**GITHUB ASSIGNMENT**

**NAME:** DHEETCHANAA K

**BRANCH:** CSE-KSRCT

**REG.NO:** 73772114130

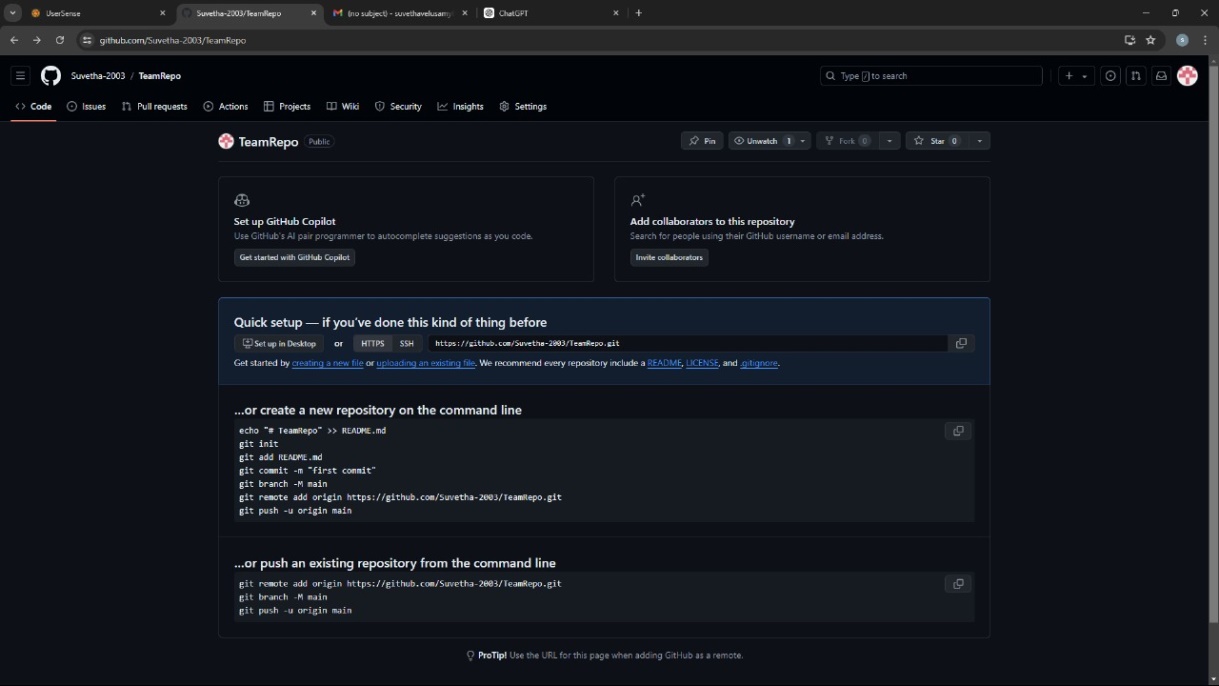
**EMAIL:** <kanagarajdheetchanaa@gmail.com>

Exercise 3

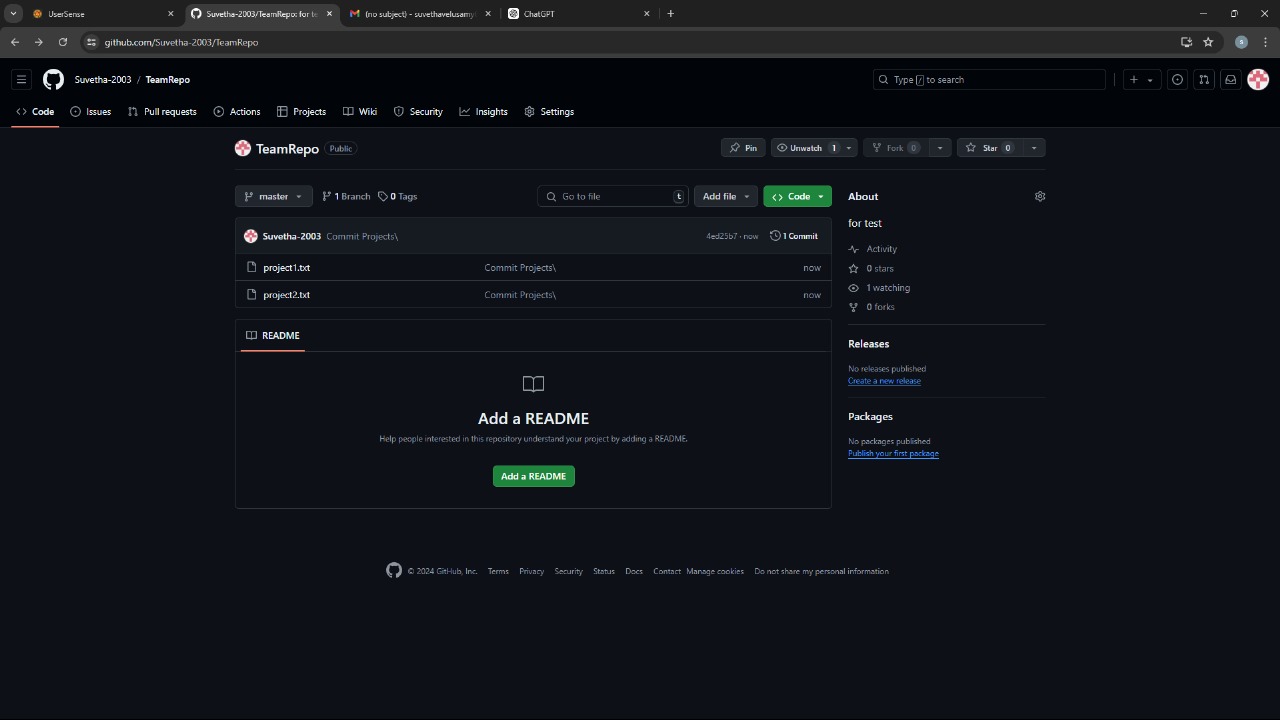
For this task, you will work in a small group. Between 2 and 4 people is about right.

Main Task

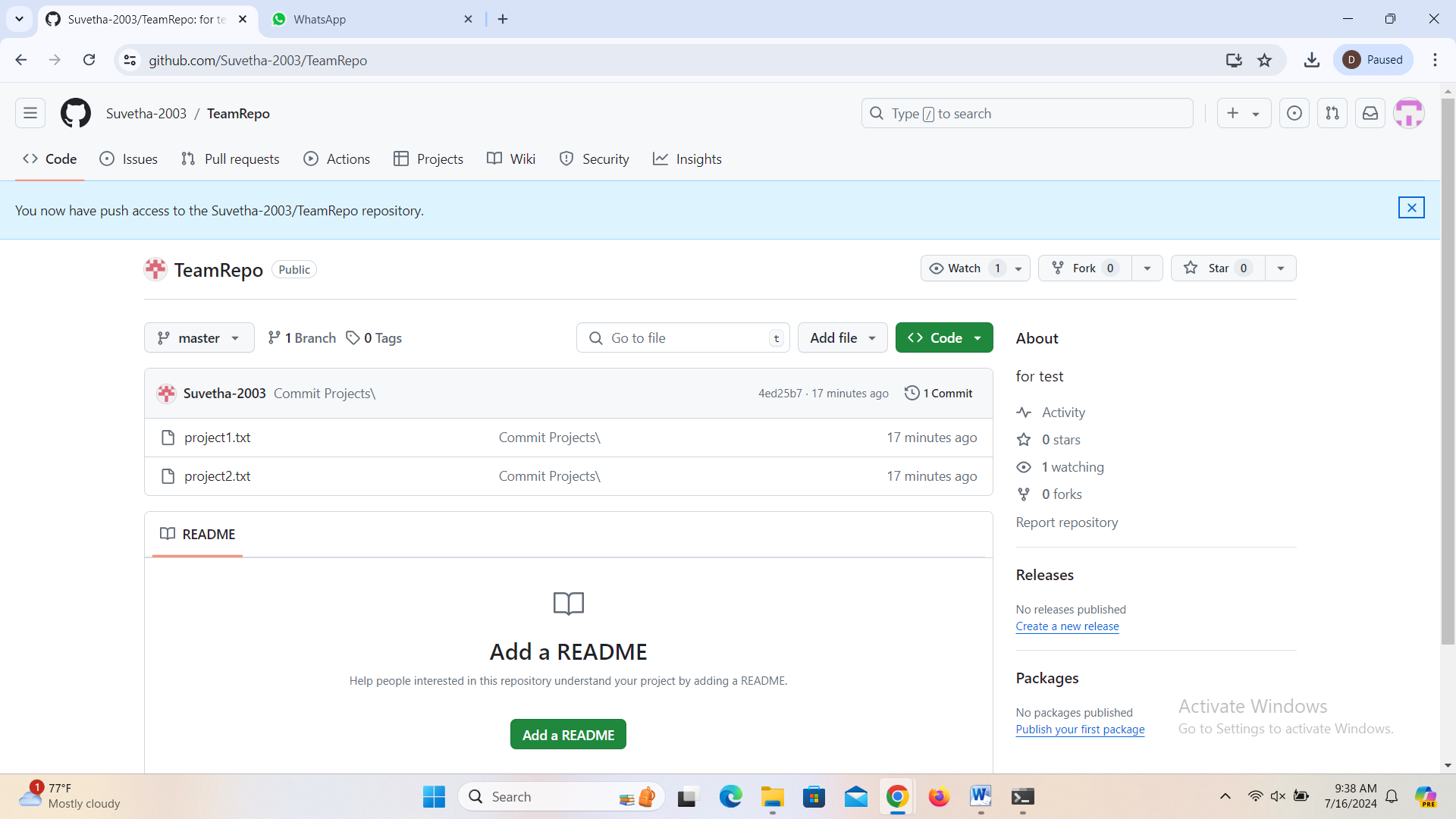
1. First, one person in the group should create a public repository using their GitHub account.



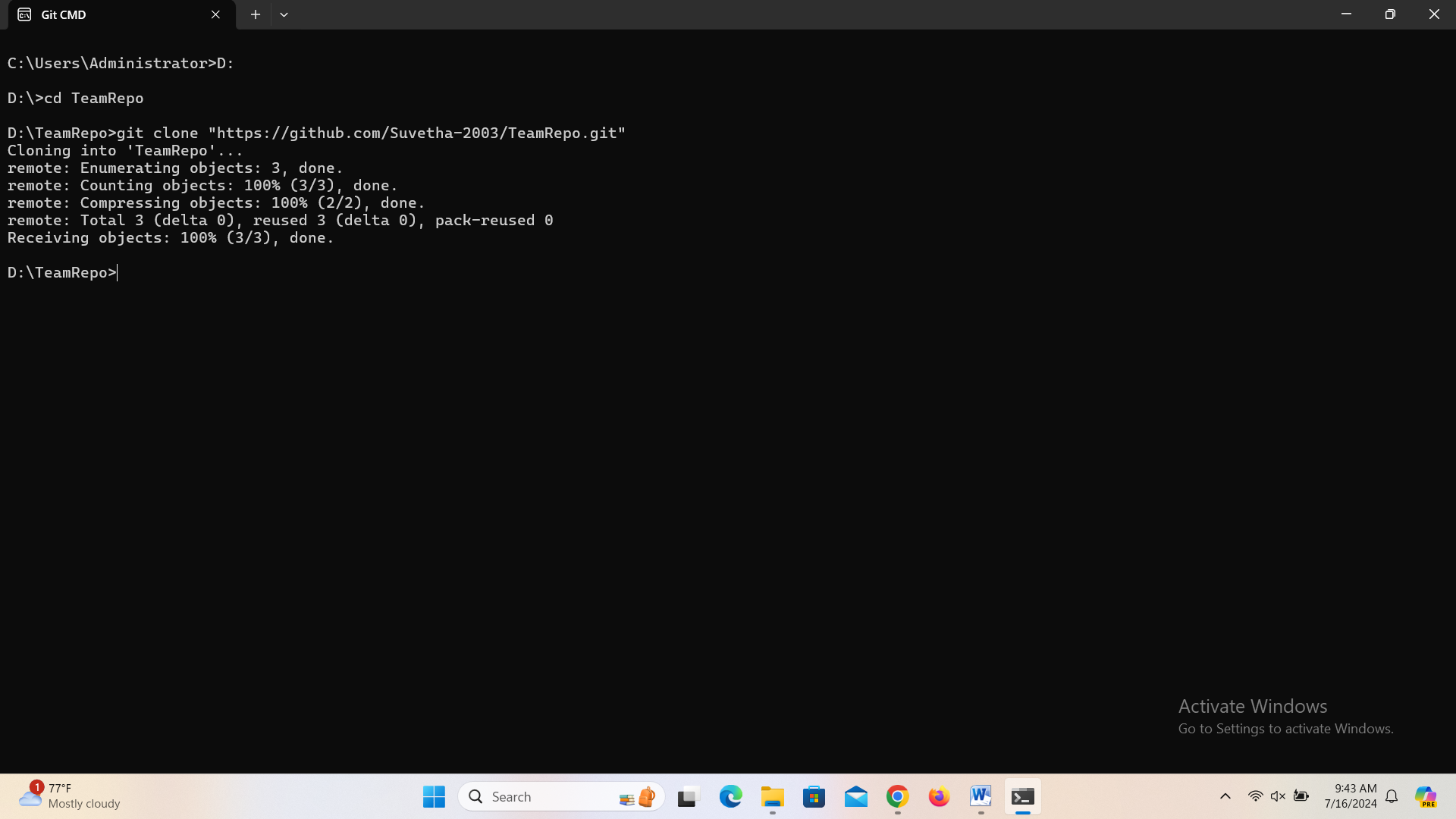
2. This same person should then follow the instructions from GitHub to add a remote, and then push their repository. Do not forget the –u flag, as suggested by GitHub!



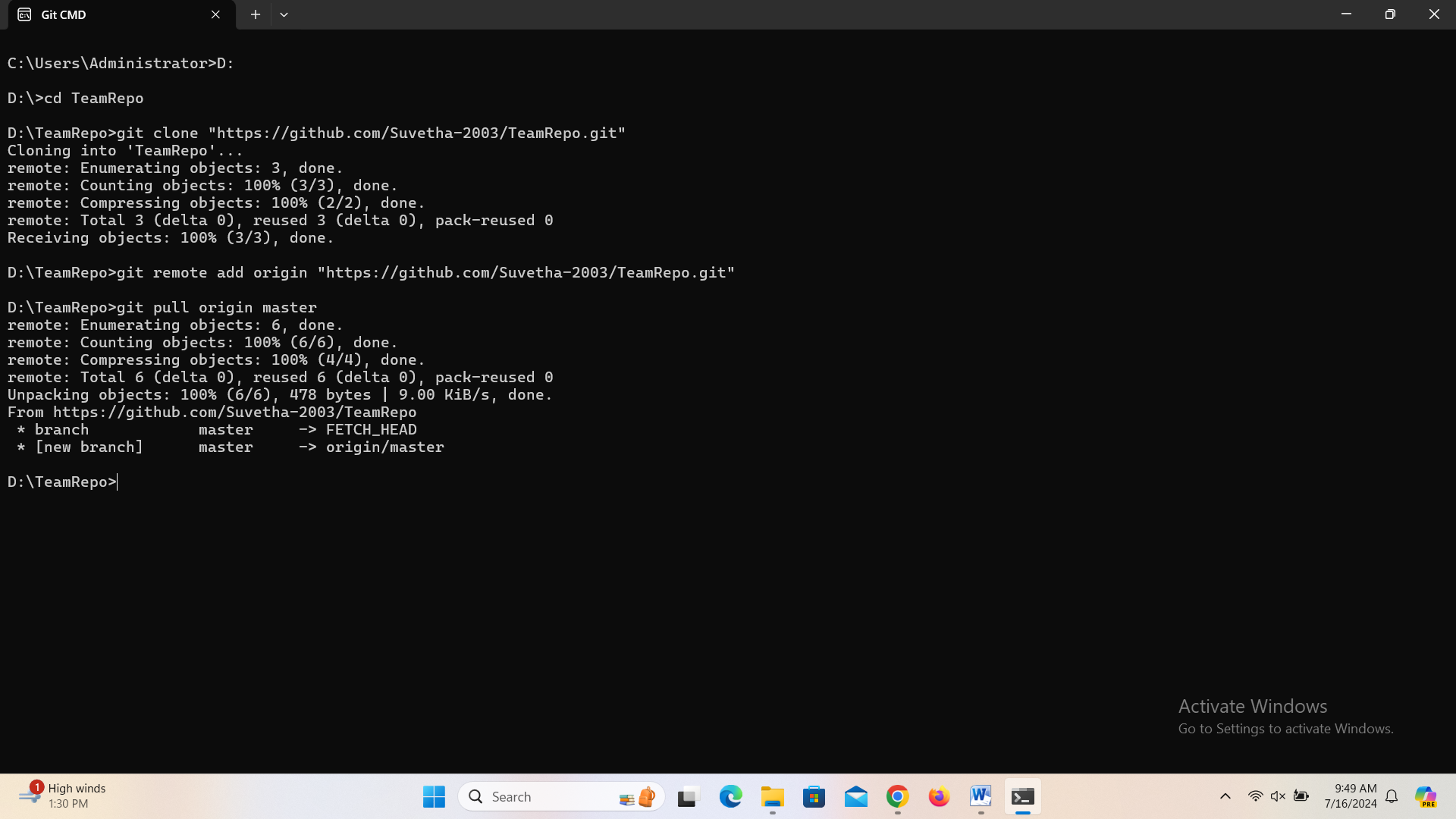
3. All of the other members of the group should then be added as collaborators, so they can commit to the repository also.



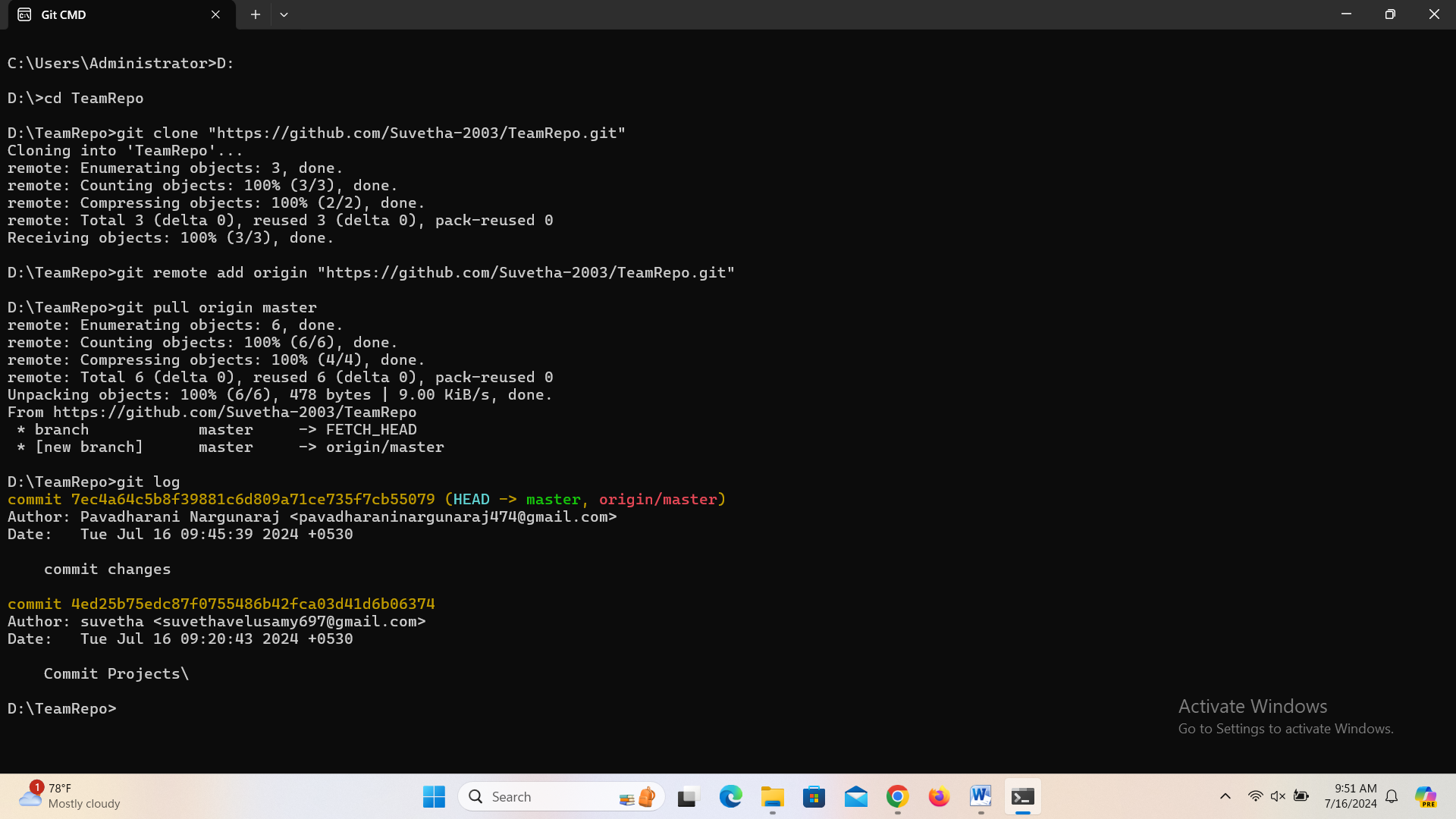
4. Next, everyone else in the group should clone the repository from GitHub. Verify that the context of the repository is what is expected.



5. One of the group members who just cloned should now make a local commit, then push it. Everyone should verify that when they pull, that commit is added to their local repository (use git log to check for it).



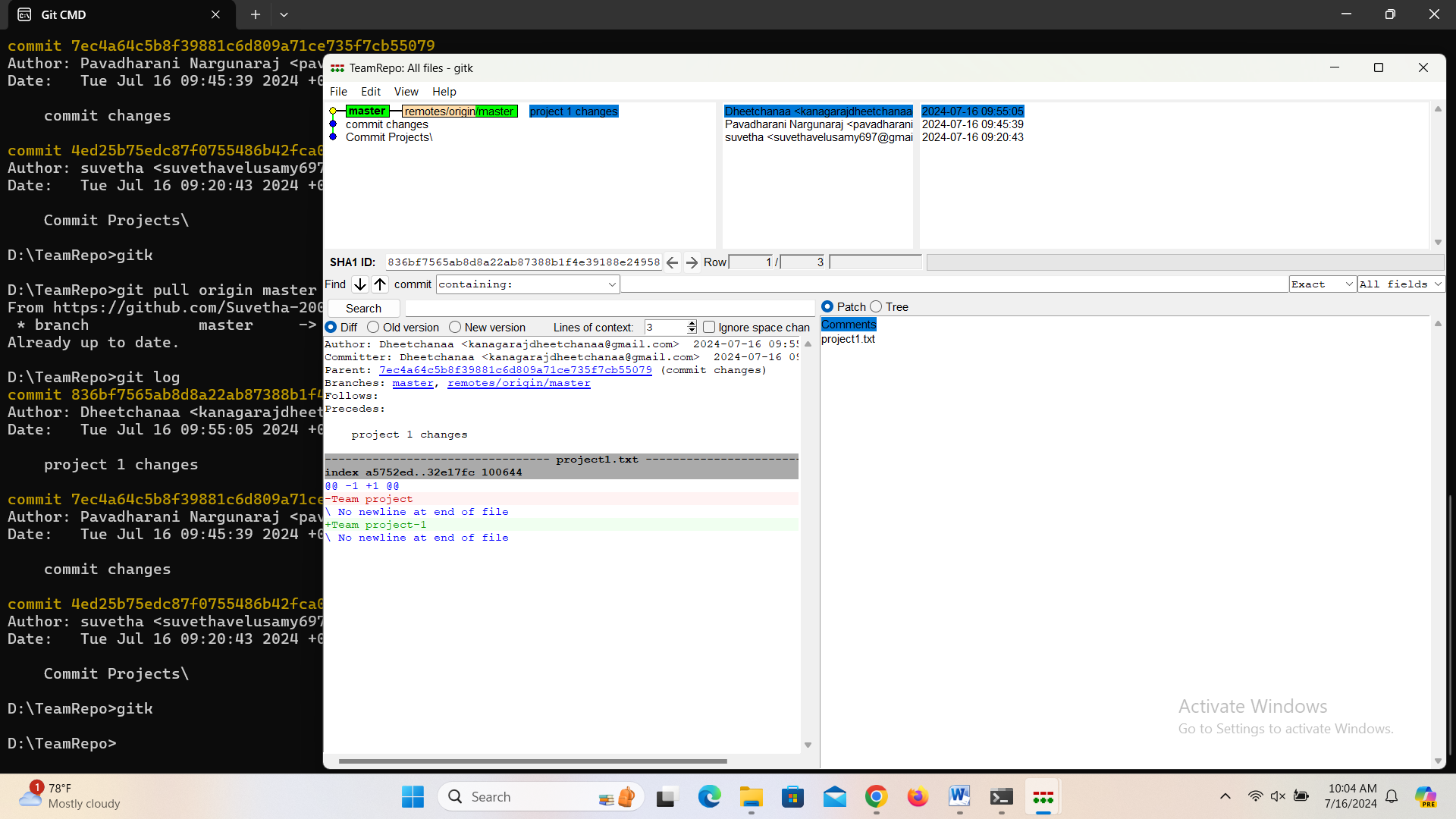
6. Look at each other’s git log output. Notice how the SHA-1 is the same for a given commit across every copy of the repository. Why is this important?



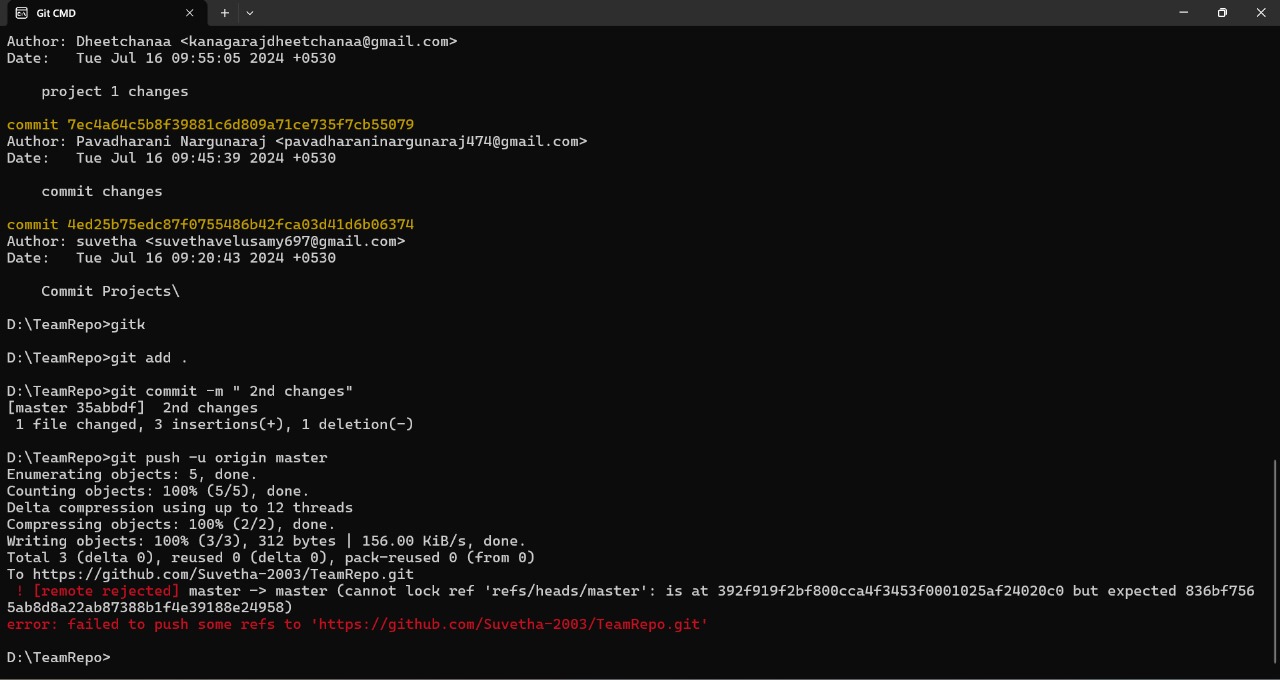
7. Two members of the group should now make a commit locally, and race to push it. To keep things simple, be sure to edit different files. What happens to the runner-up?

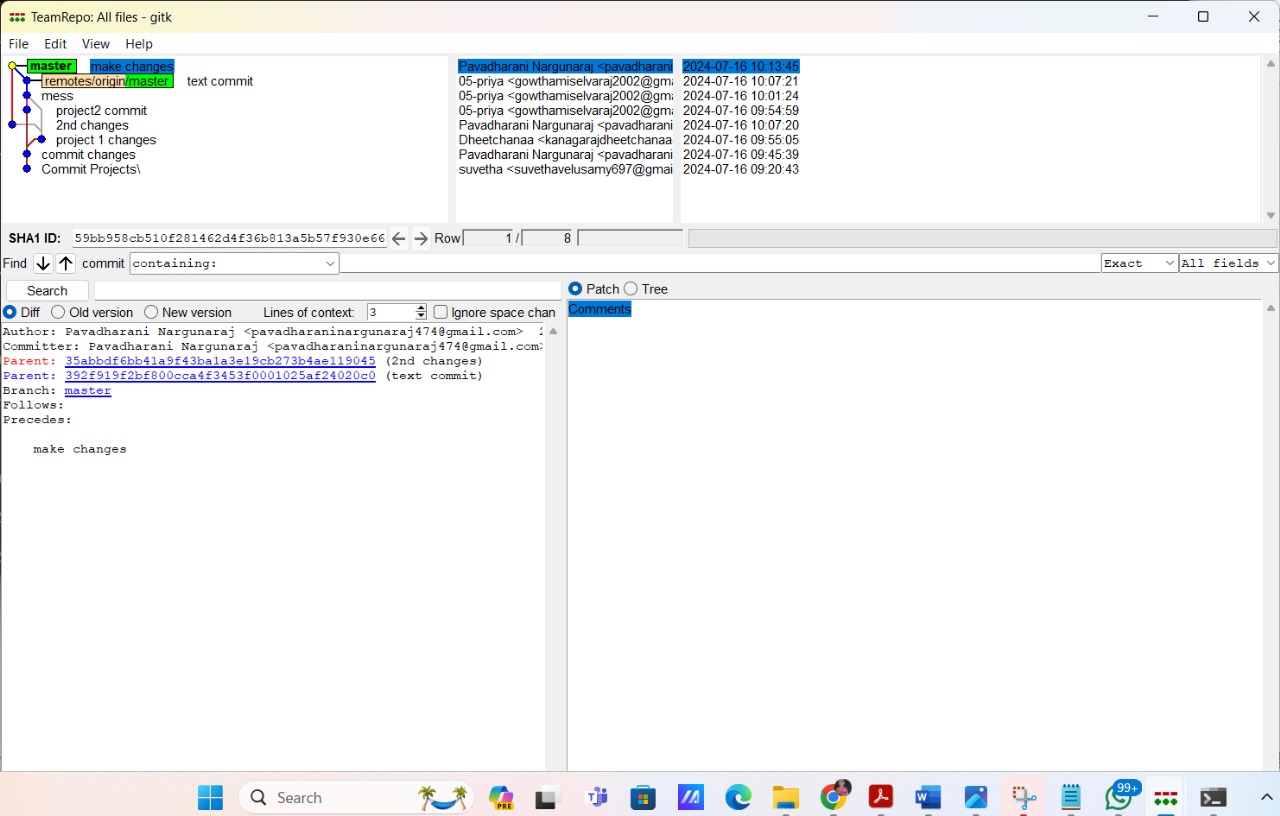


8. The runner-up should now pull. As a group, look at the output of the command. Additionally, look at the git log, and notice that there is a merge commit. You may also wish to view the DAG in gitk.



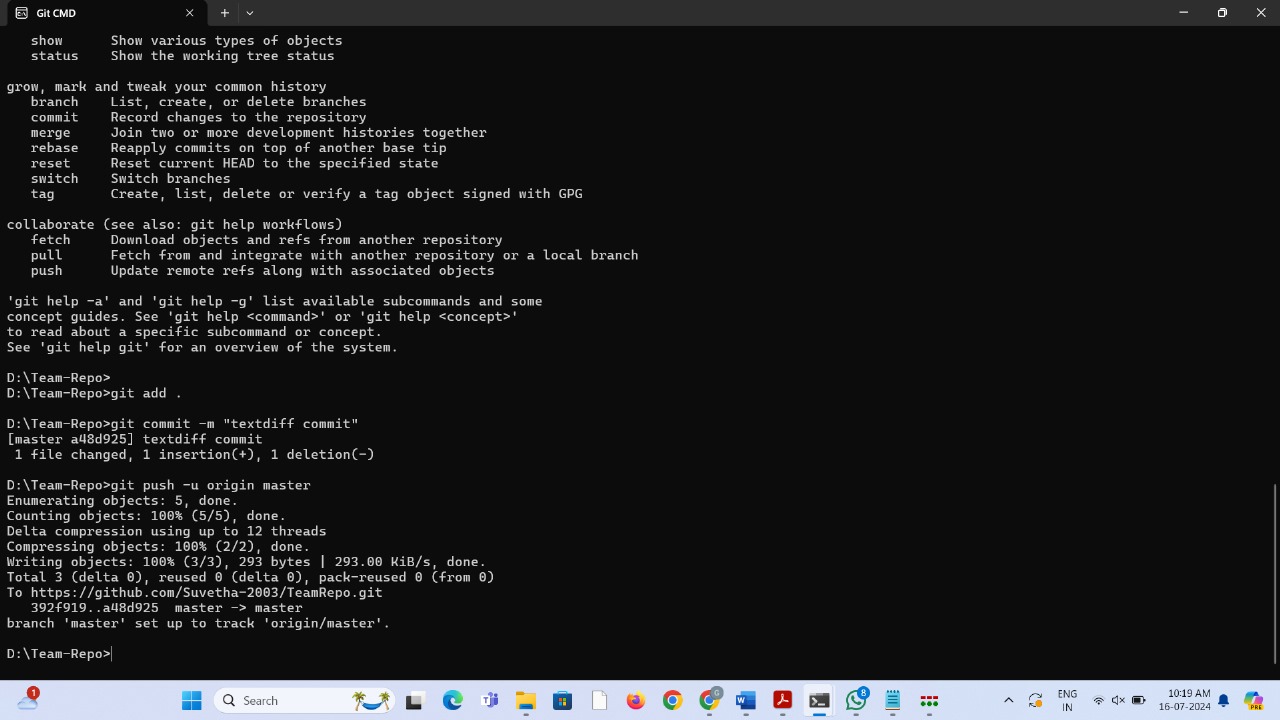
9. Repeat the last two steps a couple of times, to practice.

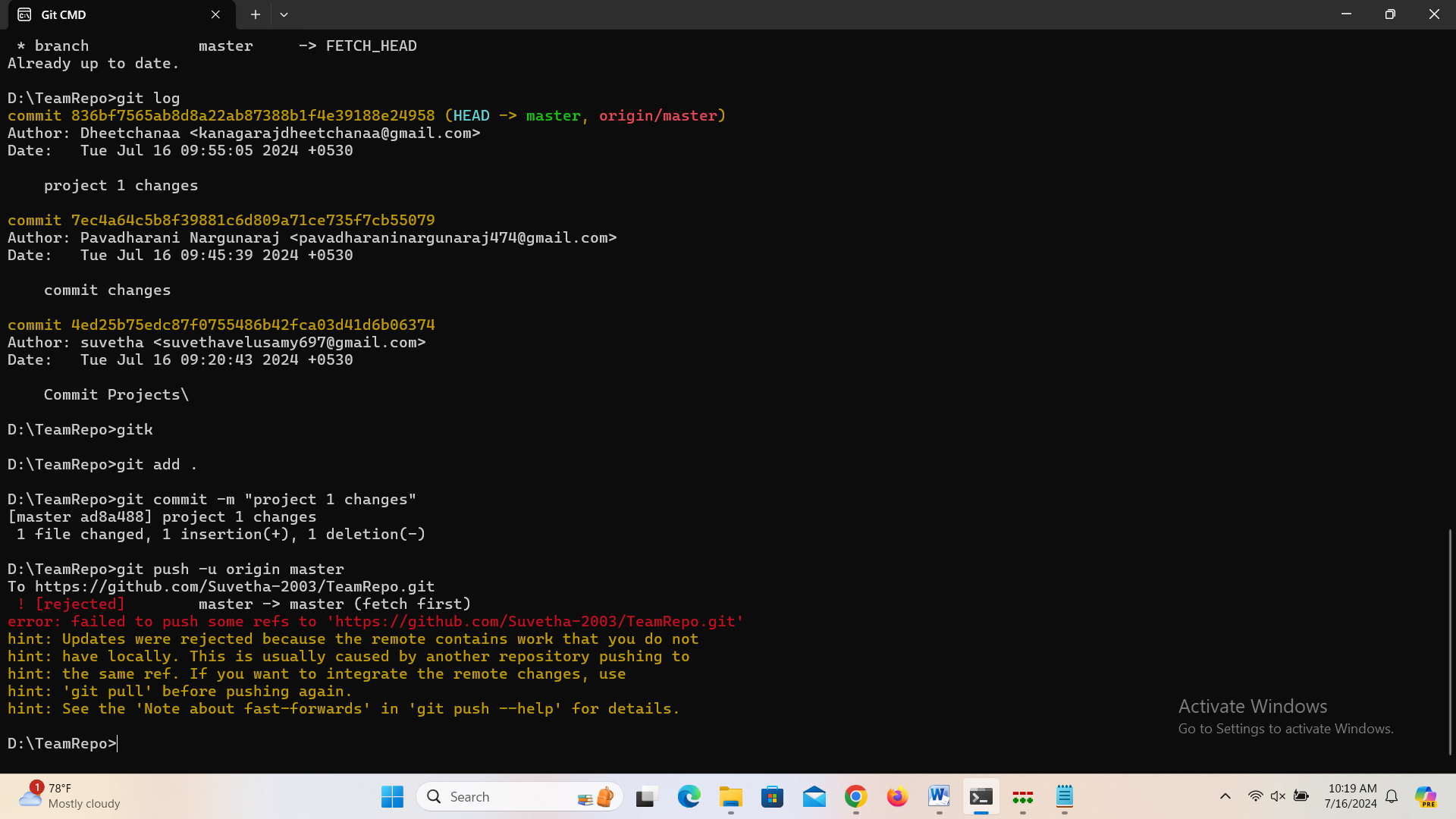




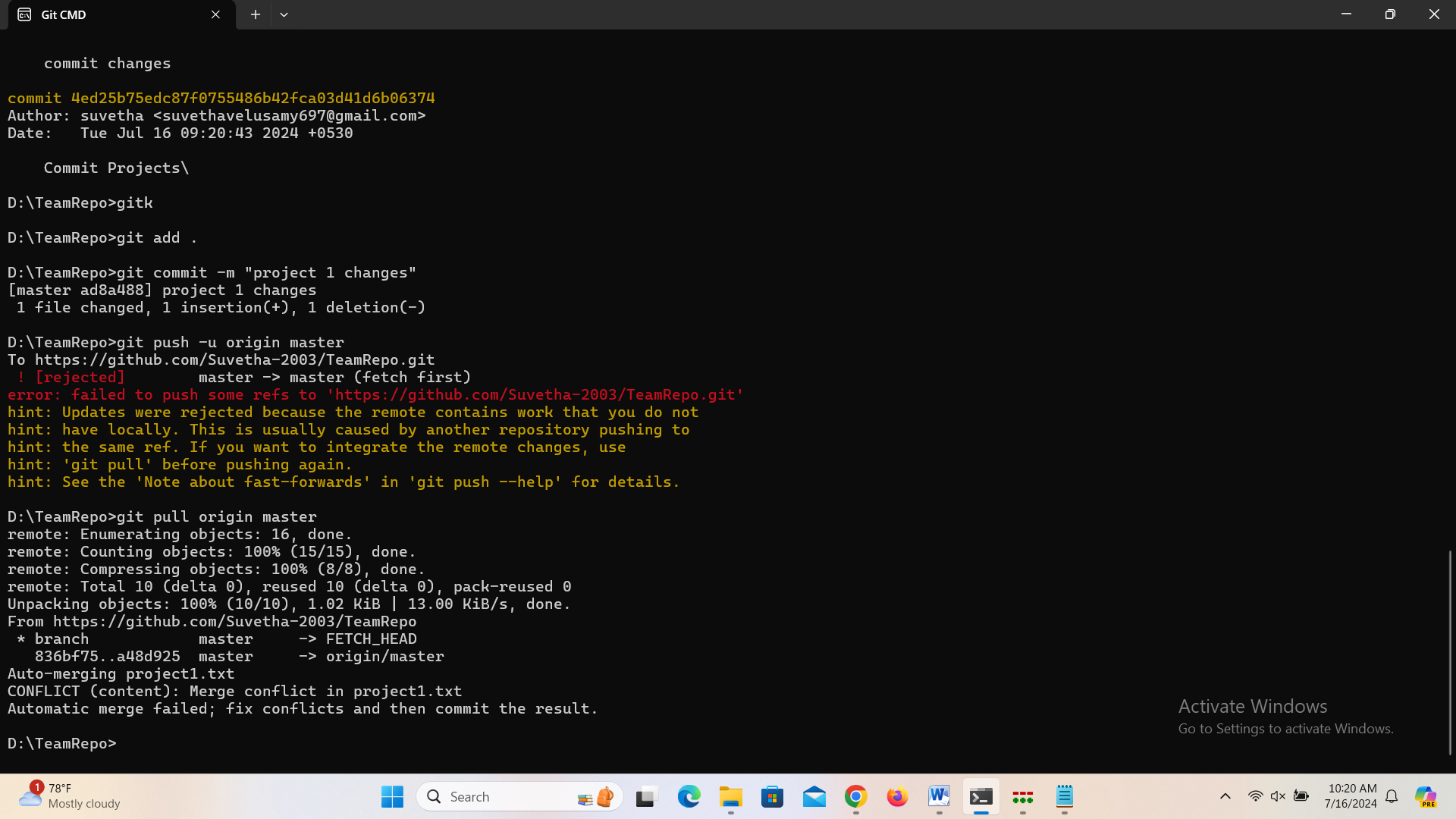
Stretch Task

1. Now create a situation where two group members both edit the same line in the same file and commit it locally. Race to push.





2. When the runner-up does a pull, they should get a merge conflict.



3. Look as a group at the file in conflict, and resolve it.

4. Use the add command to stage the fix, and then use commit to make the merge commit. Notice how this procedure is exactly the one you got used to when resolving conflicts in branches.

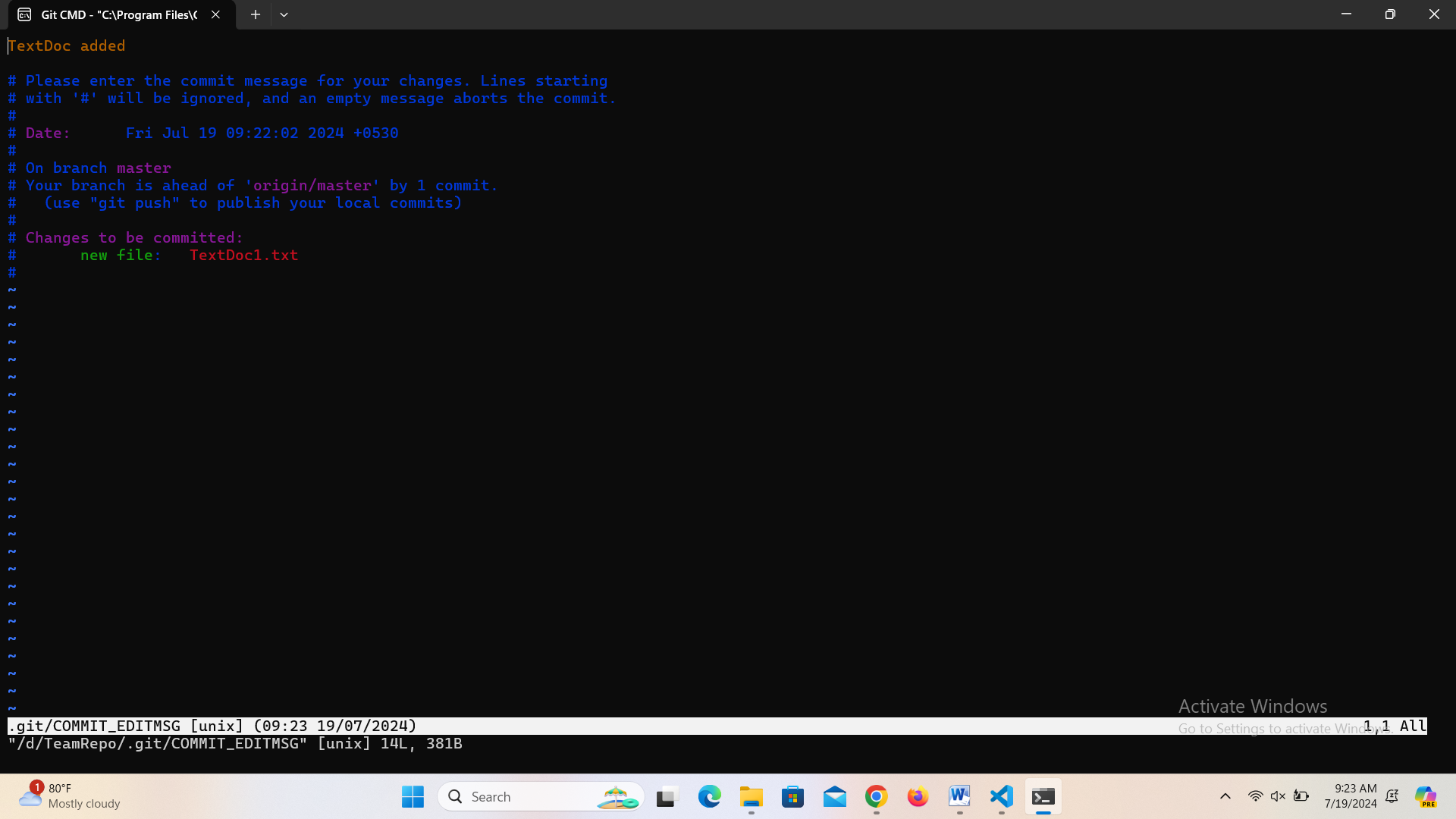
Exercise 4

Main Task

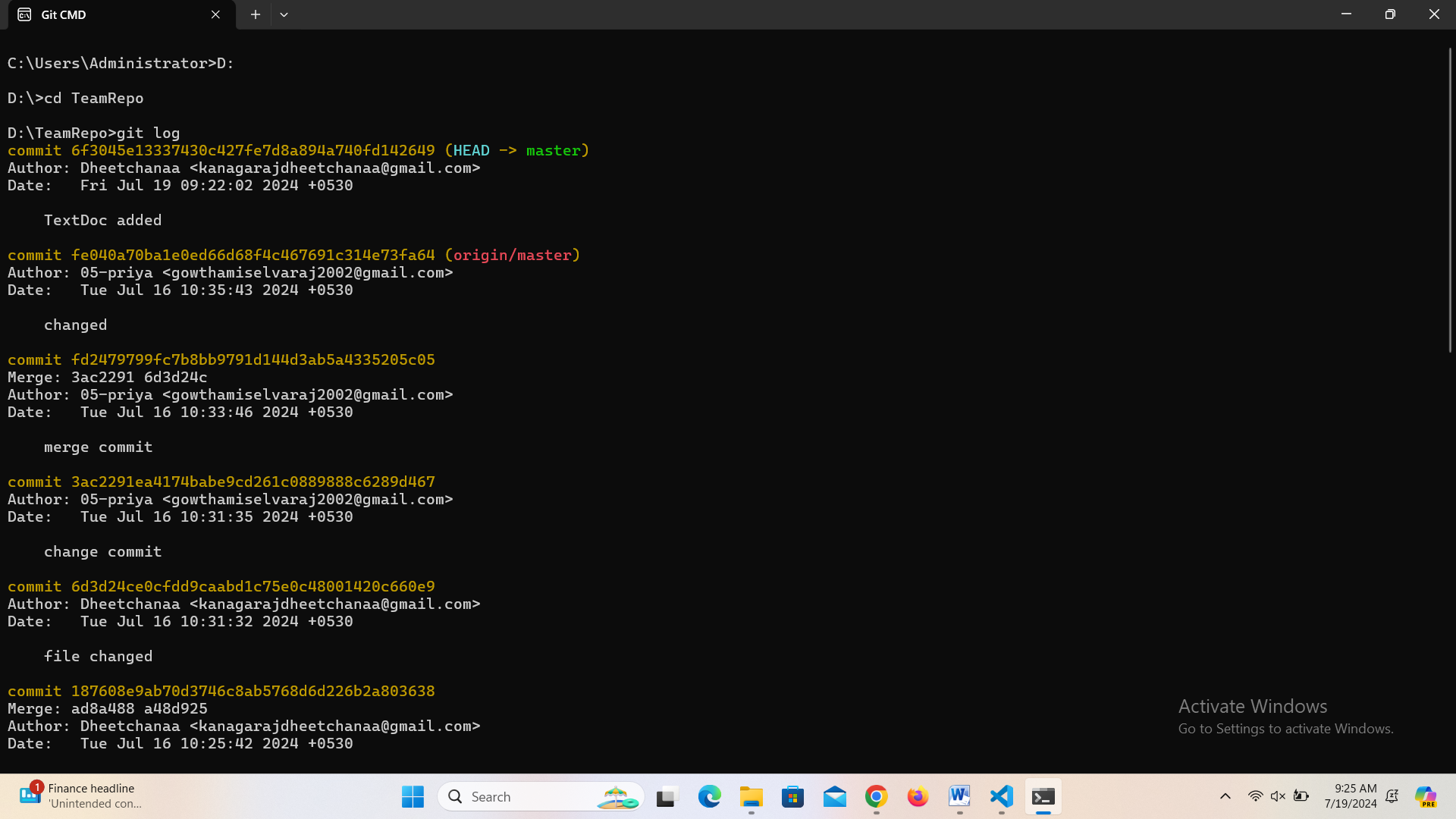
1. Make a commit, and make a silly typo in the commit message.



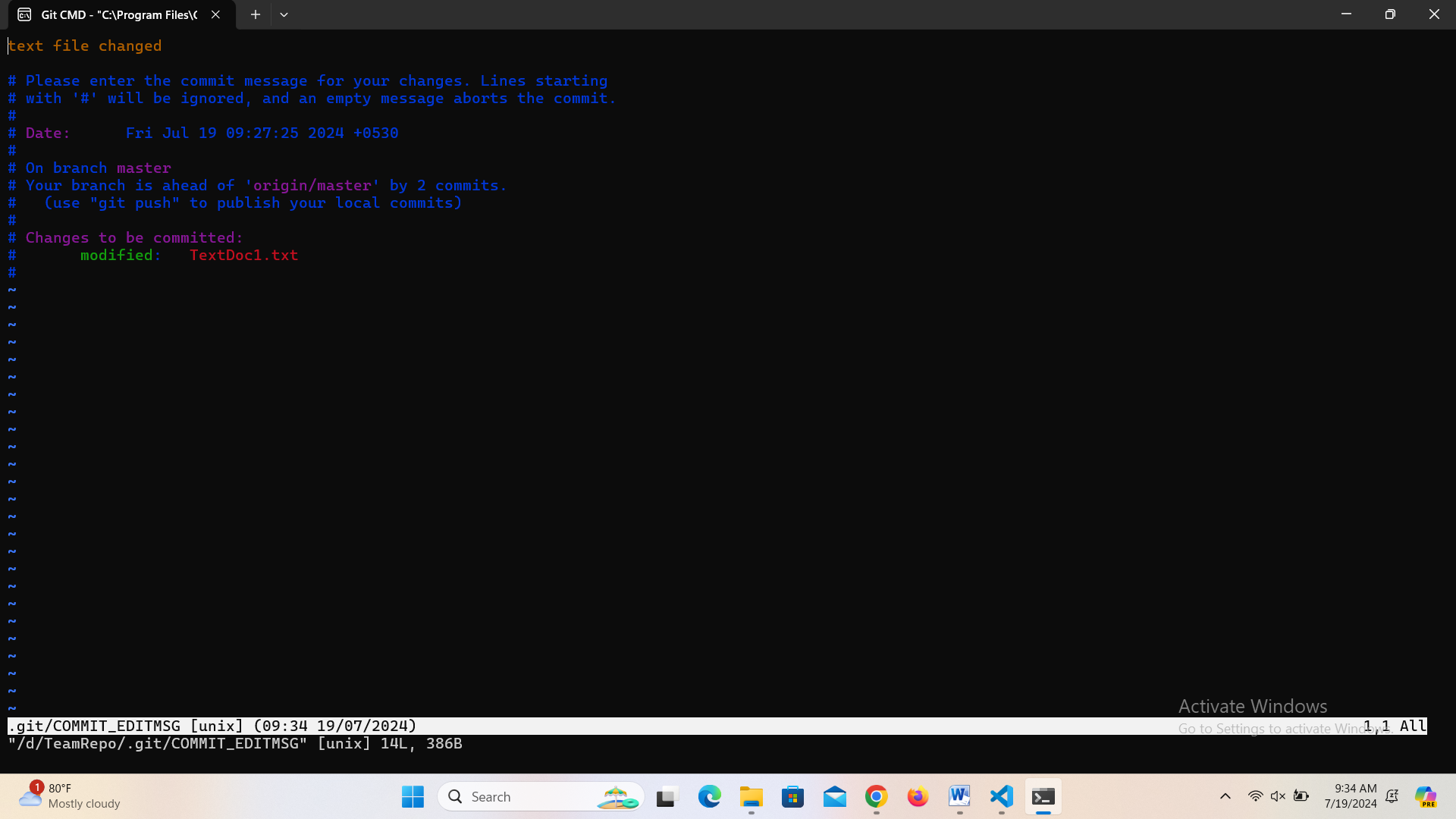
2. Use the --amend flag to enable you to fix the commit message.



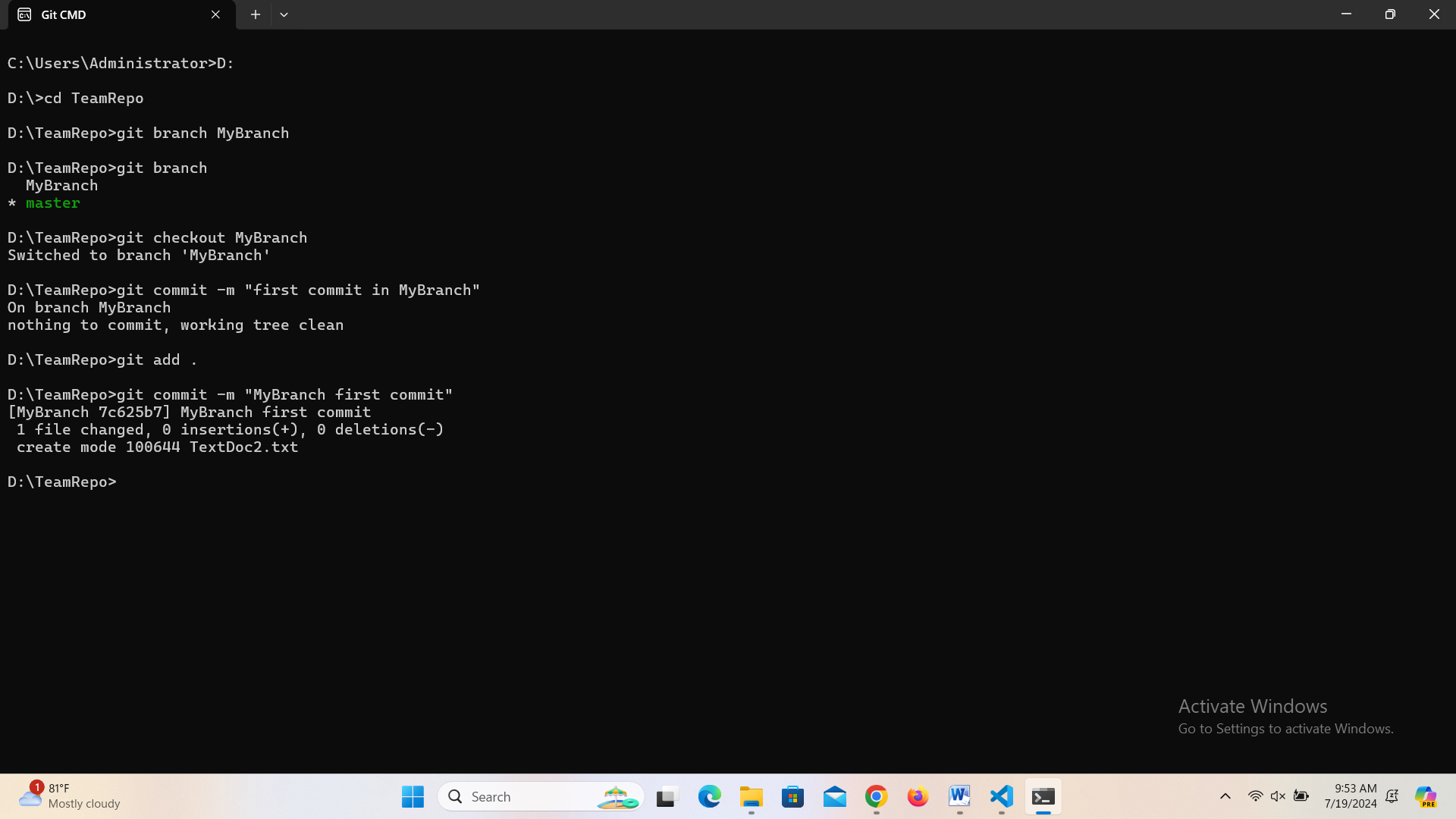
3. Look at the log and notice how the mistake is magically gone.



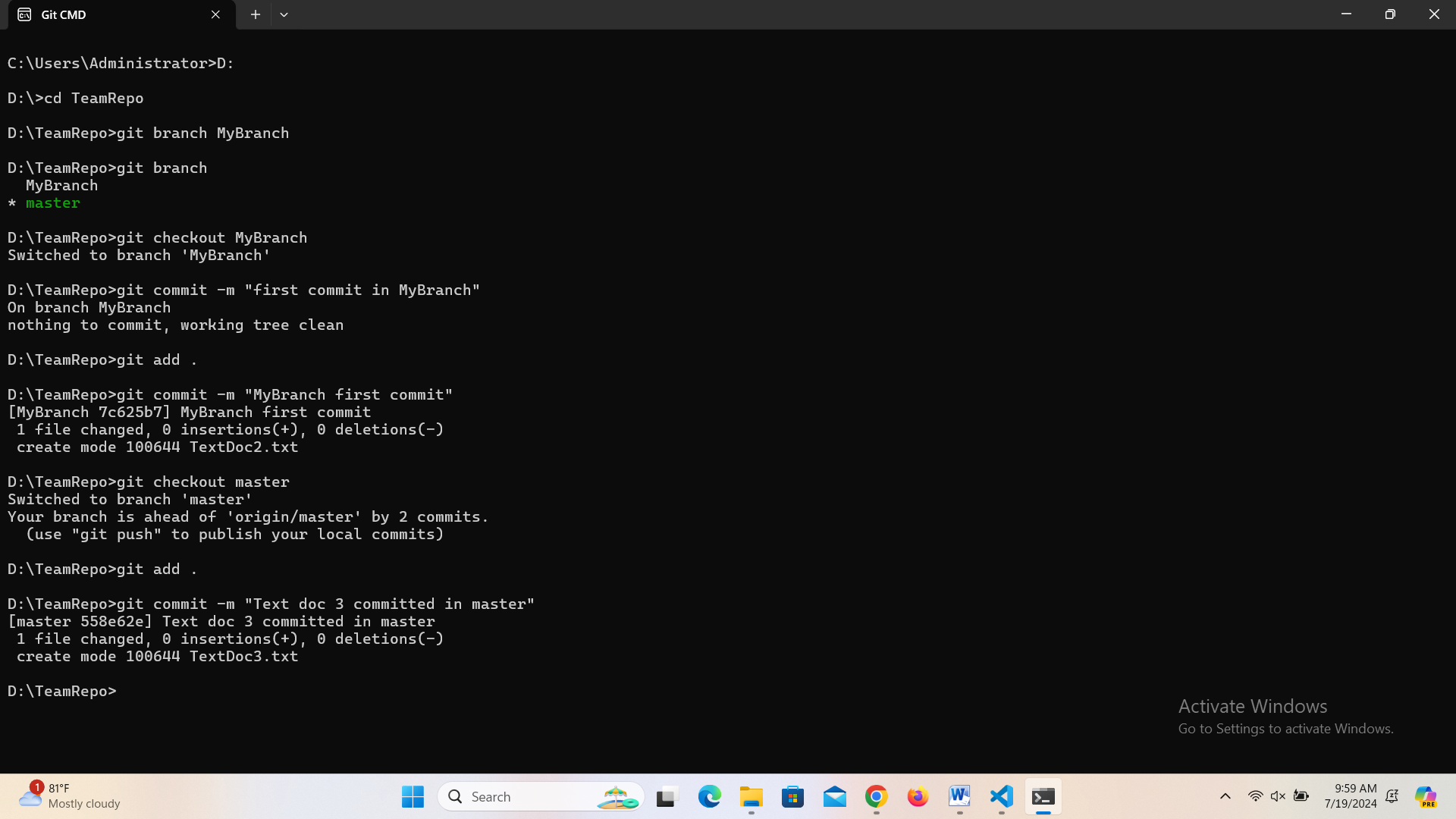
4. Now make a commit where you make a typo in one of the files. Once again, use --amend to magic away your problems.



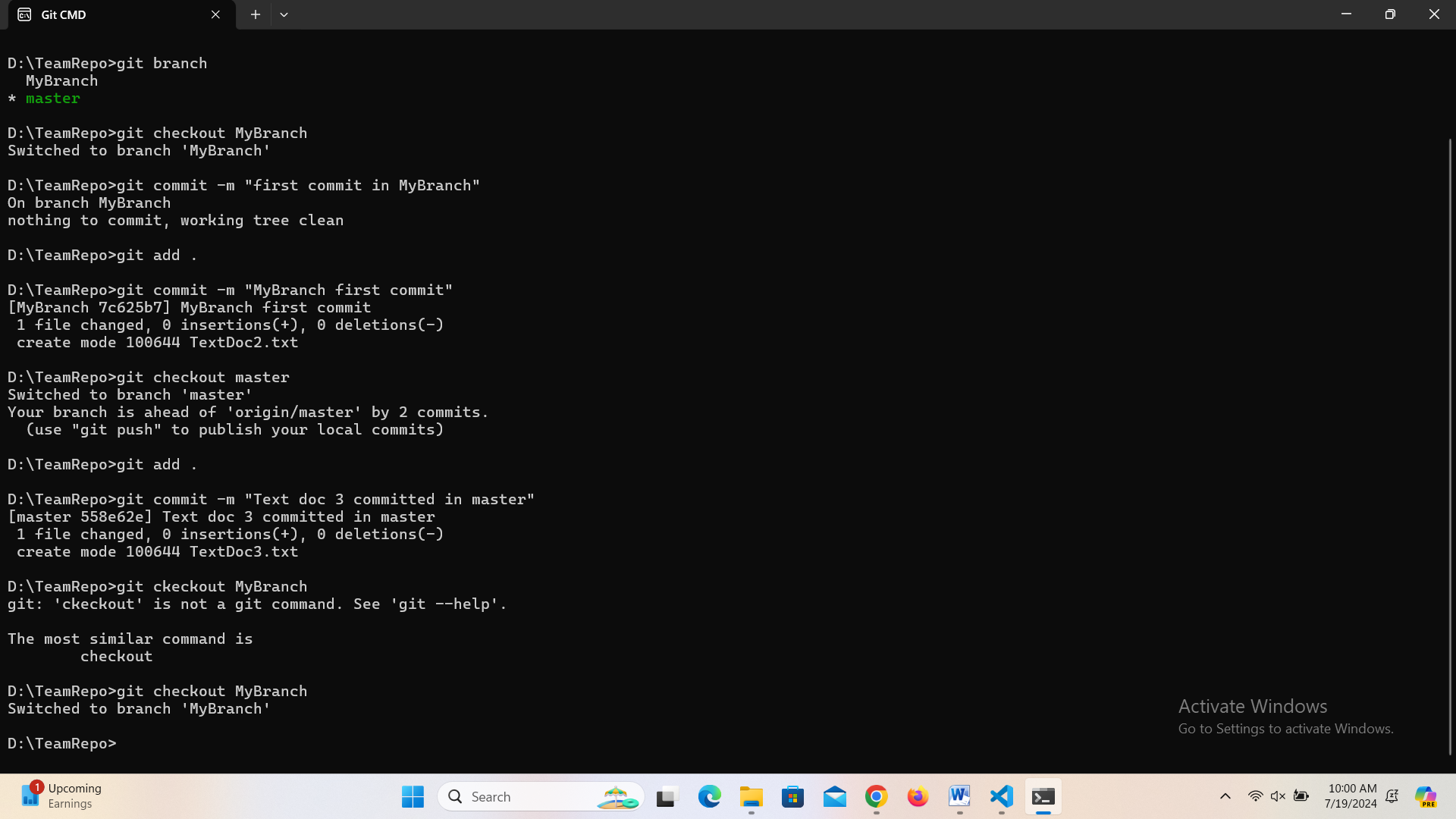
5. Create a branch. Make a commit.



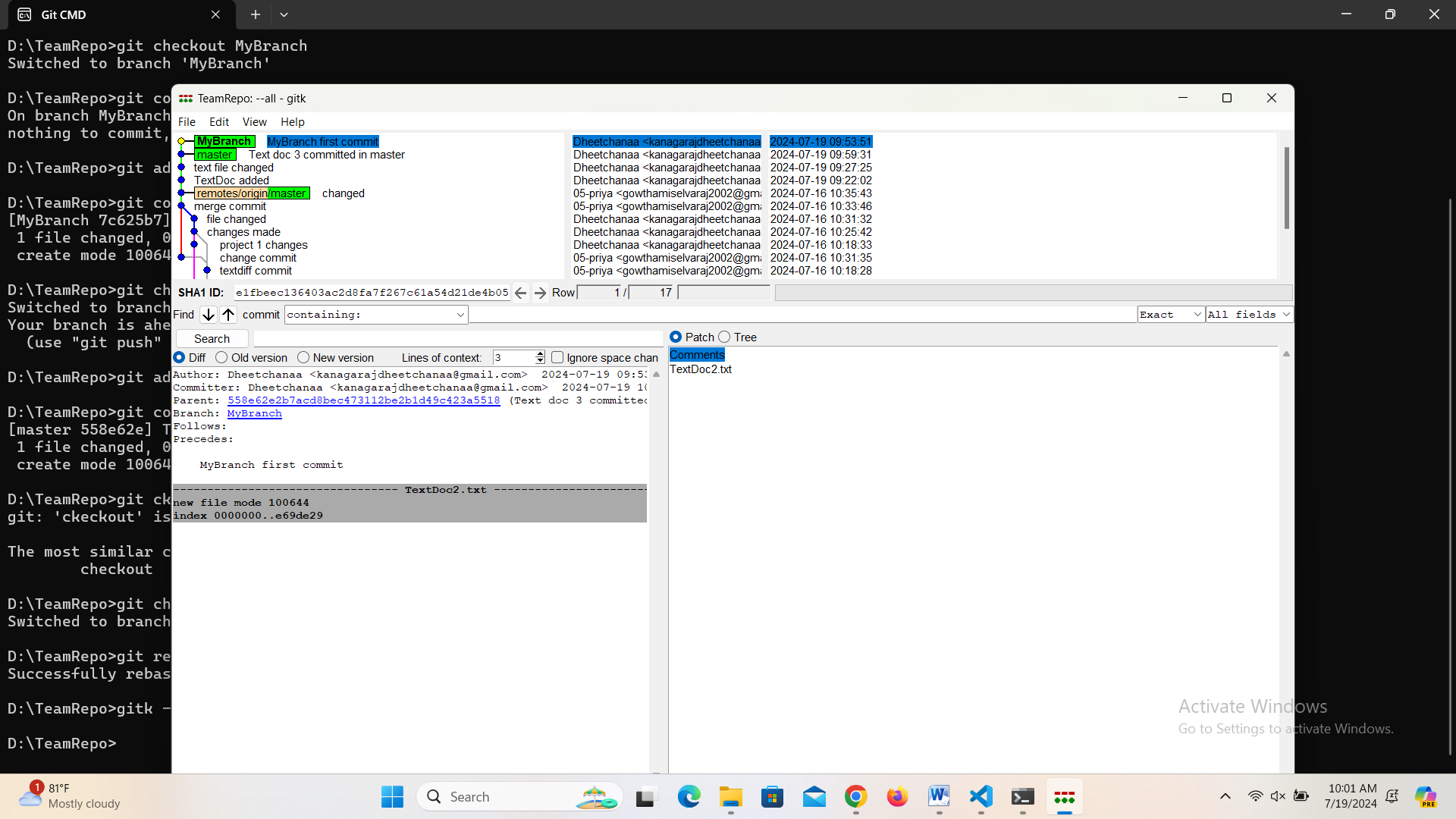
6. Now switch back to your master branch. Make a (non-conflicting) commit there also.



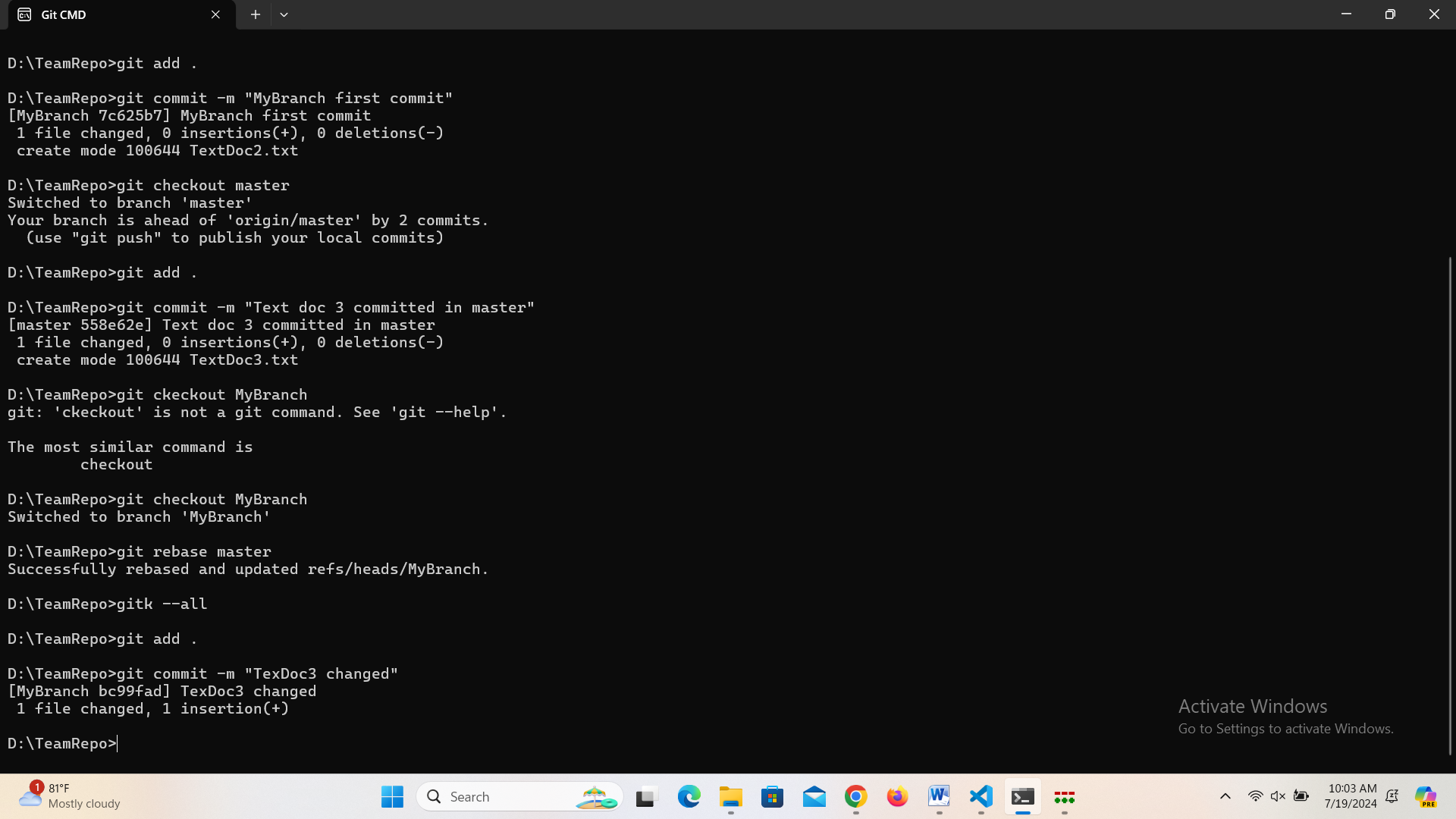
7. Now switch back to your branch.



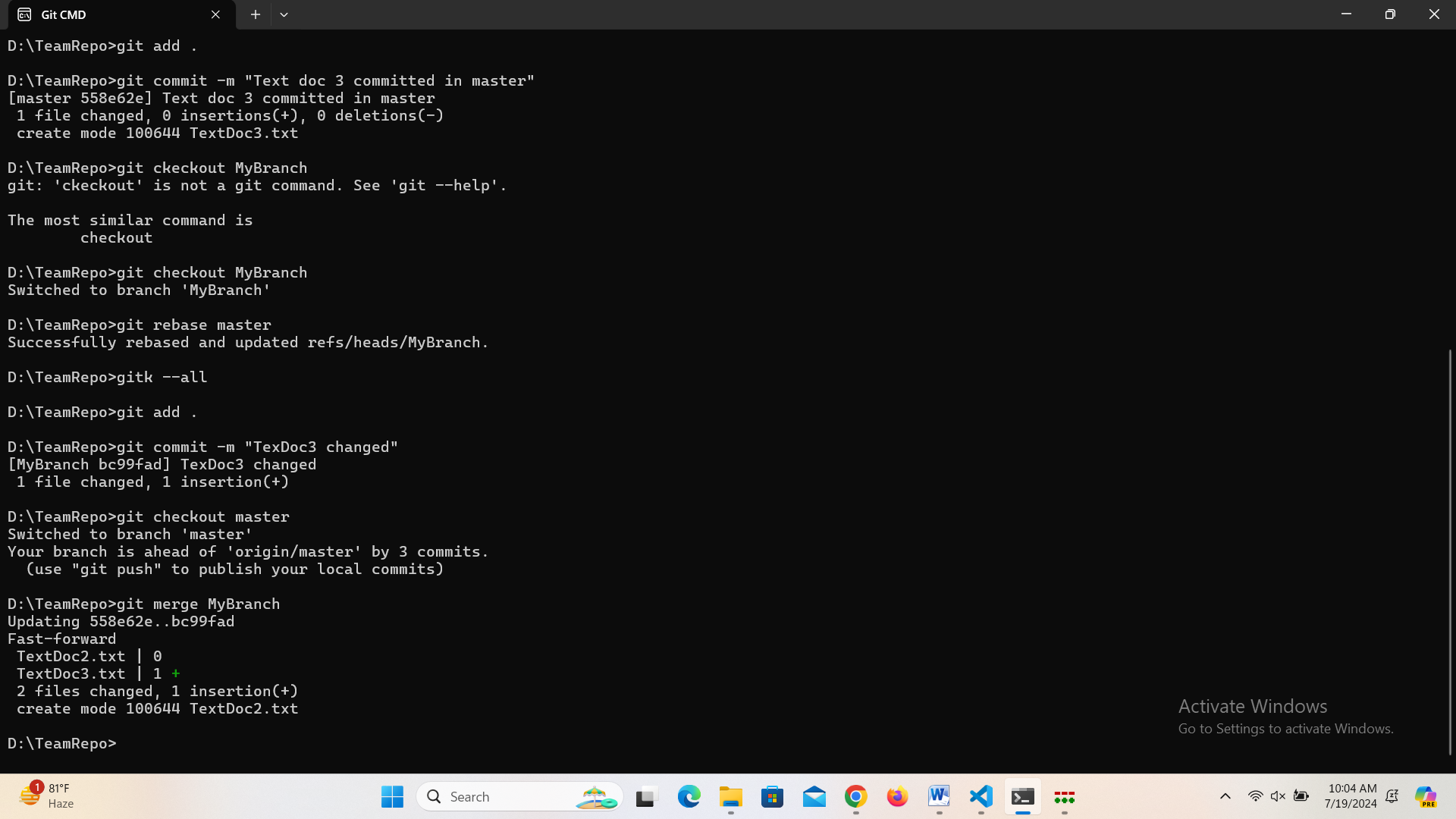
8. Use the rebase command in your branch. Look at the DAG in gitk, and note that you have the commit from the master branch, but no merge commit.



9. Make one more commit in your branch.

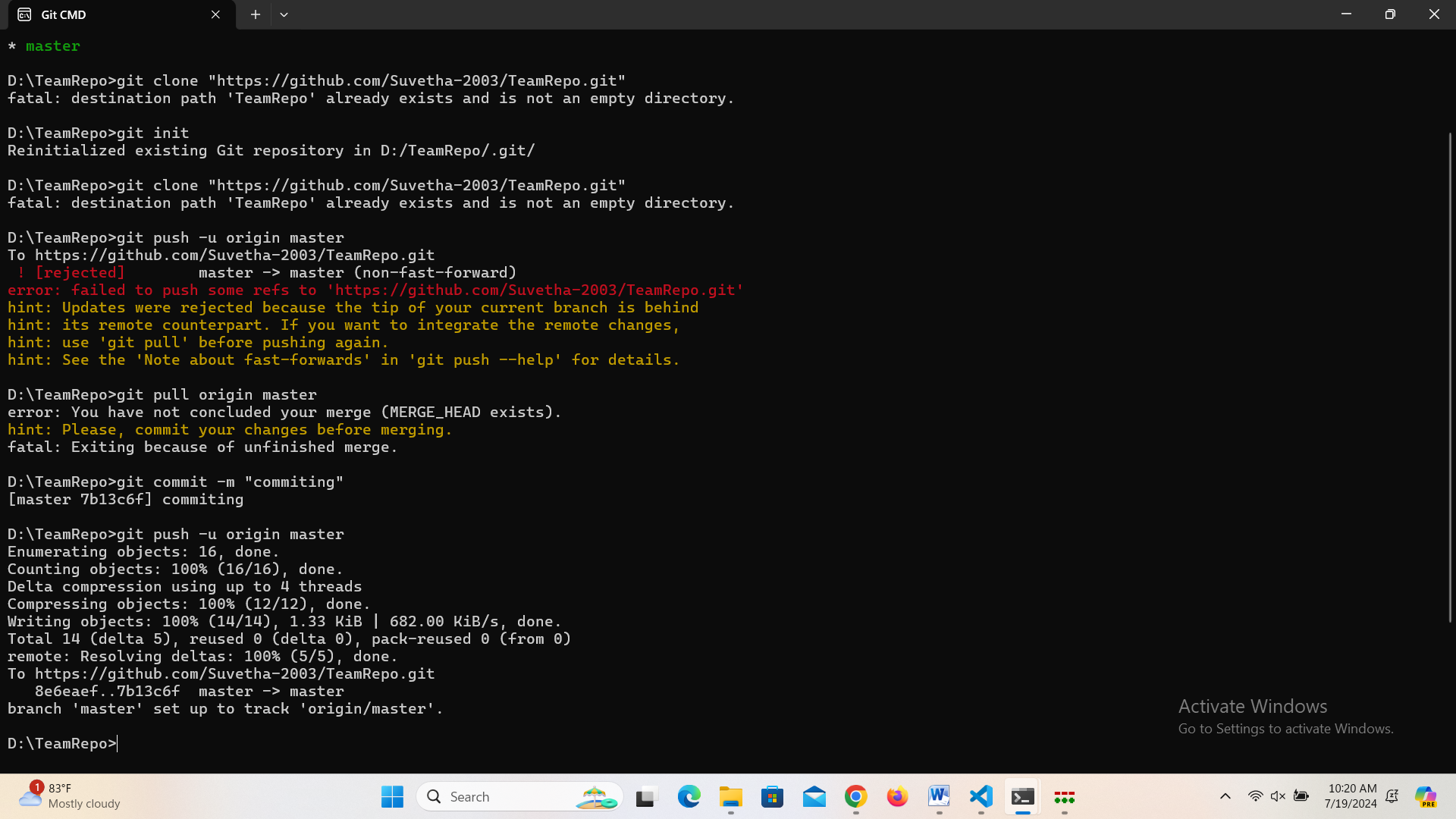


10. Return to master. Merge your branch. Notice how, thanks to the rebase, this is a fast-forward merge.

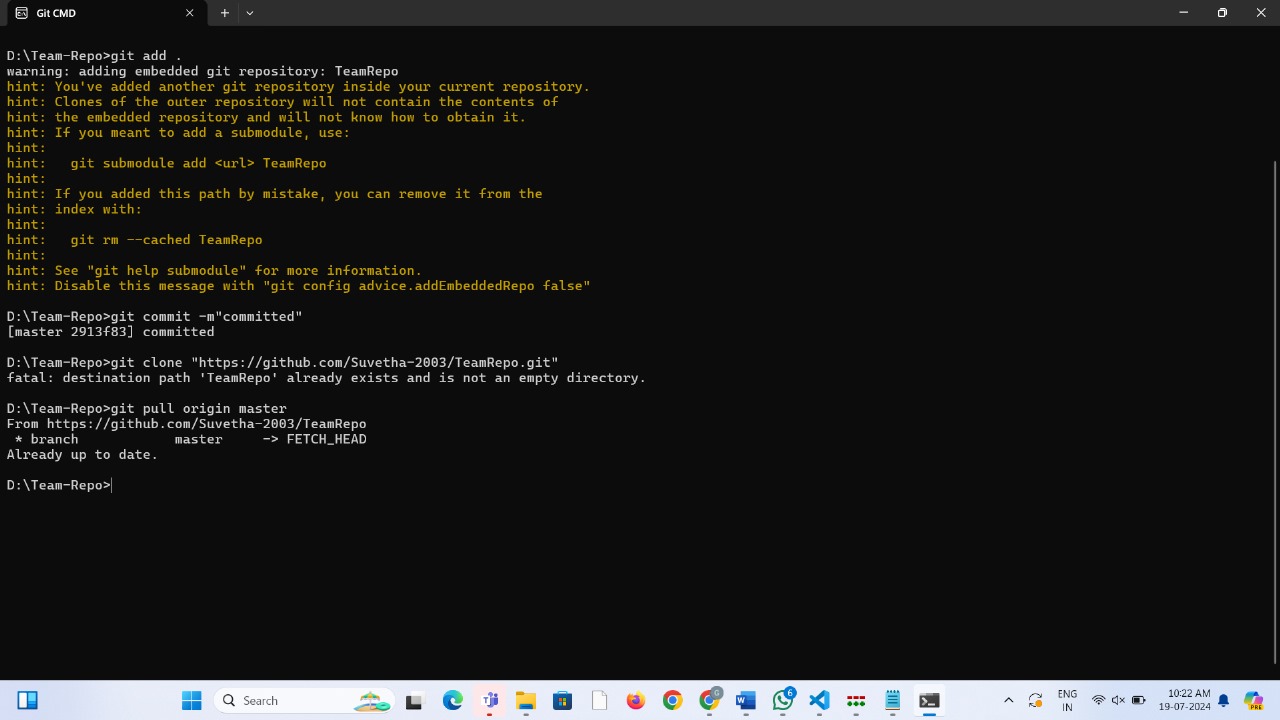


Stretch Task

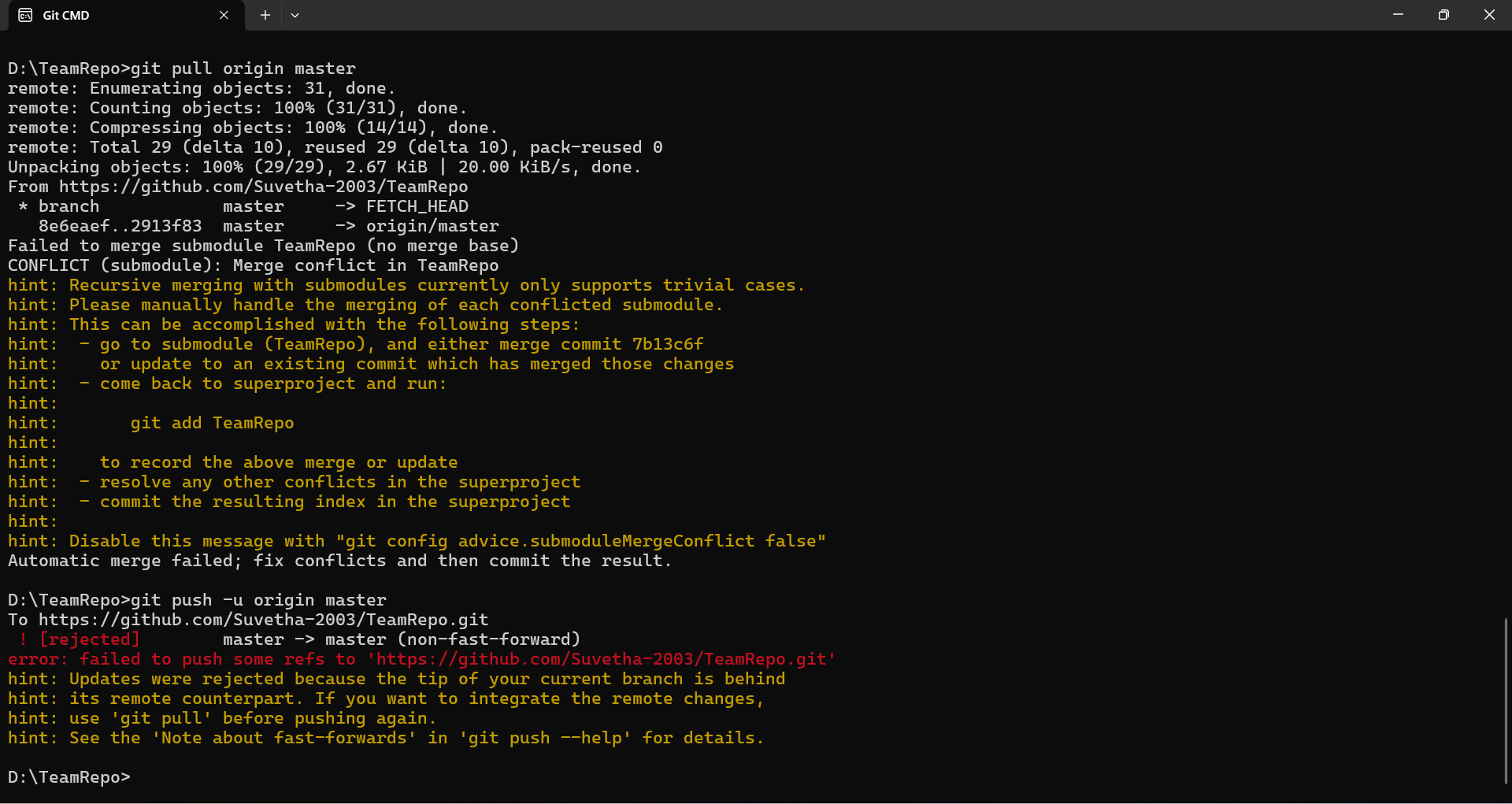
1. Find somebody from your team from the previous exercise. Have them push a commit to the central repository.



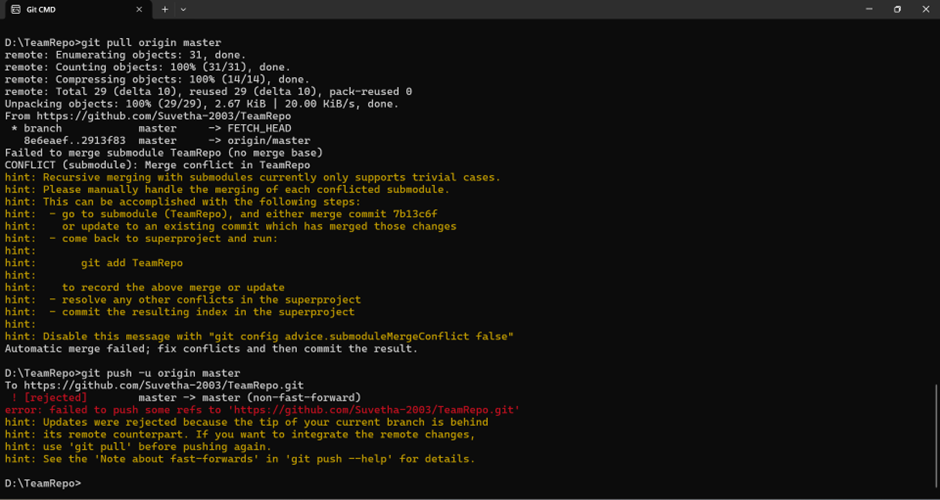
2. Make a commit locally yourself also. Note that you should not have pulled their commit at this point.



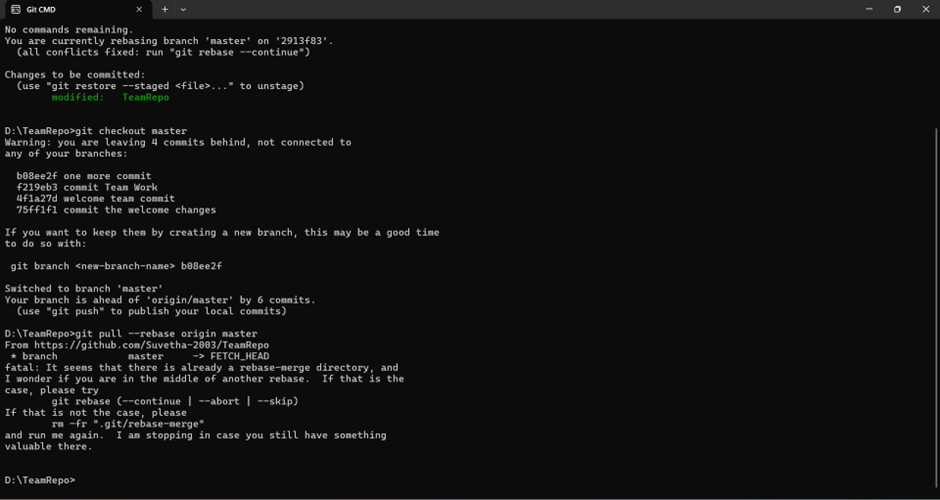
3. Try to push, and watch it fail.

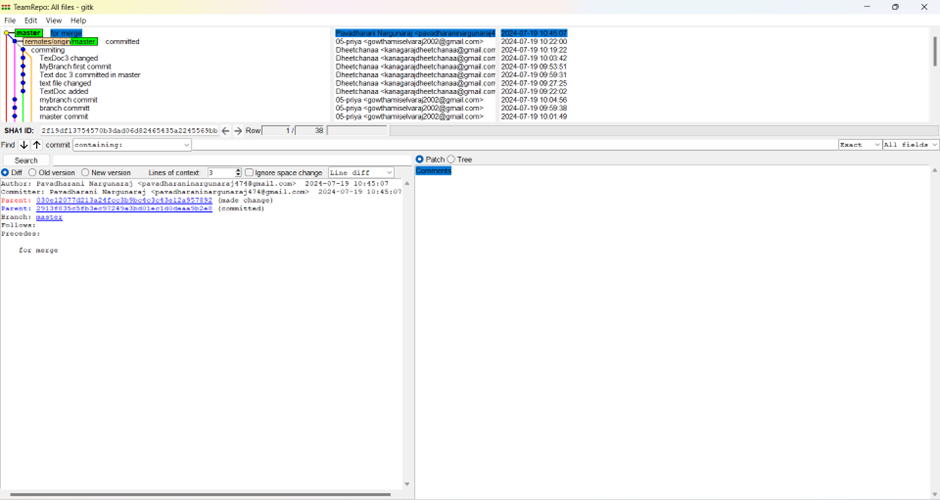


4. Now, pull but using the --rebase flag.



5. Use git log and gitk to verify that there is no merge commit, and the DAG is linear.





6. Notice that your commit is the latest one, even though temporally the other member of your team made their commit afterwards. Why is this?