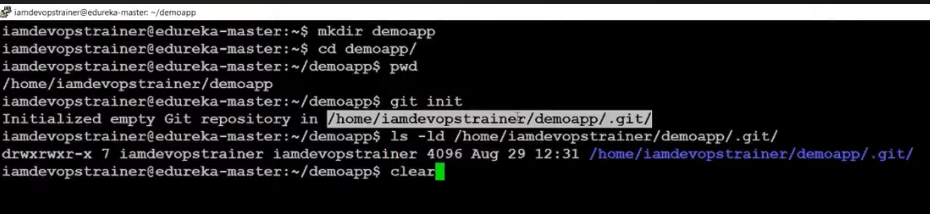
**GitHub Workflow:**

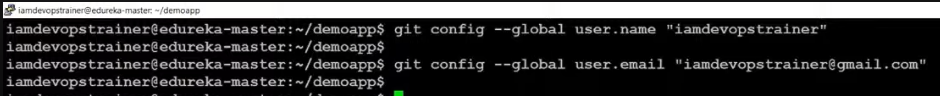
Diagram

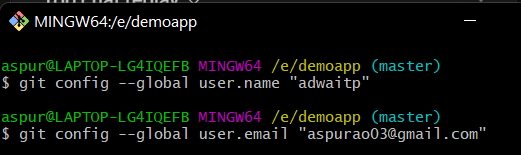
Description automatically generated



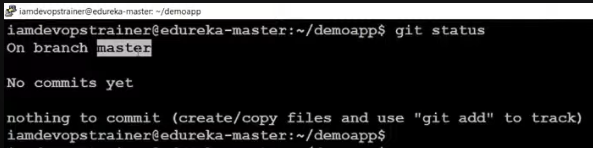


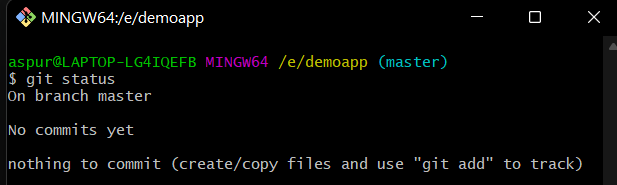
**Git Configure:** Used to track any changes made to the repo along with the developers name and email address so that other users can contact the developer via it

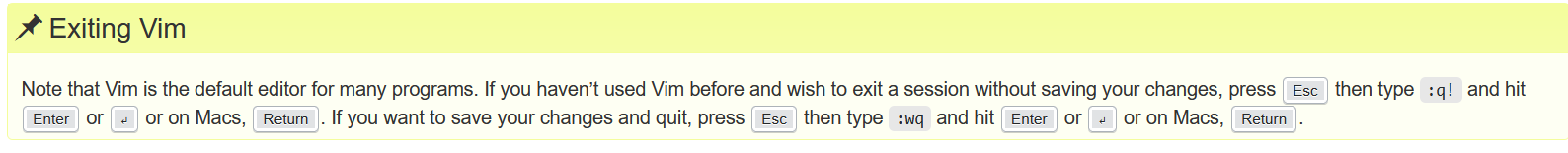


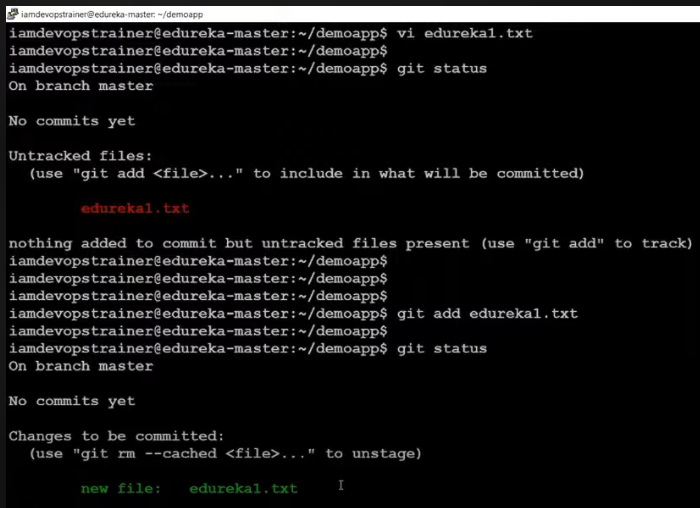


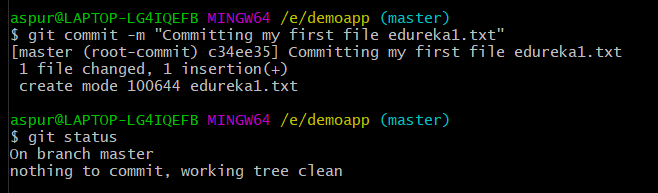
Master is the default branch which is created when you initialize a repository.

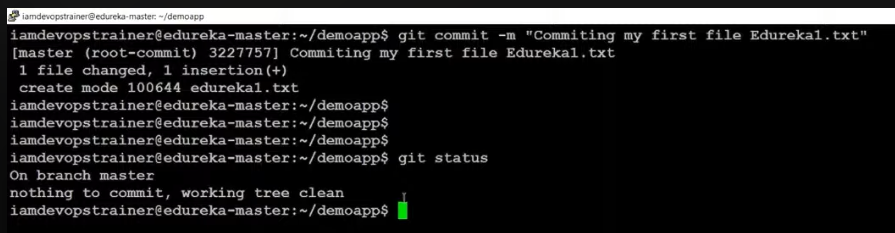


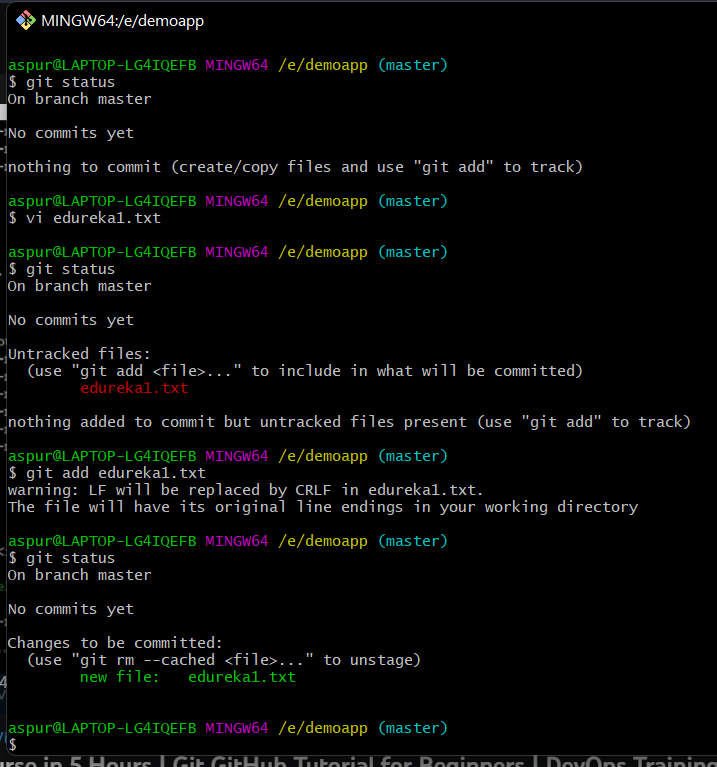








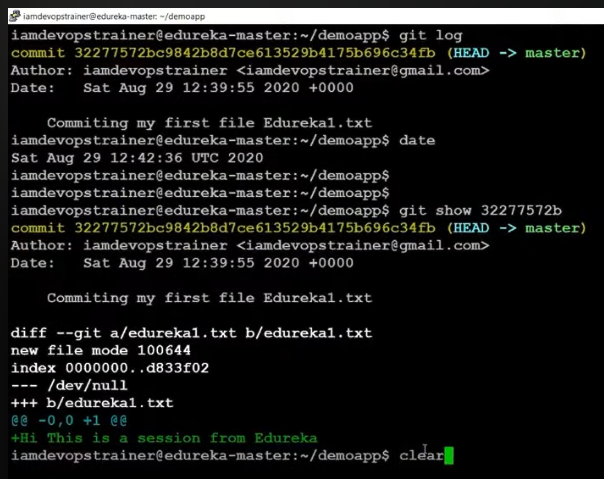




**Git Log:** Gives List of all commit messages made in the directory.

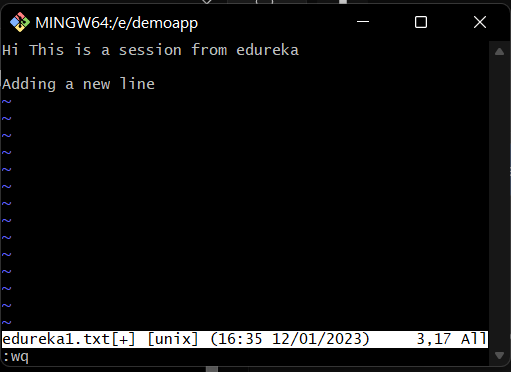
**Git Show:** Check the details related to any commit message.

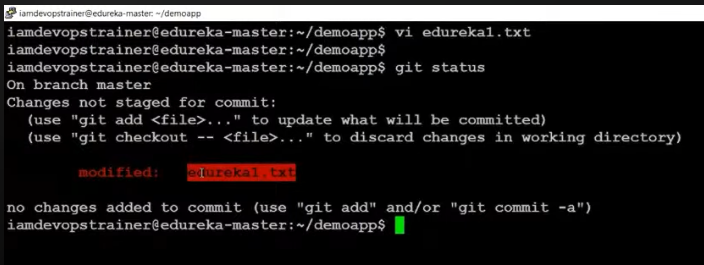
**Syntax:** git show <commit id 7-8 characters>





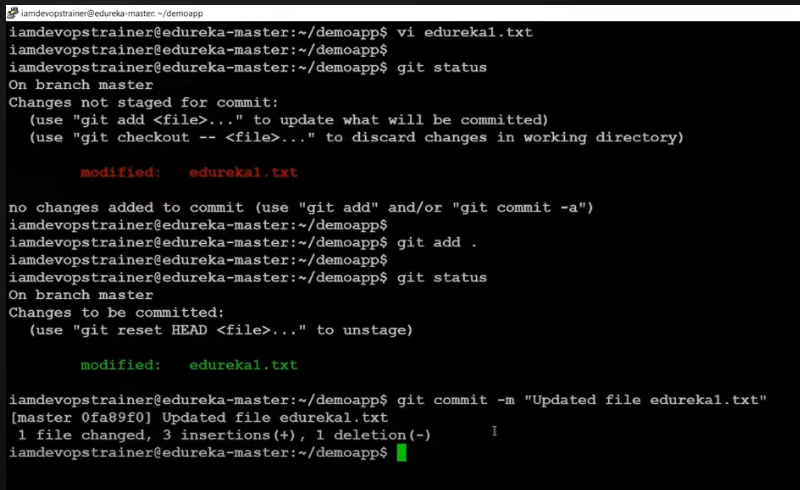
Changes made

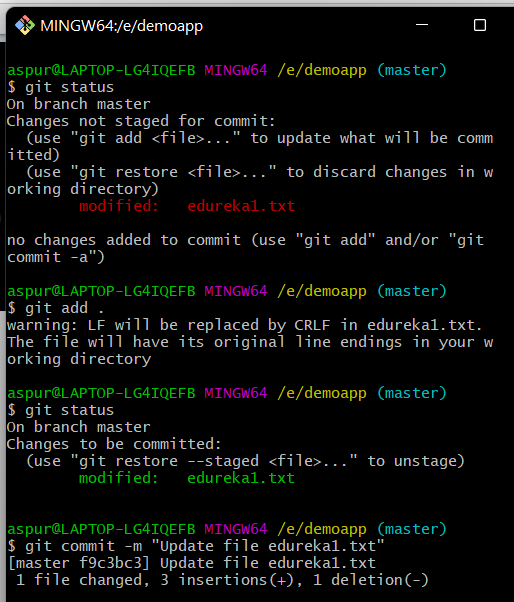




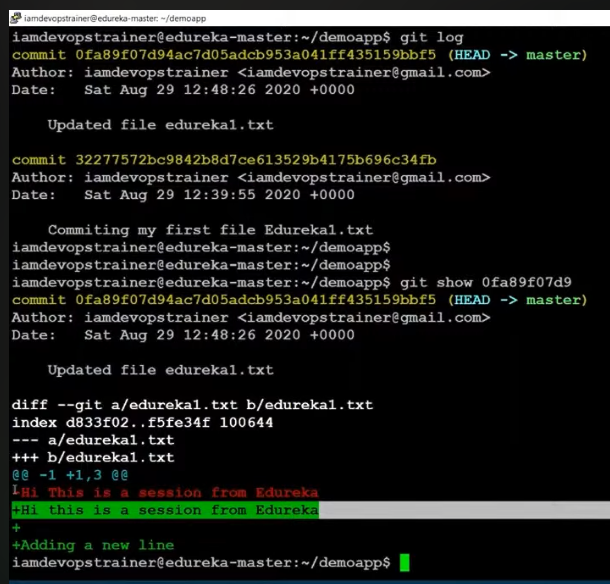
To move all files from working directory to staging area :

**Syntax:** git add .

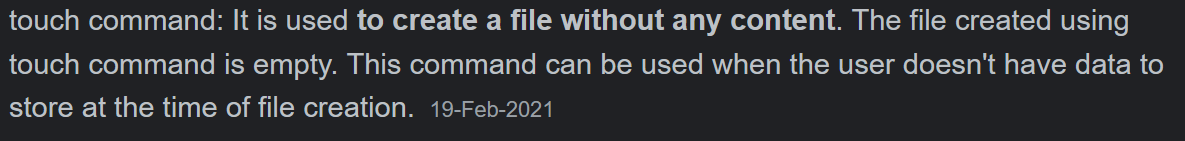


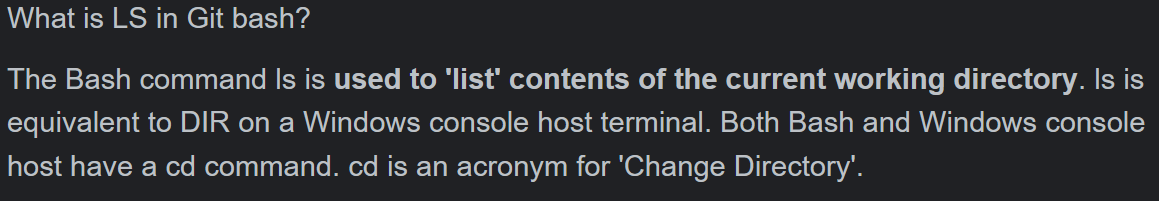


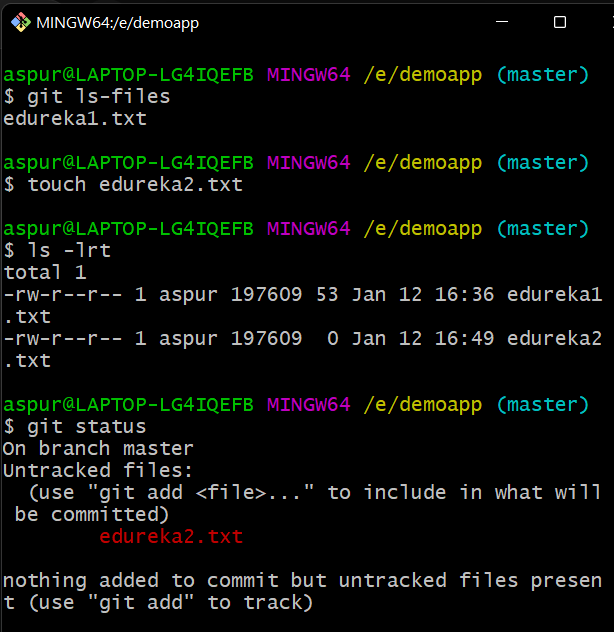
While using git log the previous file is shown in red with a ‘-’ sign and new file i.e. the changes are shown with green colour and ‘+’ sign

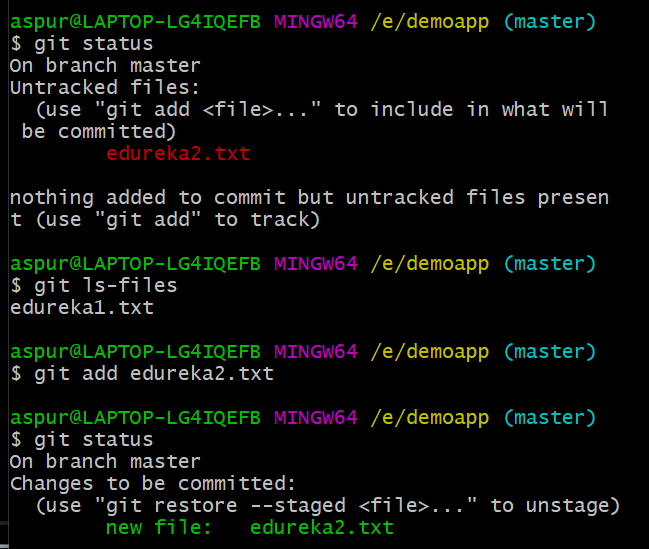


**Git ls-files:** To see the files tracked by the local repository

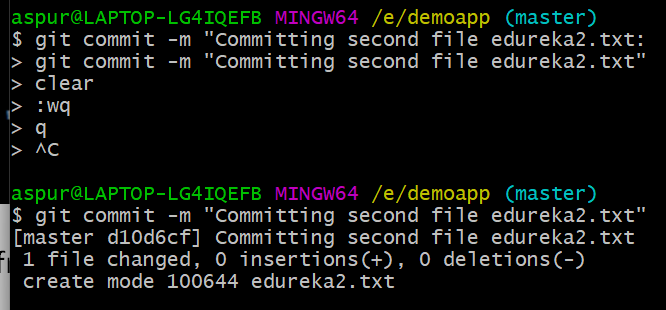


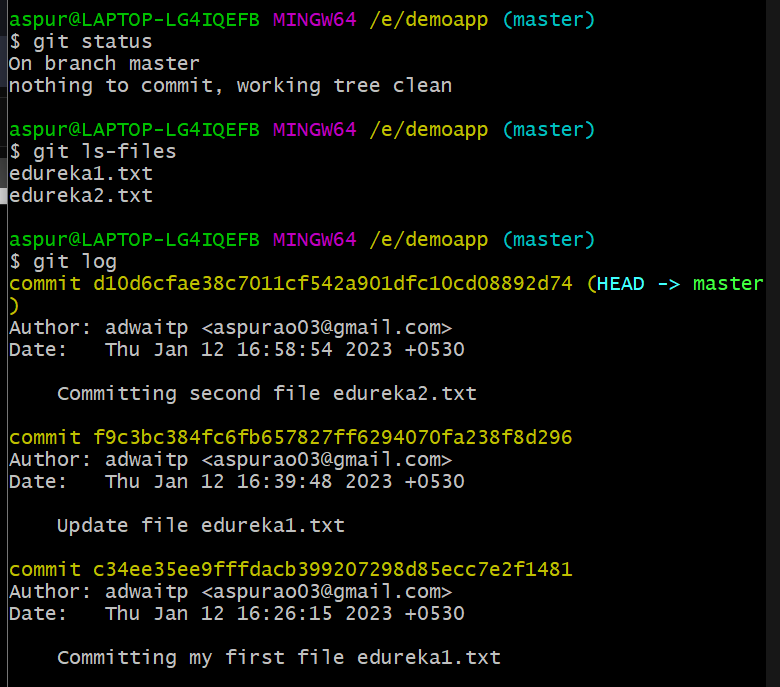


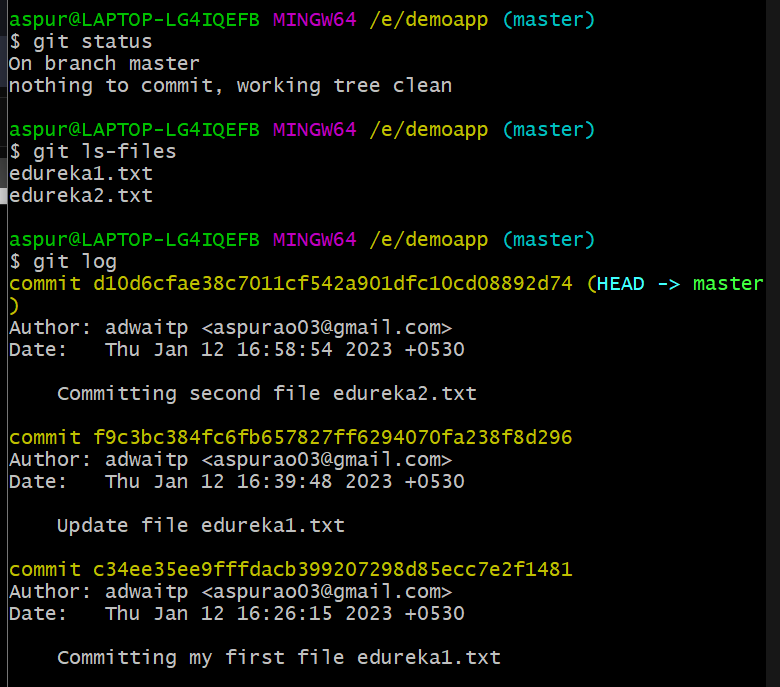




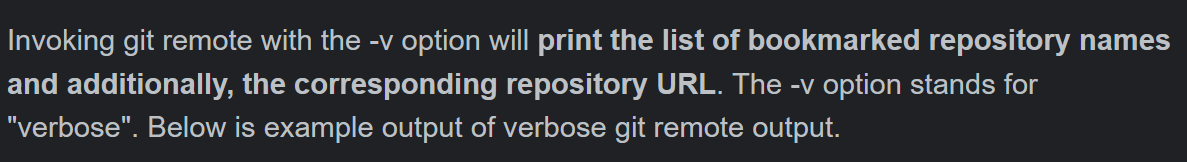
Press Ctrl + C if ‘>’ sign occurs in git bash and you want to exit from it





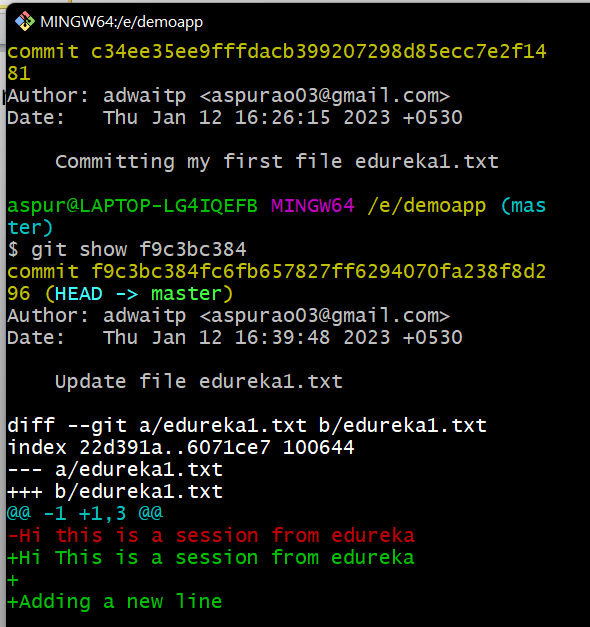


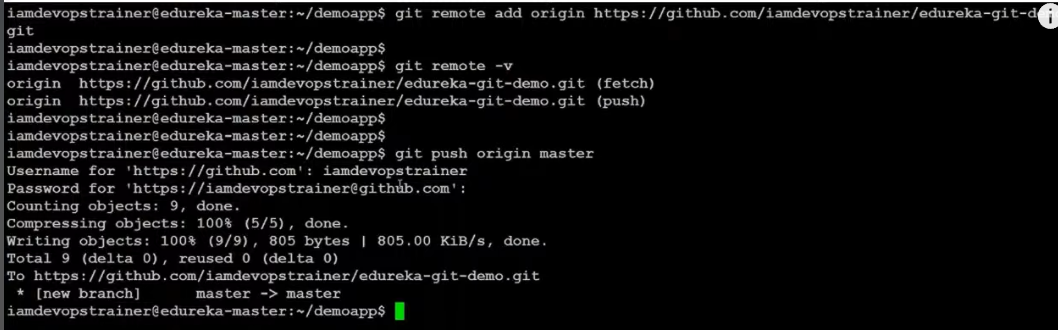
**Git remote add origin command:** To add your remote repository and link it with your local repository.

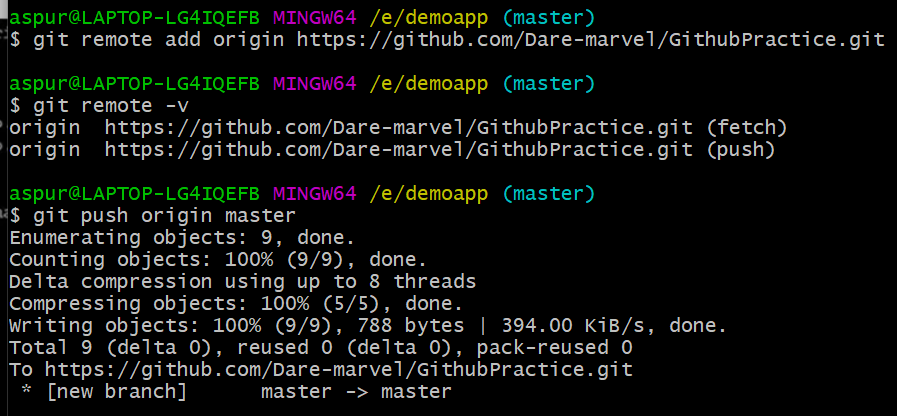


**Git Push:** Push all changes from local repo to remoter repository.

**Syntax**: git push origin master(alias- name for our repo)

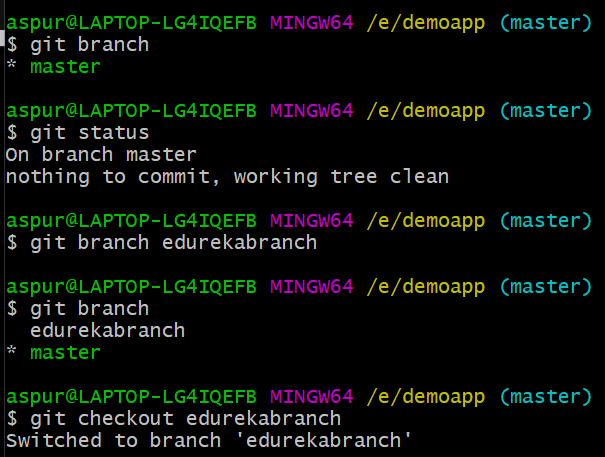




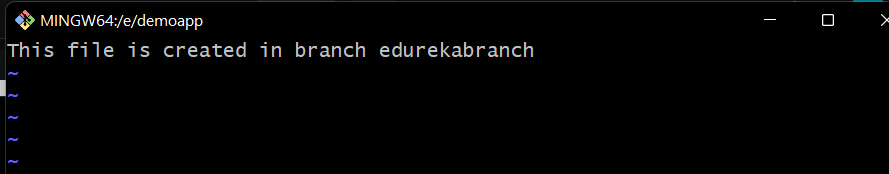


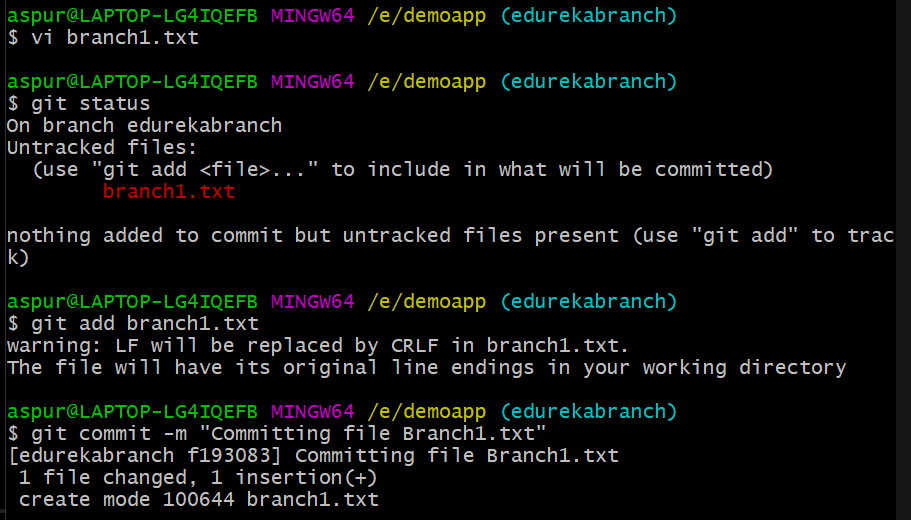
**Clear:** Used to clear Git Bash Terminal

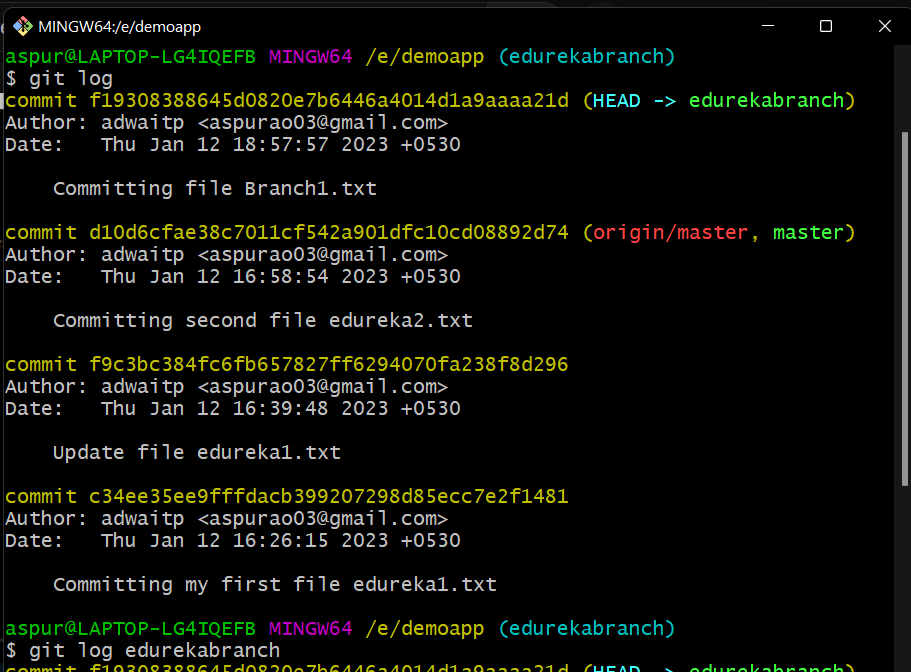
Git checkout <branch-name>: To switch branches.

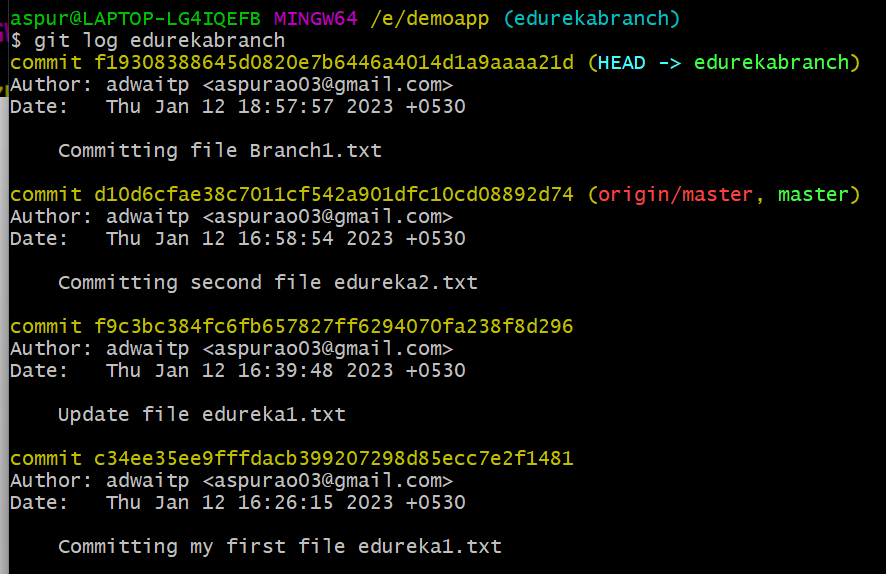


When we create a new branch, it will contain all the changes in the master branch









If newly created branch with gitbash isn’t visible on GitHub:

If you have created a new branch using Git Bash locally, but it is not visible on GitHub, you may need to push the branch to GitHub.

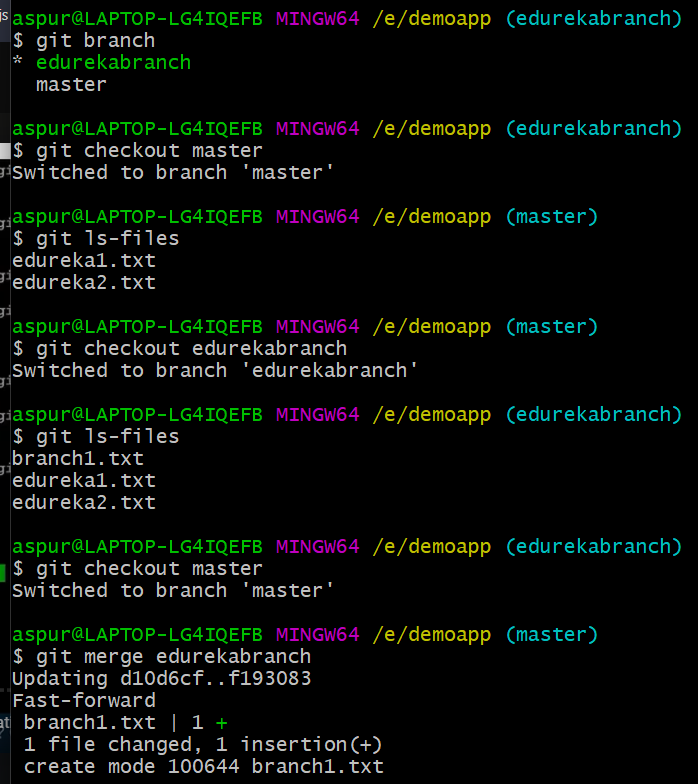
Here are the steps you can follow to push the new branch to GitHub:

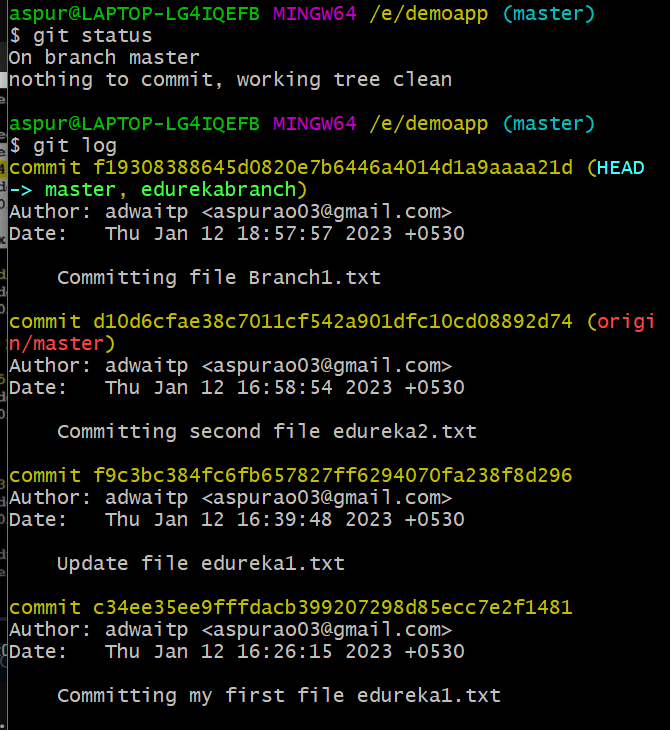
1. First, make sure you have committed your changes locally on the new branch using the command **git commit -m "commit message"**.
2. Next, use the command **git push -u origin <branch-name>** to push the new branch to the remote repository on GitHub. The **-u** option sets the upstream branch to track the newly created branch on GitHub, so you can use **git push** to push changes to the branch in the future.

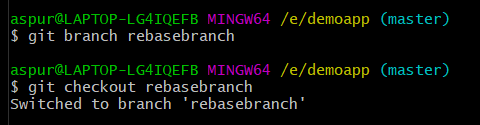
Replace **<branch-name>** with the name of your new branch.

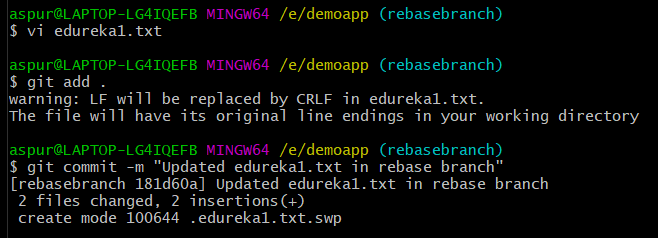
1. If you encounter any errors, such as authentication issues or permission errors, make sure you have the correct permissions to push changes to the repository on GitHub. You may need to authenticate with your GitHub credentials using the command **git config --global credential.helper cache**.
2. Finally, check the repository on GitHub to see if the new branch has been created and is visible.

**Merging Two branches**



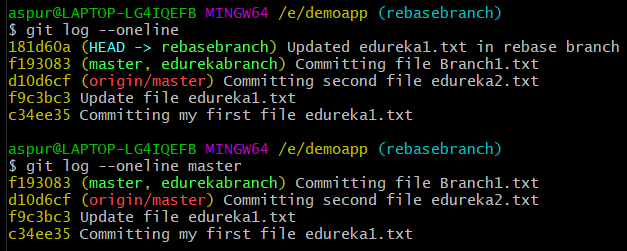




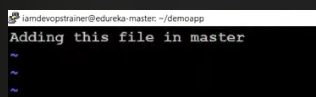


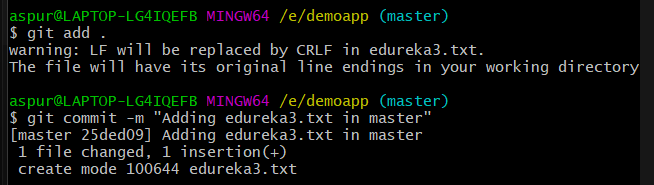
Get list of all commits in one line use command:

**Syntax**: git log –oneline









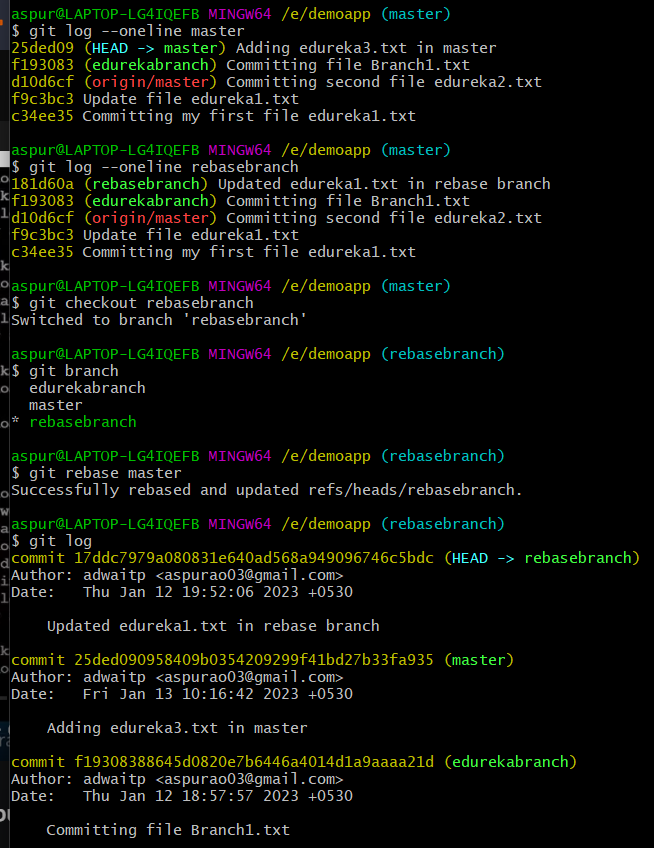
The **git rebase** command is used to apply the changes of one branch on top of another branch, essentially rewriting the commit history of the target branch.

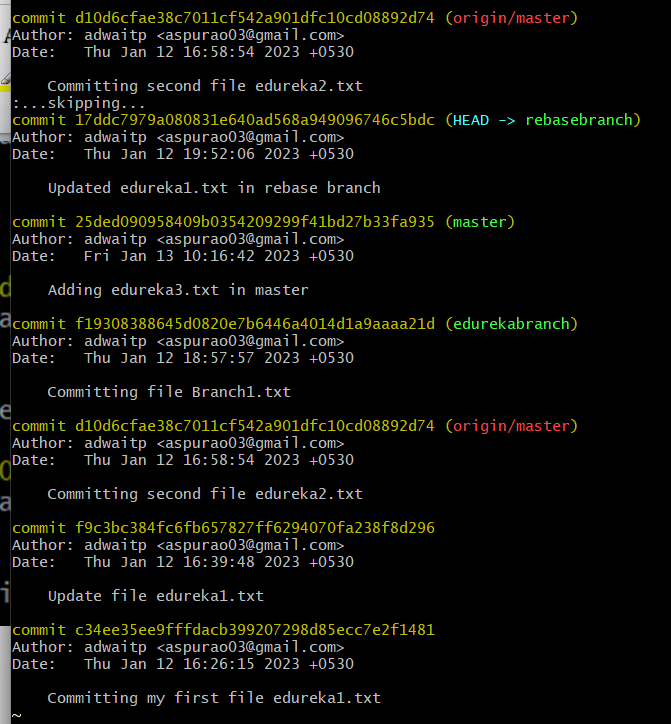
More specifically, the **git rebase** command takes a series of commits from one branch (let's call it **feature**) and applies them on top of another branch (let's call it **master**). This results in a new branch (in this case, **feature**) with the same changes as before, but now based on the latest changes in **master**.

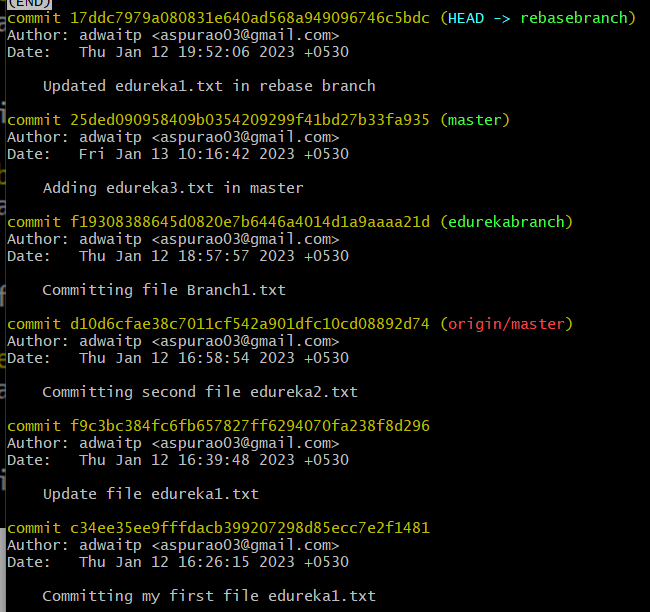
Here are the steps involved in performing a git rebase:

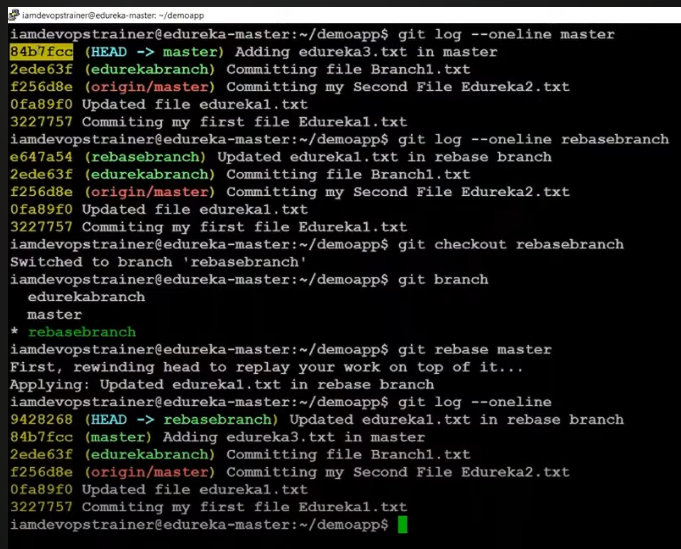
1. Switch to the branch you want to rebase (in our example, **feature**) using the command **git checkout feature**.
2. Use the command **git rebase master** to start the rebase process. This will take the changes from the **feature** branch and apply them on top of the latest changes in the **master** branch.
3. If there are any conflicts between the changes in the two branches, Git will prompt you to resolve them manually. You can use a tool like a text editor or Git's built-in conflict resolution tool to resolve the conflicts.
4. Once the conflicts are resolved, use the command **git rebase --continue** to continue with the rebase process.
5. If everything goes smoothly, the **feature** branch will now be based on the latest changes in **master**, with all of its changes applied on top.

It's important to note that rebasing changes the commit history of the target branch, so it's generally not recommended to rebase a branch that has already been pushed to a shared repository. This can cause confusion and conflicts for other team members who may be working on the same branch. Instead, it's best to use **git merge** to incorporate changes from other branches into a shared branch.

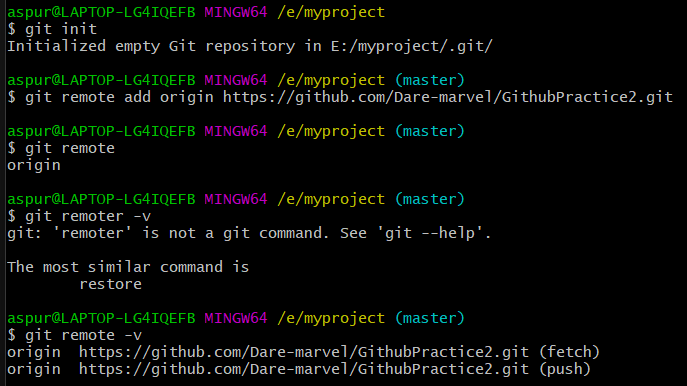


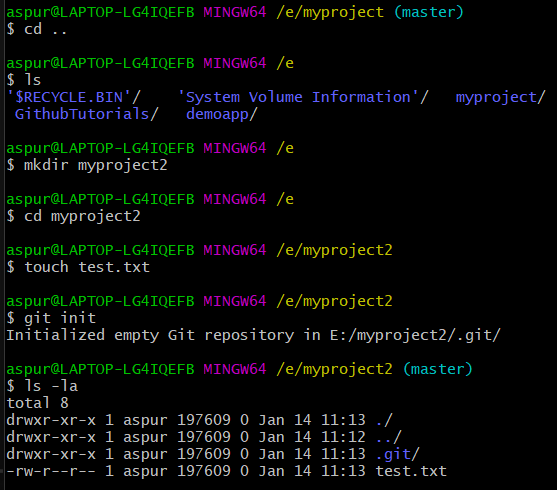


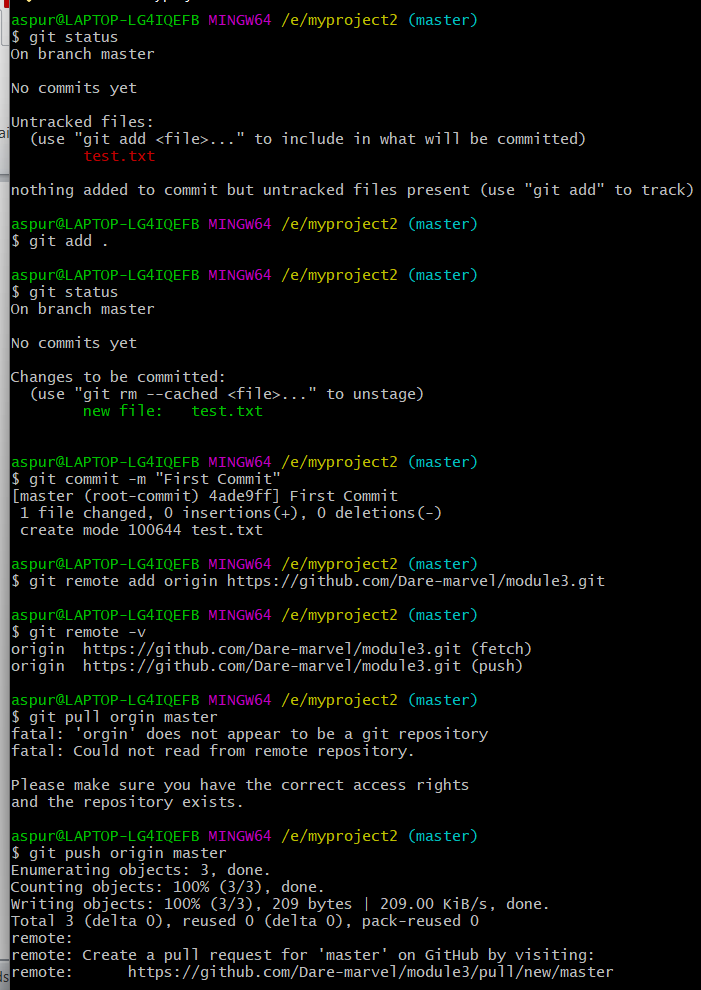




Initializing my Second Repository













**Git Pull:**

Graphical user interface

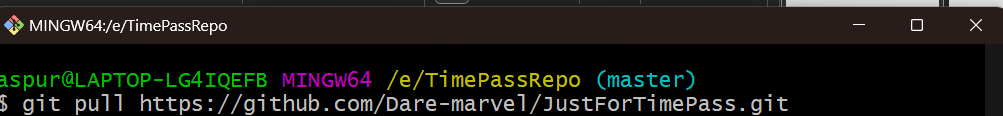
Description automatically generated

Text

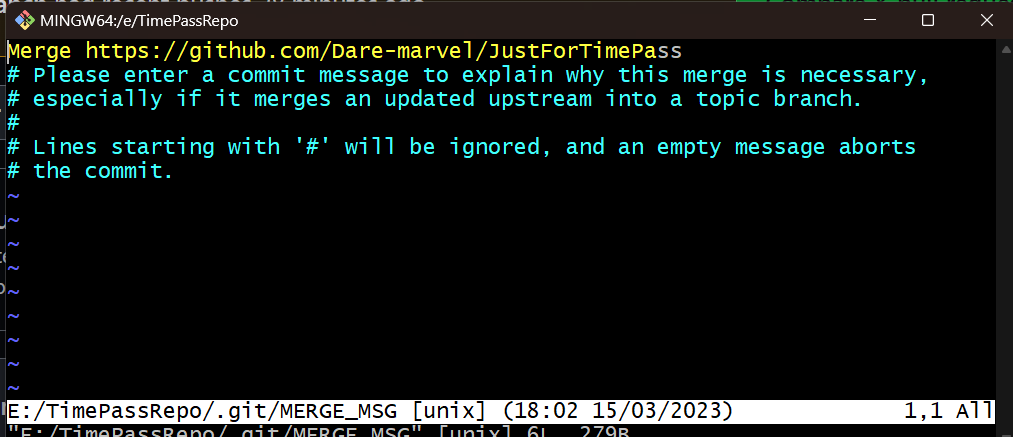
Description automatically generated

A picture containing timeline

Description automatically generated



Type :qa and press <Enter> to exit Vim



Text

Description automatically generated