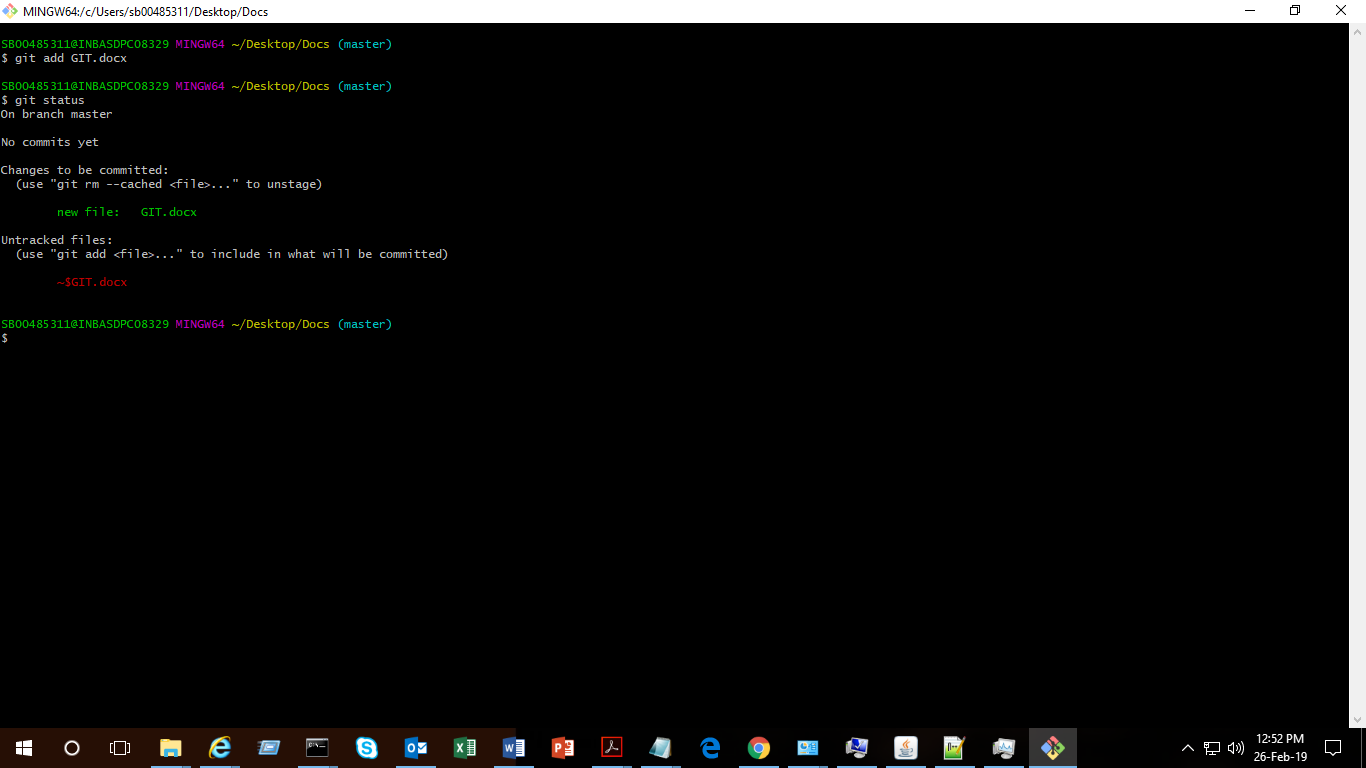


Fig2: using **‘git add \*’** cmd

We use the ‘git add’ cmd to add the files.

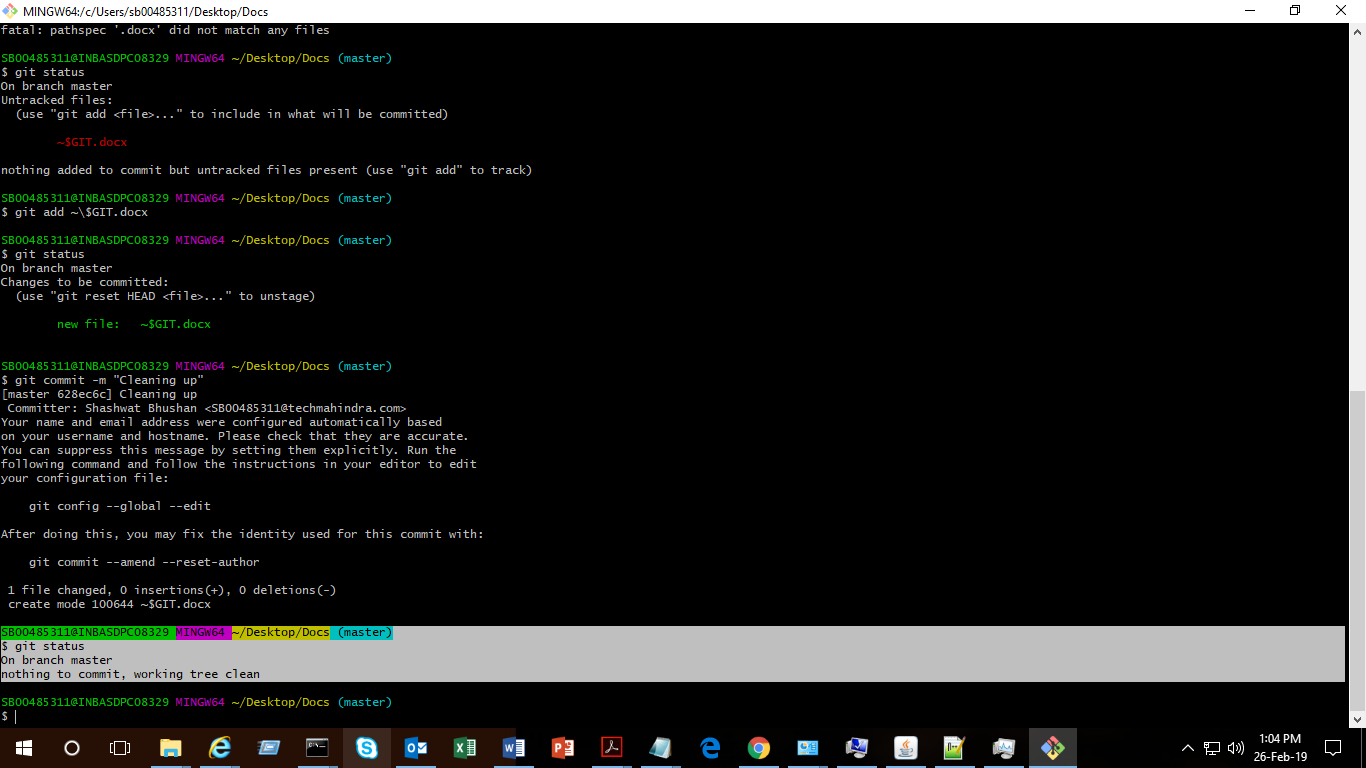


Fig3: Using **‘git commit’** command

Add the folder to the remote repository using the cmd **‘git remote add origin** [**https://github.com/ShashwatBhushan29/Repo.git**](https://github.com/ShashwatBhushan29/Repo.git)**’** and **‘git push -u origin master’**

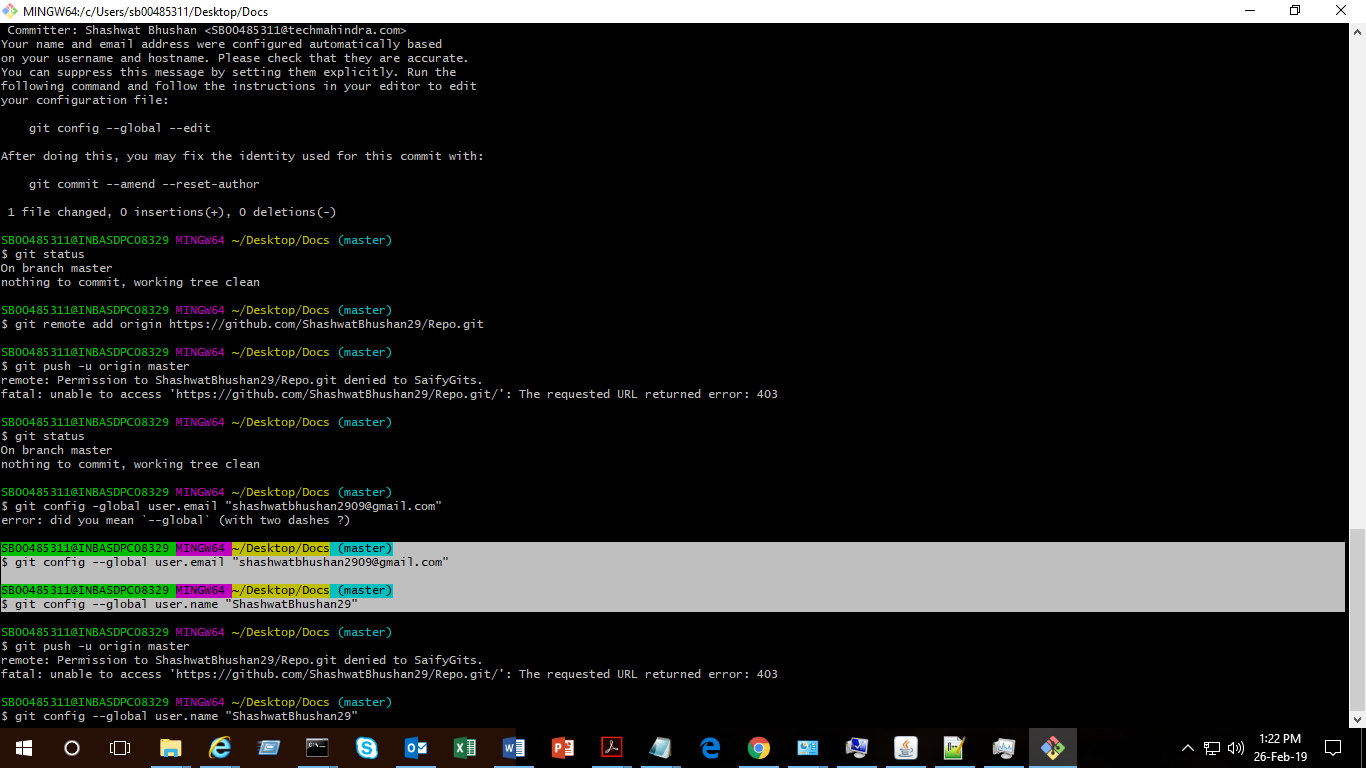


Fig4: Configuring the user name and email for git bash using **‘git config --global user.email** [**shashwatbhushan2909@gmail.com**](mailto:shashwatbhushan2909@gmail.com)**’** and **‘git config --global user.name "ShashwatBhushan29"**’

**Note:** the user name and email id should be the same as used for GitHub account

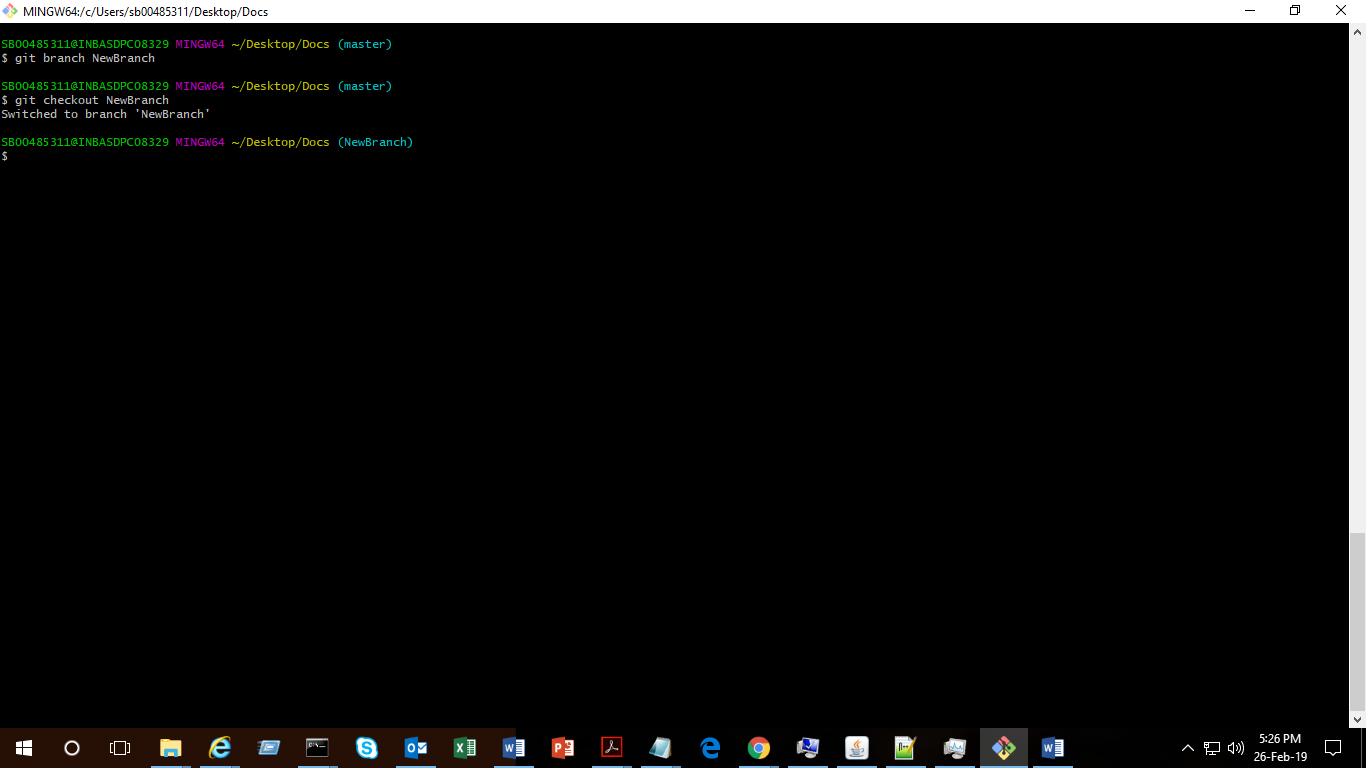


Fig5: Create a new branch and check out to that branch. Cmds- **‘git branch “BranchName”’**  and **“git checkout “branchName”**

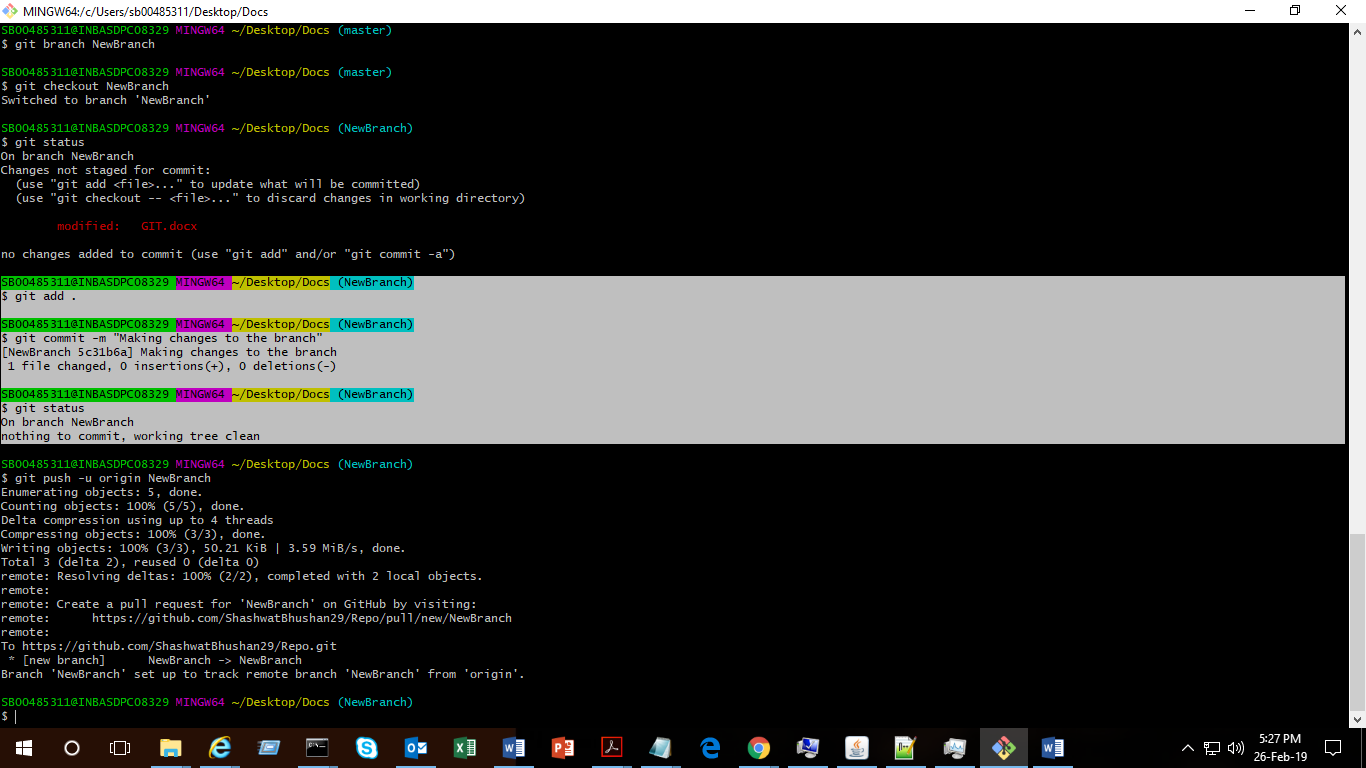


Fig6: After making the changes and pushing the content to the new branch you will be able to see the branch in the GitHub UI. Add the files to staging and push to the branch using the cmd **‘git push –u origin “branchName”’.**

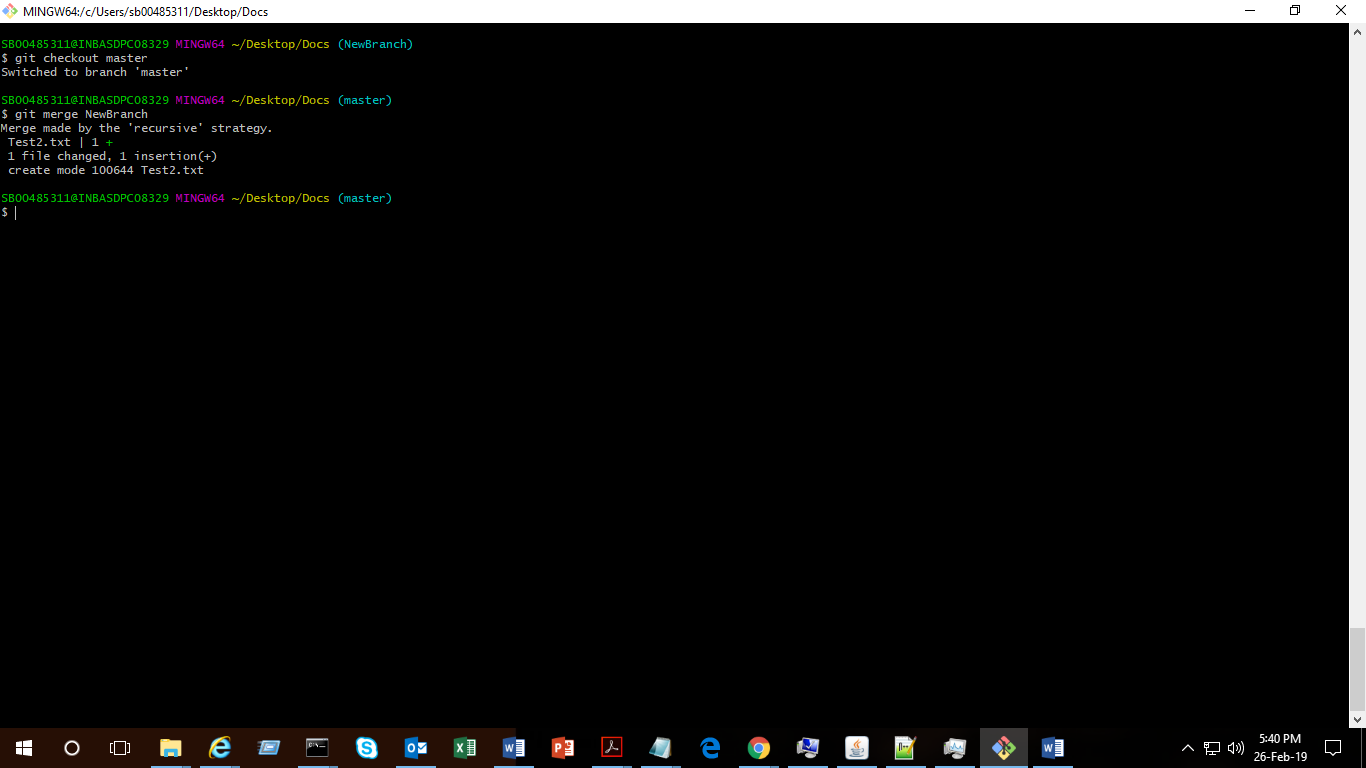


Fig7: To merge the new branch to master branch **checkout to the master branch** and use the **‘git merge “branchName”’** to merge the changes in new branch to master

Push the changes on master also after the merging is done. Only then will it show on the GitHub UI.

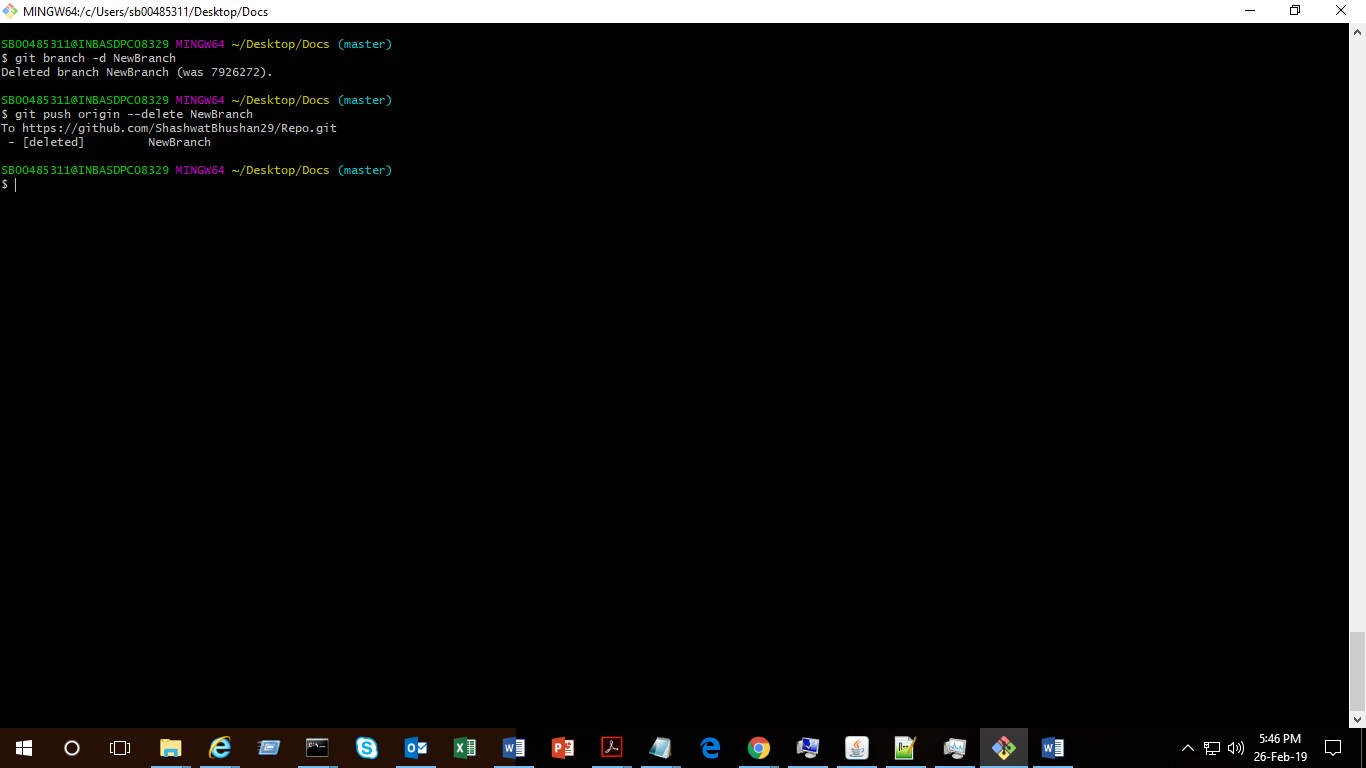


Fig8: After the use of new branch is completed we can delete it. Cmd- **git branch –d NewBranch** (deletes from local only)

Use the command **git push origin --delete NewBranch** to remove the branch from remote repo

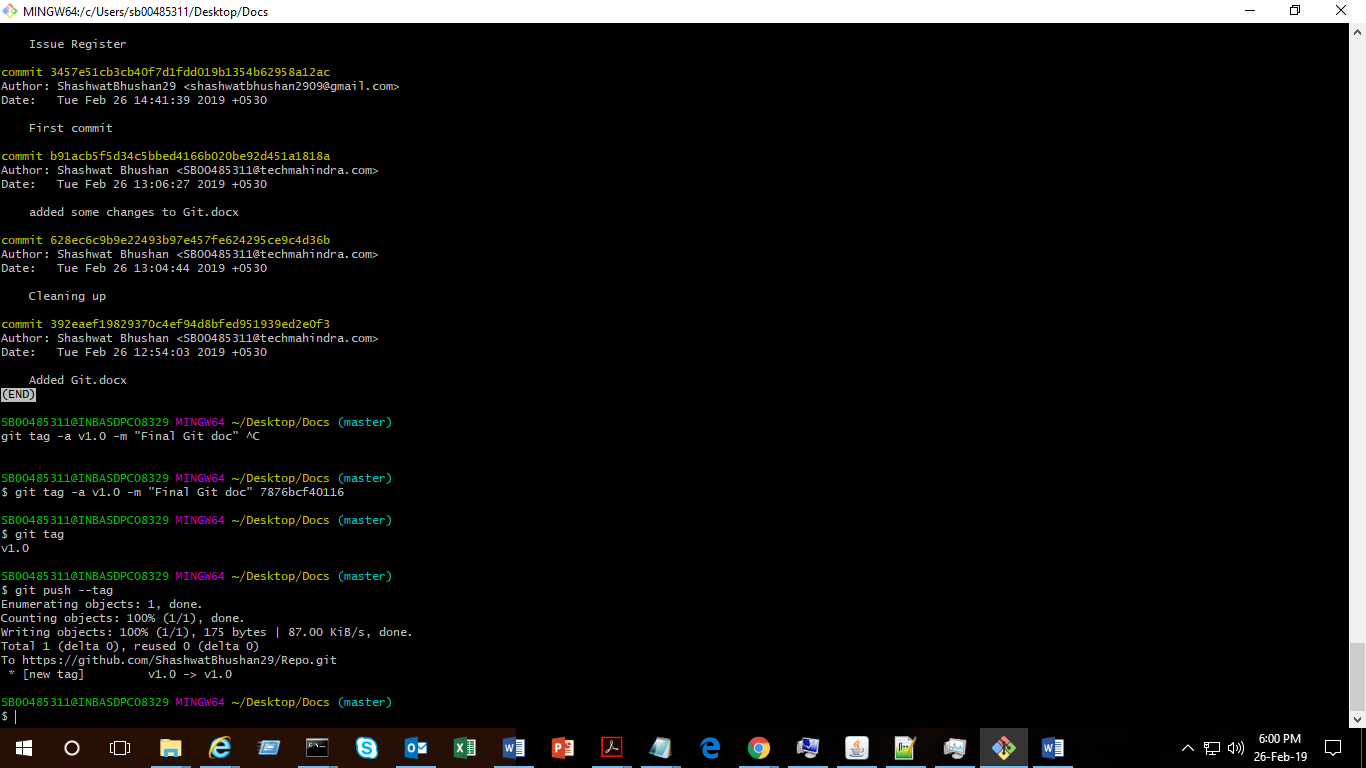


Fig9: Creating tags. Cmd- **git tag –a v1.0 –m “Message” 13qwer**

Here it isn’t necessary to use –a or –m . ‘-a’ is for an annotated tag with a message following up after ‘-m’. **13qwer** will be the SHA1 code for a particular commit.

Using **git tag v1.0** creates a tag without the annotations.

Note: The tag should also be pushed to the remote repository with the cmd **‘git push tag’**