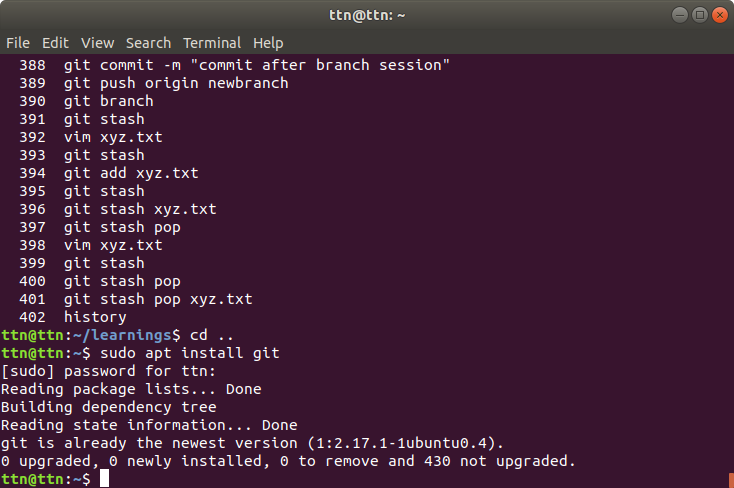
**Exercise for Version Control and Git**

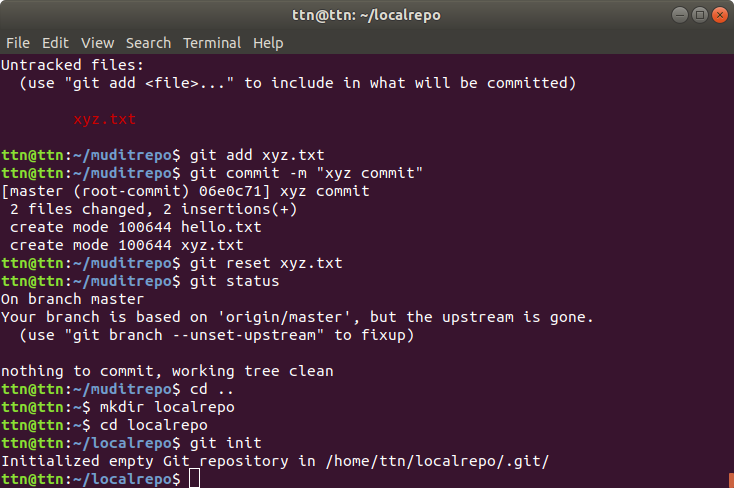
**1. GIT Setup**

Ans : sudo apt install git



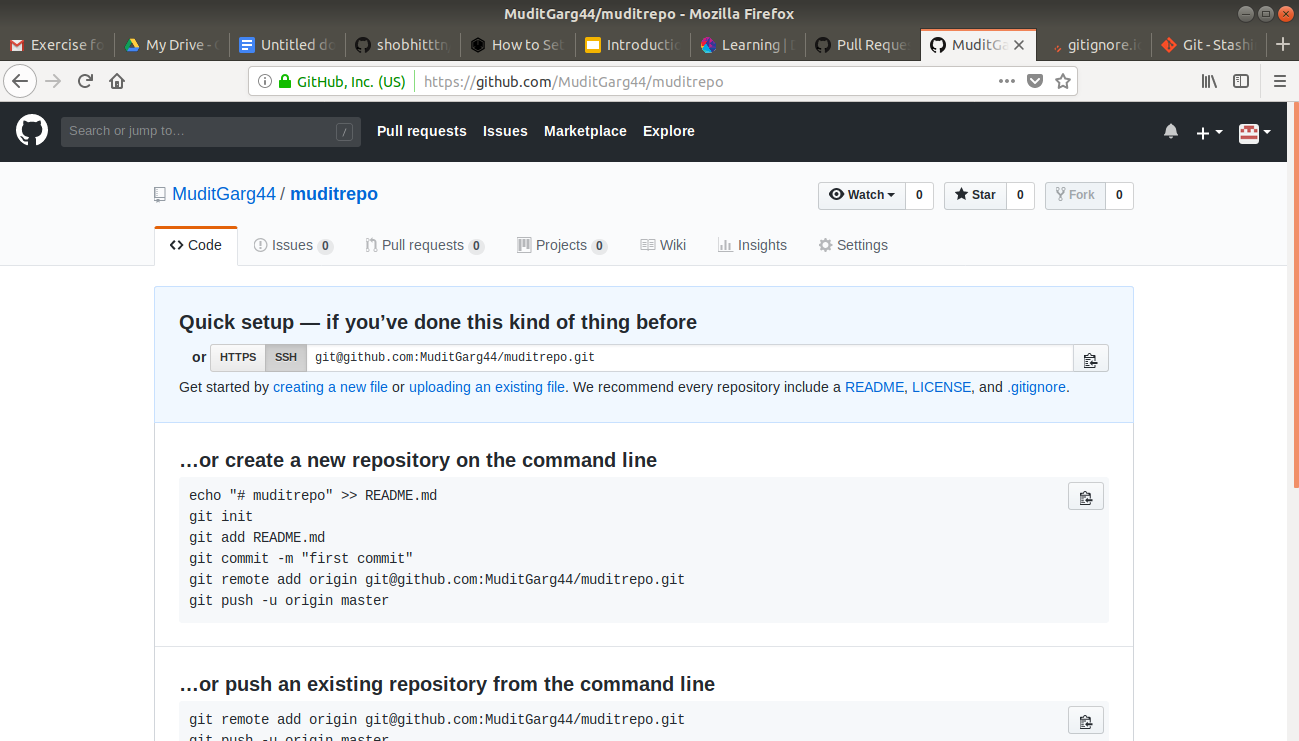
**2. Initialize a Git Repository**

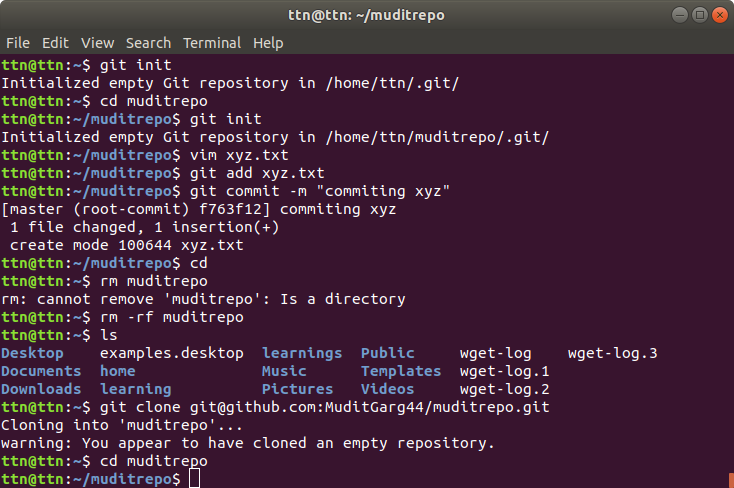
Ans : Initializing a local repository in screenshot using git init.



**OR**

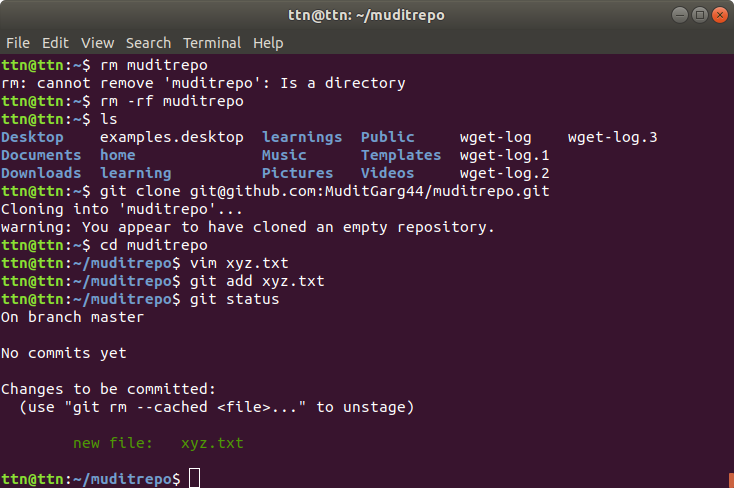
A remote repository is added on GitHub account and it is cloned into local terminal using SSH url(as we learned in our training session).





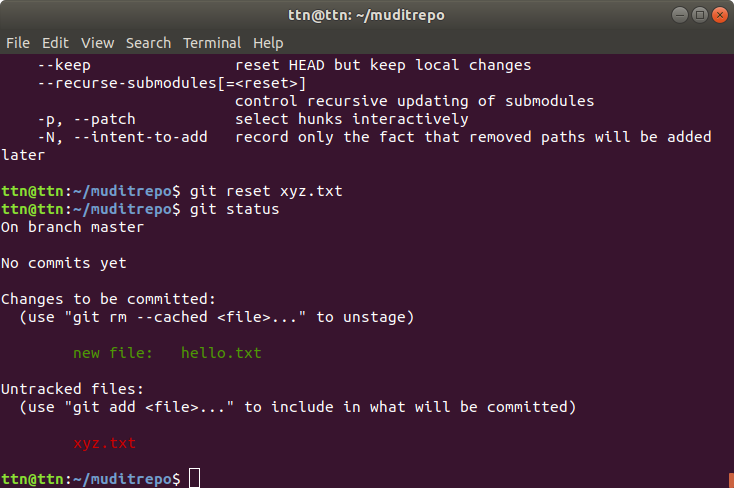
**3. Add files to the repository**

Ans :



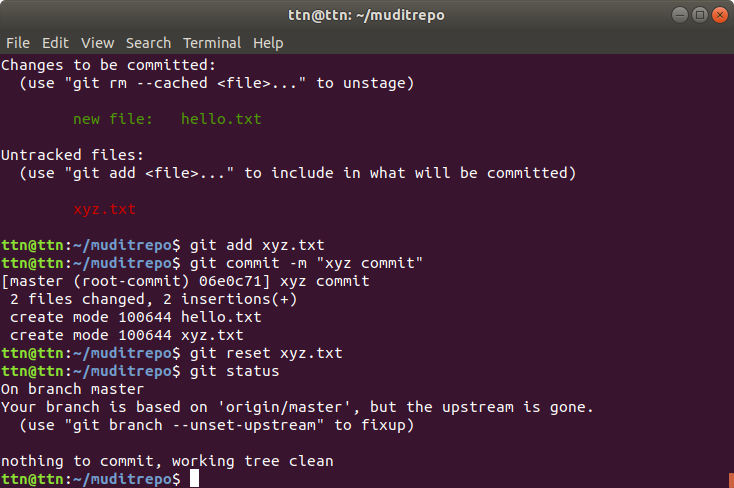
**4. Unstage 1 file**

Ans :



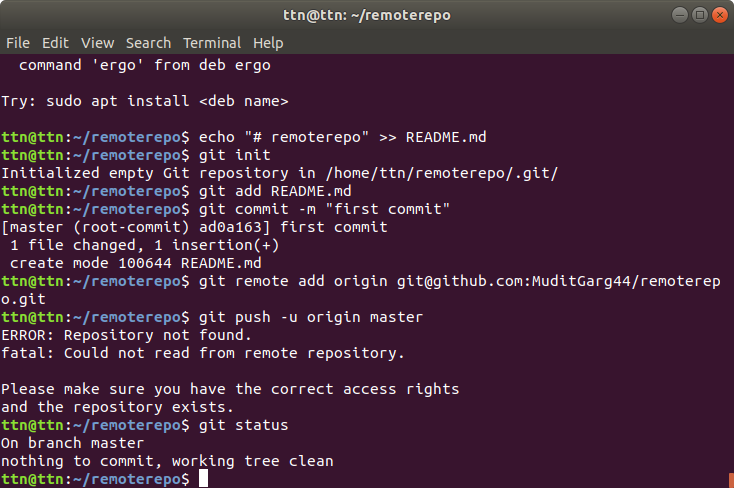
**5. Commit the file**

Ans :



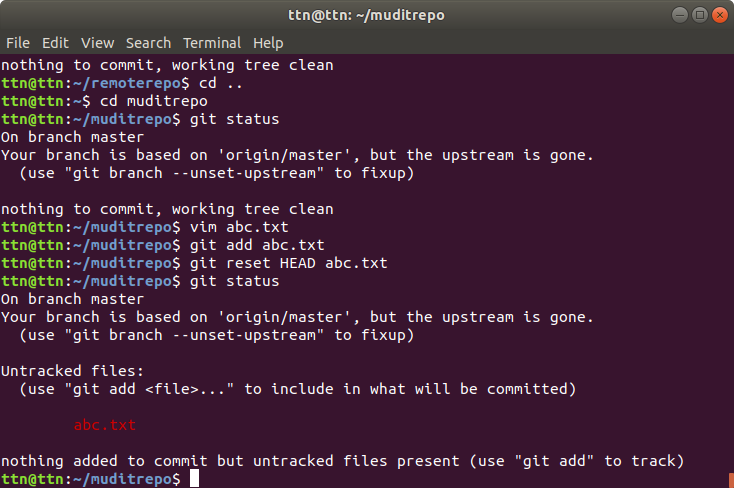
**6. Add a remote**

Ans : A new repository is created from command line then added as remote.



**7. Undo changes to a particular file**

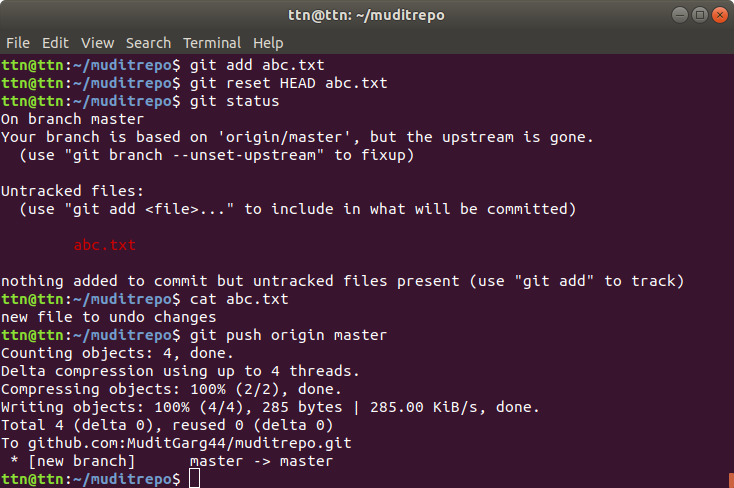
Ans : Undo changes to a staged file using reset option



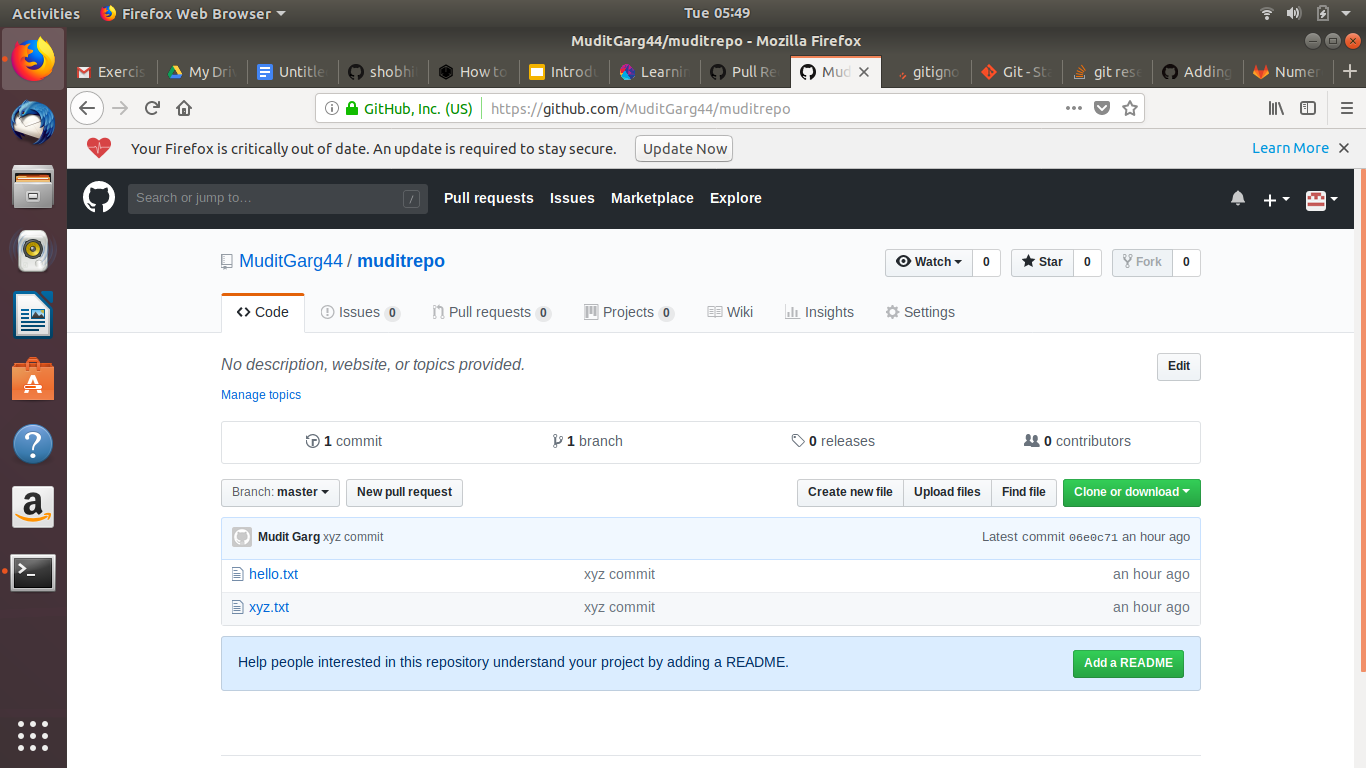
**8. Push changes to Github**

Ans : I had 2 files xyz.text and hello.txt already committed in muditrepo repository.

These committed files are pushed using git push command.

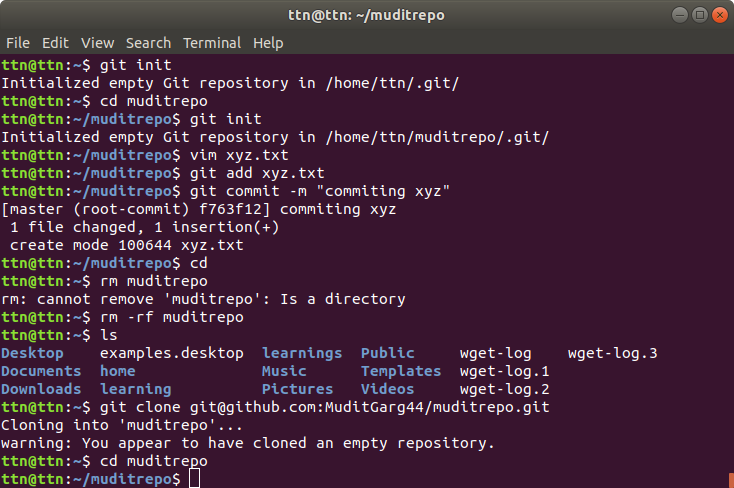


Pushed files on Github Repository :



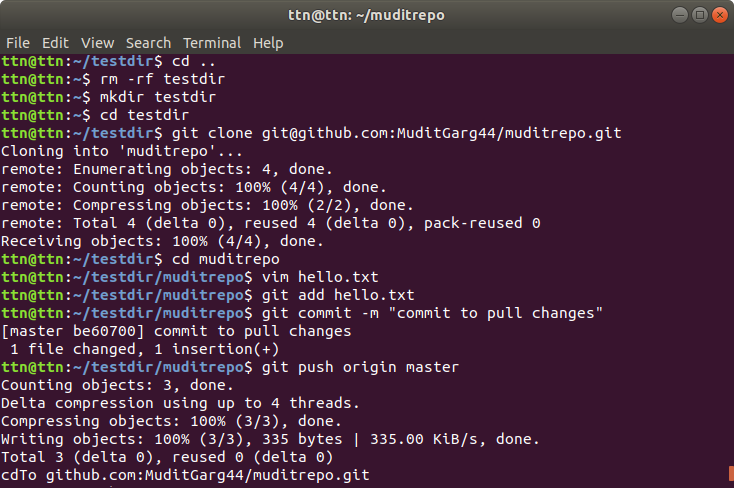
**9. Clone the Repository.**

Ans : Cloning the repository :

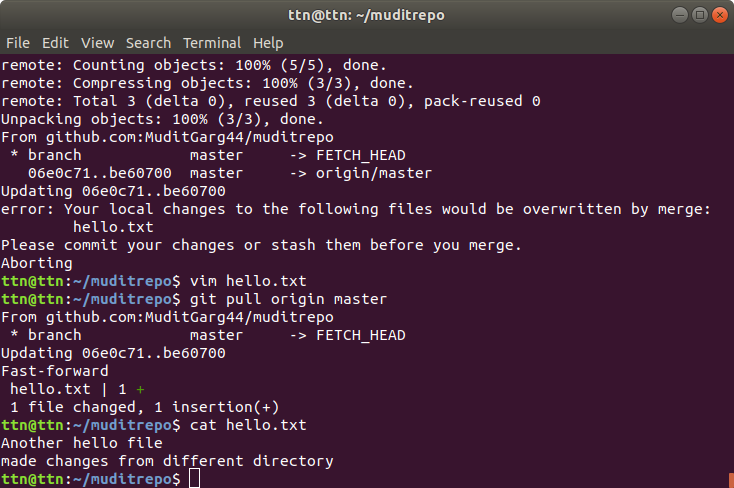


**10. Add changes to one of the copies and pull the changes in the other.**

Ans . I have created another directory as “testdir” and made clone of muditrepo repository in it.

Then from here, i made a new hello.txt file, updated changes into it, added, committed and pushed it to remote. 

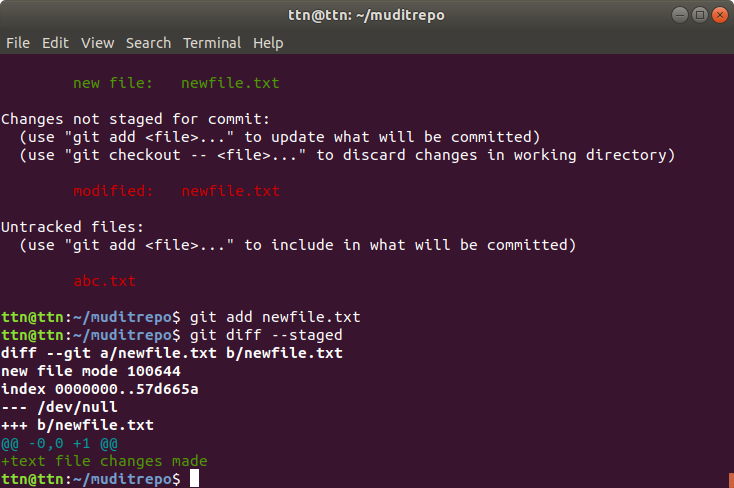
Later, i went back to my original repository directory muditrepo and used command git pull origin master to pull changes from remote. The update hello.txt file was saved onto local machine.



**11. Check differences between a file and its staged version.**

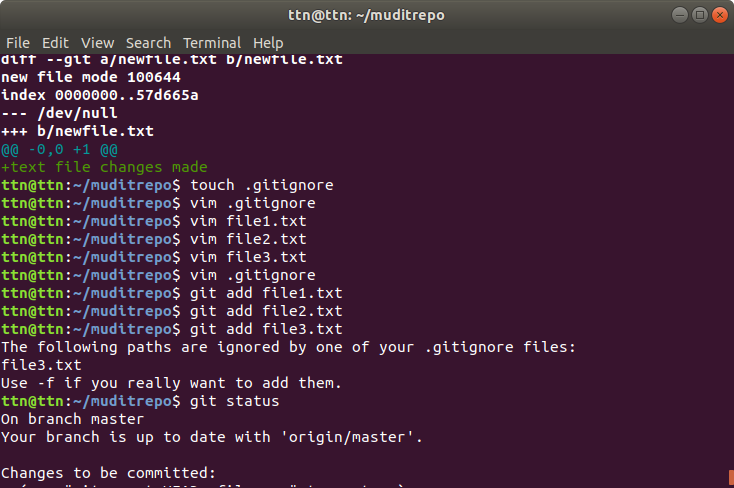
Ans : A new file “newfile.txt” has been created ad staged.

Then git dif --staged is used to show changes that have been staged after git add is used.



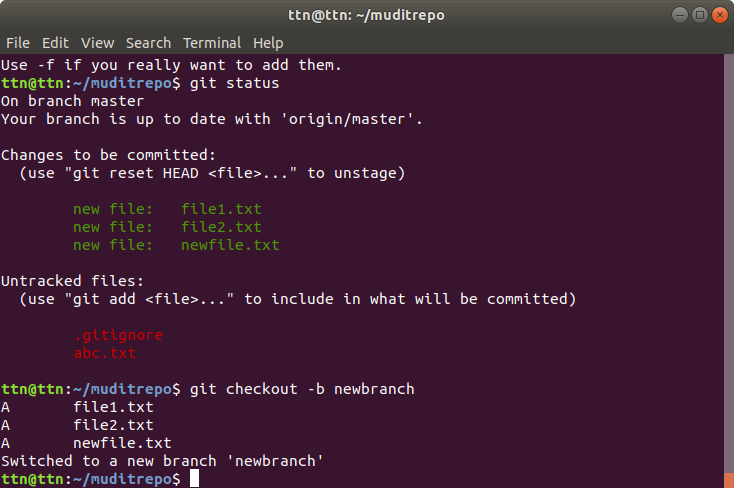
**12. Ignore a few files to be checked in.**

Ans : create a .gitignore file in project repository. Then add those file names in .getignore that you want to be ignored while commiting. In my case, .gitignore even prevented a file to be staged



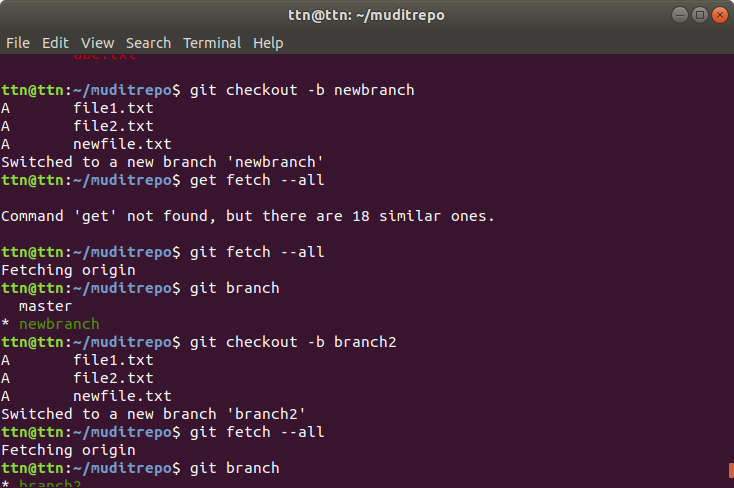
**13. Create a new branch.**

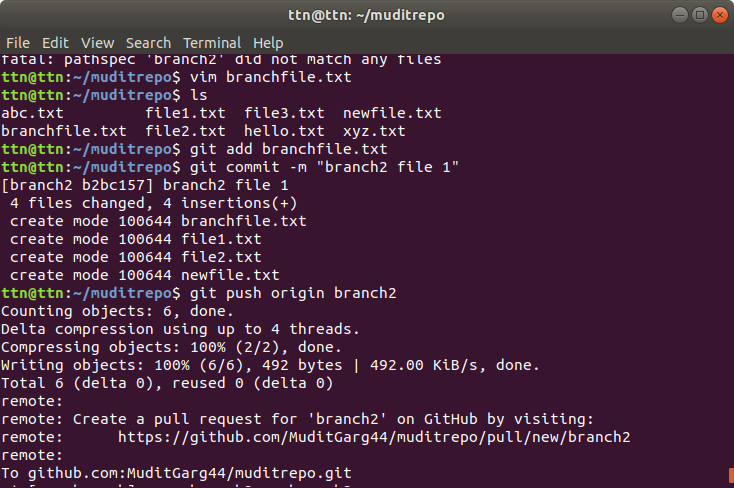
Ans : -b option creates a new branch. “Newbranch” is name of newly created branch. Checkout is used to instantly navigate to newly create branch.

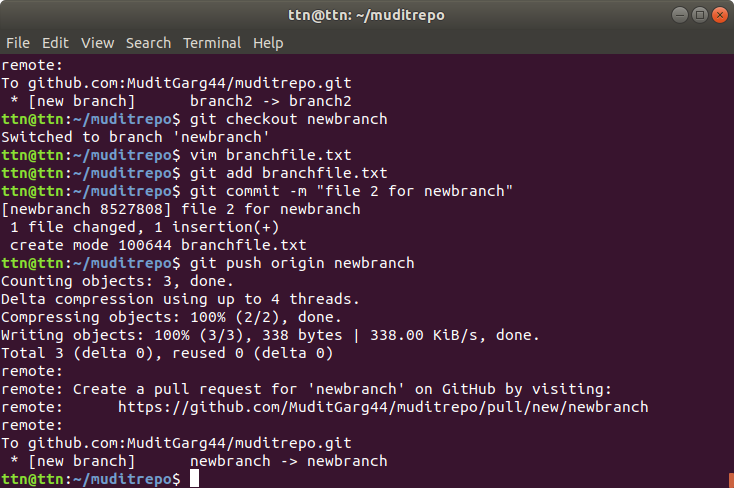


**15. Edit the same file at the same line on both branches and commit**

Ans : I have created 2 different branches as “newbranch” and “branch2” apart from master branch. Then a file with same name is created in these 2 branches with same data on 1st line in each.

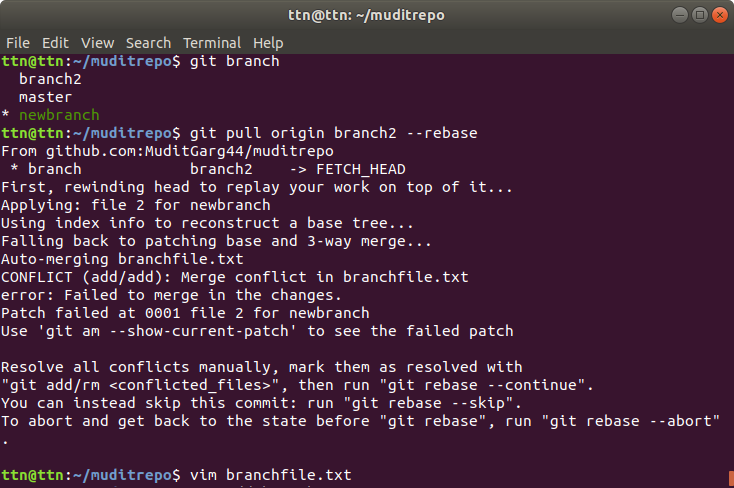




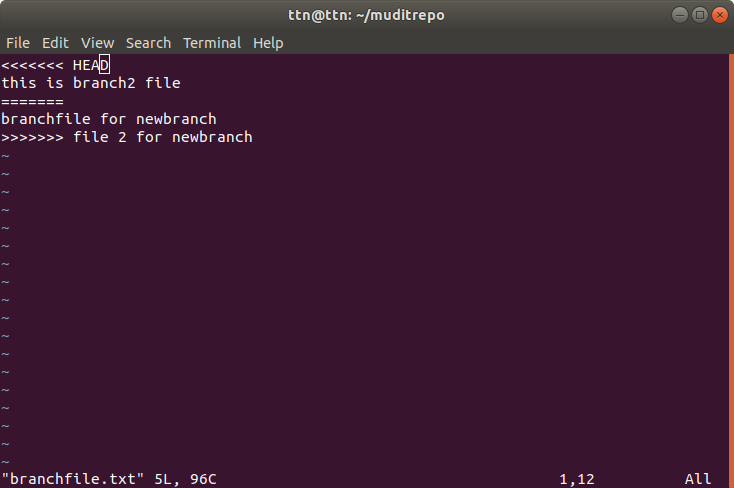


**16. Try merging and resolve merge conflicts**

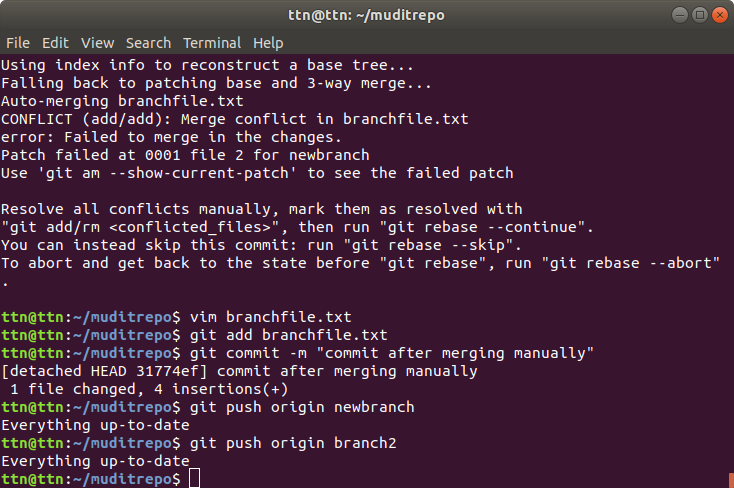
Ans : branchfile from branch2 was pulled into newbranch and auto merge failed due to conflicts.



Conflicts were handled manually :

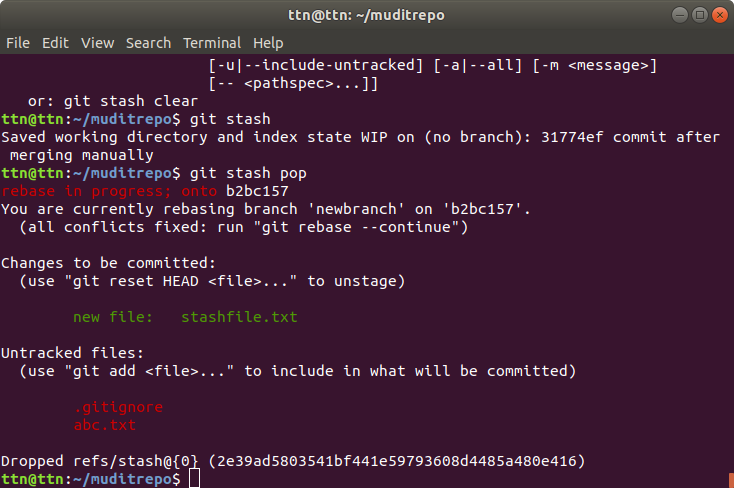


Then merged and commited :



**17. Stash the changes and pop them**

Ans : git stash is used when a user doesn’t wants to commit changes but wants them to be saved temporarily so that they can be resumed later. Stash pop is used to continue from where user left off. Or in technical terms, stash pop throws away topmost stash by applying it, which means that the saved stash cannot be reused.



**18 . Add the following code to your .bashrc file : color\_prompt="yes"**

**parse\_git\_branch() {**

**git branch 2> /dev/null | sed -e '/^[^\*]/d' -e 's/\* \(.\*\)/(\1)/'**

**}**

**if [ "$color\_prompt" = yes ]; then**

**PS1='\u@\h\[\033[00m\]:\[\033[01;34m\]\W\[\033[01;31m\] $(parse\_git\_branch)\[\033[00m\]\$ '**

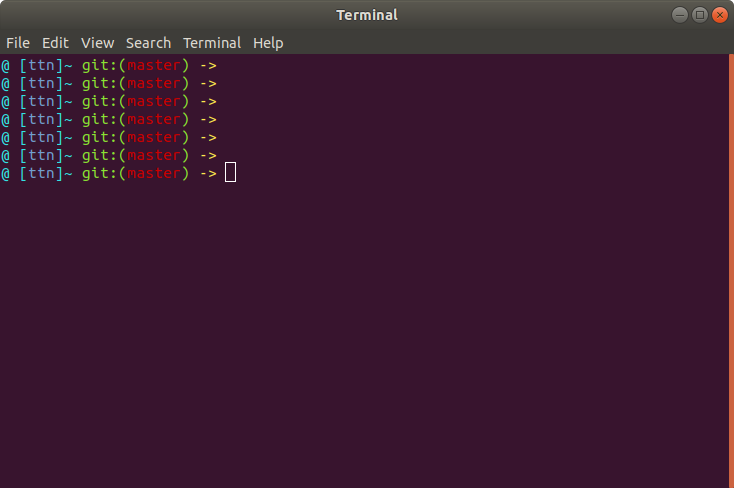
**else**

**PS1='\u@\h:\W $(parse\_git\_branch)\$ '**

**fi**

**unset color\_prompt force\_color\_prompt**

Ans :



**Doubt question :**

4. “Diverge branches with commits.”

Whats the meaning of diverging the branches and its achieved?