**Exercise 1**

A screen shot of a computer screen

Description automatically generated1. Create a new directory and change into it.

2. Use the init command to create a Git repository in that directory.

A black background with white text

Description automatically generated

3. Observe that there is now a .git directory.

A screenshot of a computer

Description automatically generated

4. Create a README file.

A close-up of a white background

Description automatically generated

5. Look at the output of the status command; the README you created should appear as an untracked file.

A black screen with white text

Description automatically generated

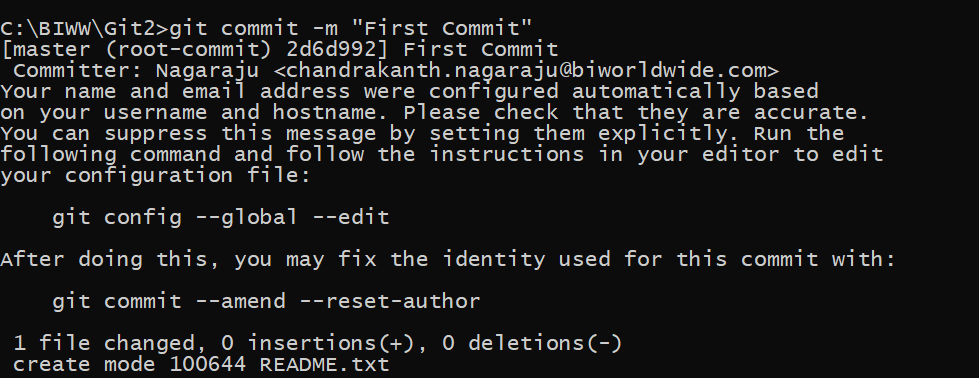
6. Use the add command to add the new file to the staging area. Again, look at the output of

the status command.

A screen shot of a computer

Description automatically generated

7. Now use the commit command to commit the contents of the staging area.



8. Create a src directory and add a couple of files to it.

A close-up of a message

Description automatically generated

A screenshot of a computer

Description automatically generated

9. Use the add command, but name the directory, not the individual files. Use the status

command. See how both files have been staged. Commit them.

A screenshot of a computer program

Description automatically generated

10. Make a change to one of the files. Use the diff command to view the details of the change.

A screen shot of a computer

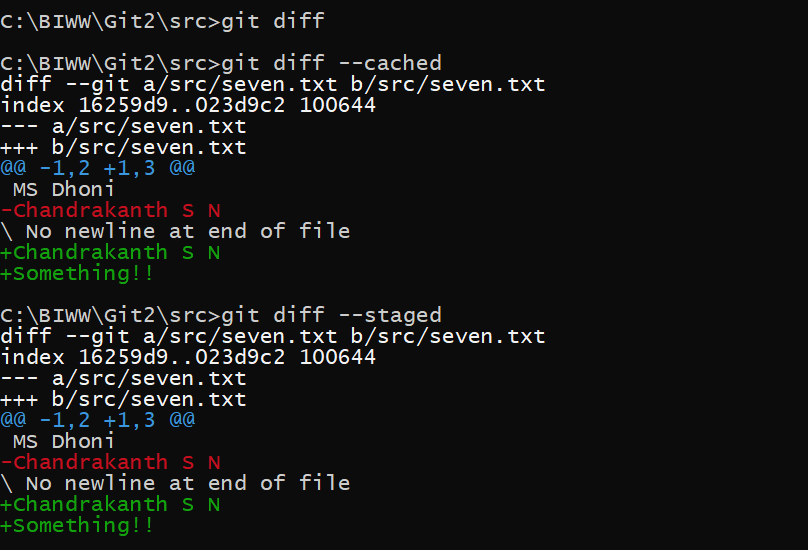
Description automatically generated

11. Next, add the changed file, and notice how it moves to the staging area in the status

output. Also observe that the diff command you did before using add now gives no output.

Why not? What do you have to do to see a diff of the things in the staging area? (Hint:

review the slides if you can’t remember.)

A computer screen with white text

Description automatically generated

12. Now – without committing – make another change to the same file you changed in step 10. Look at the status output, and the diff output. Notice how you can have both staged and unstaged changes, even when you’re talking about a single file. Observe the difference when you use the add command to stage the latest round of changes. Finally, commit them. You should now have started to get a feel for the staging area.

A computer screen shot of a program

Description automatically generated

13. Use the log command in order to see all of the commits you made so far.

A computer screen with white and yellow text

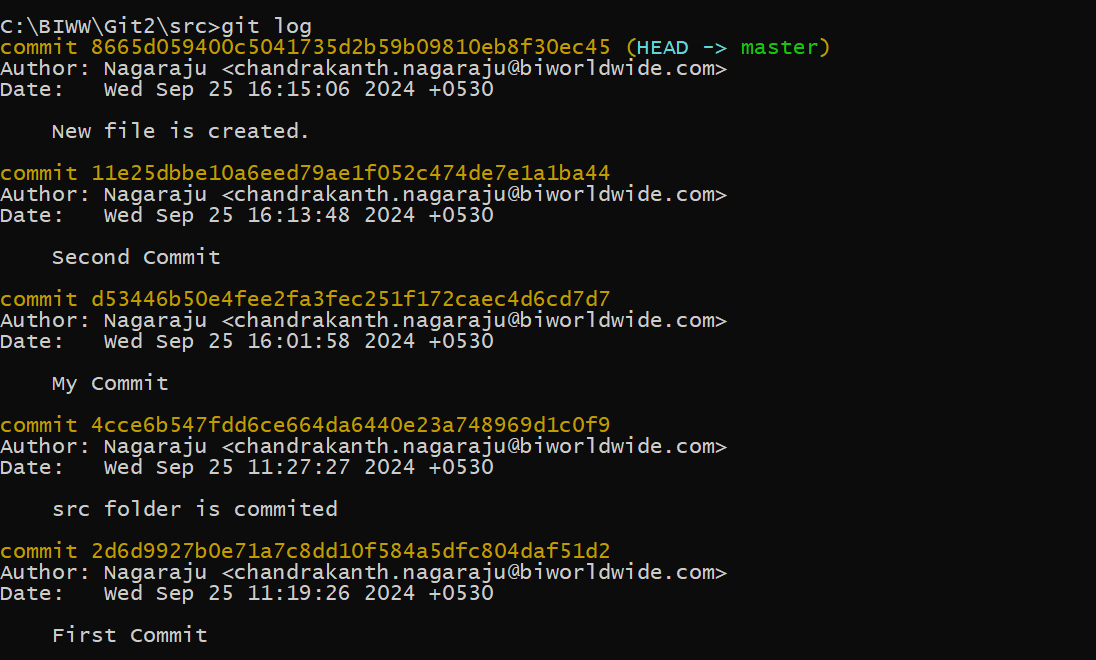
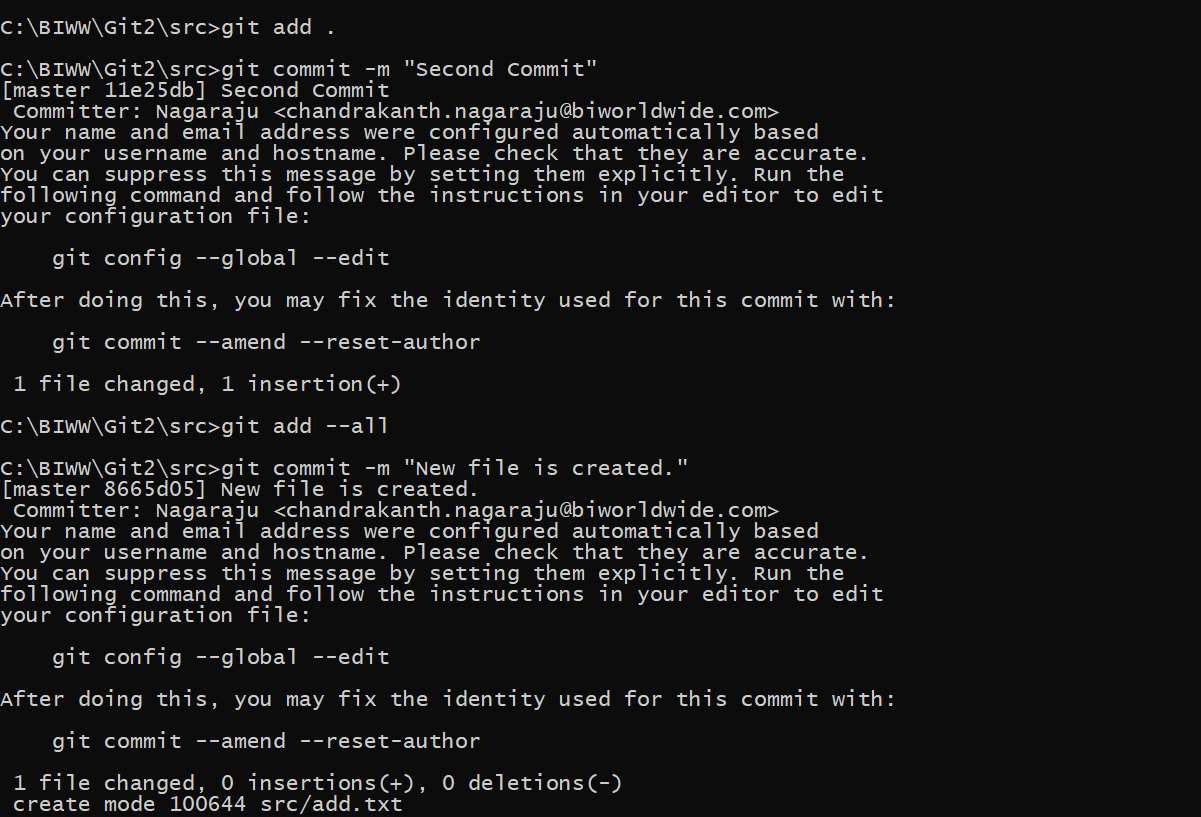
Description automatically generated

14. Use the show command to look at an individual commit. How many characters of the

commit identifier can you get away with typing at a minimum?

A screenshot of a computer program

Description automatically generated

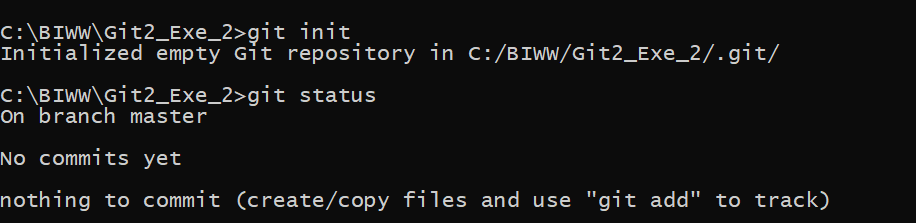
15. Make a couple more commits, at least one of which should add an extra file.A screenshot of a computer

Description automatically generatedA black screen with white text

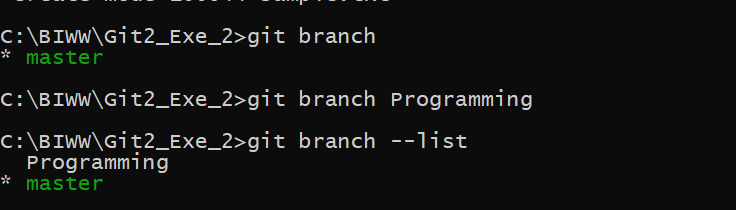
Description automatically generated

**Exercise 2**

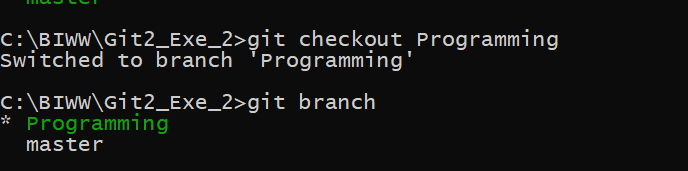
1. Run the status command. Notice how it tells you what branch you are in.



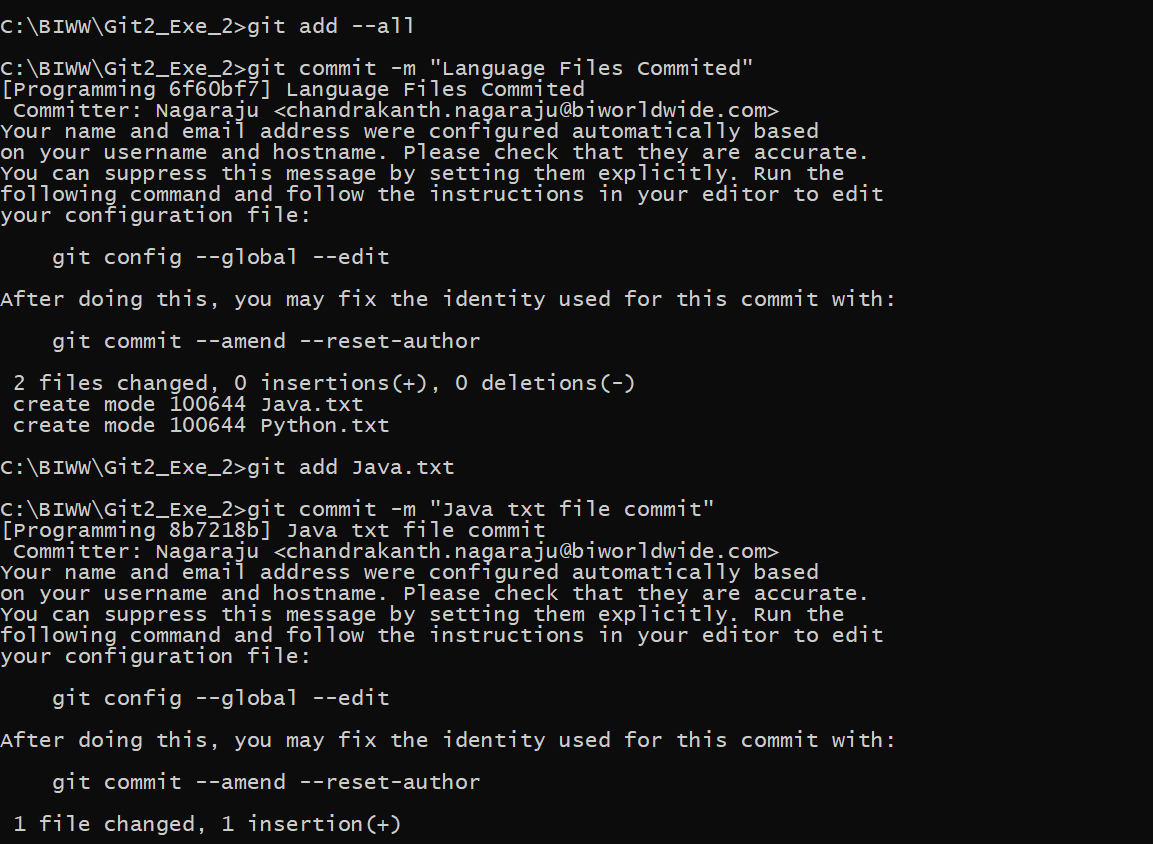
2. Use the branch command to create a new branch.



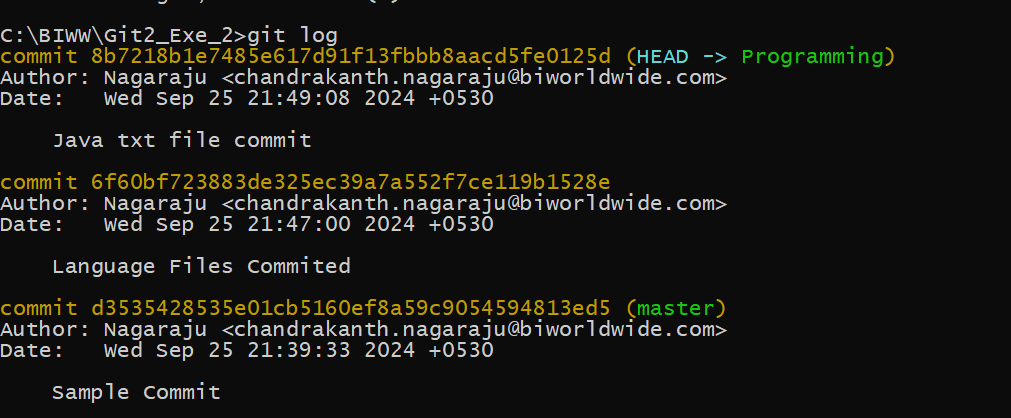
3. Use the checkout command to switch to it.



4. Make a couple of commits in the branch – perhaps adding a new file and/or editing existing ones.



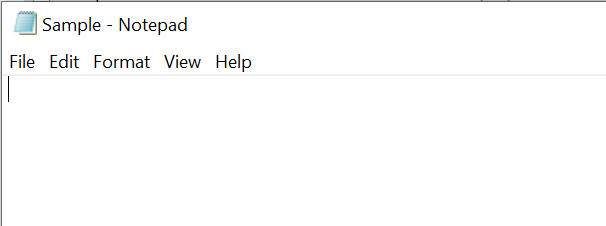
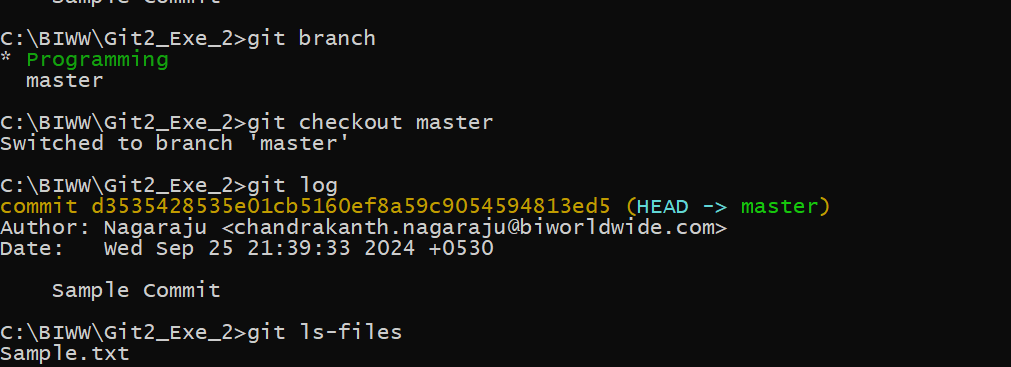
5. Use the log command to see the latest commits. The two you just made should be at the

top of the list.

6. Use the checkout command to switch back to the master branch. Run log again. Notice

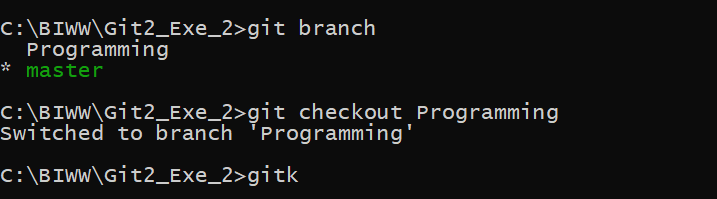
your commits don’t show up now. Check the files also – they should have their original

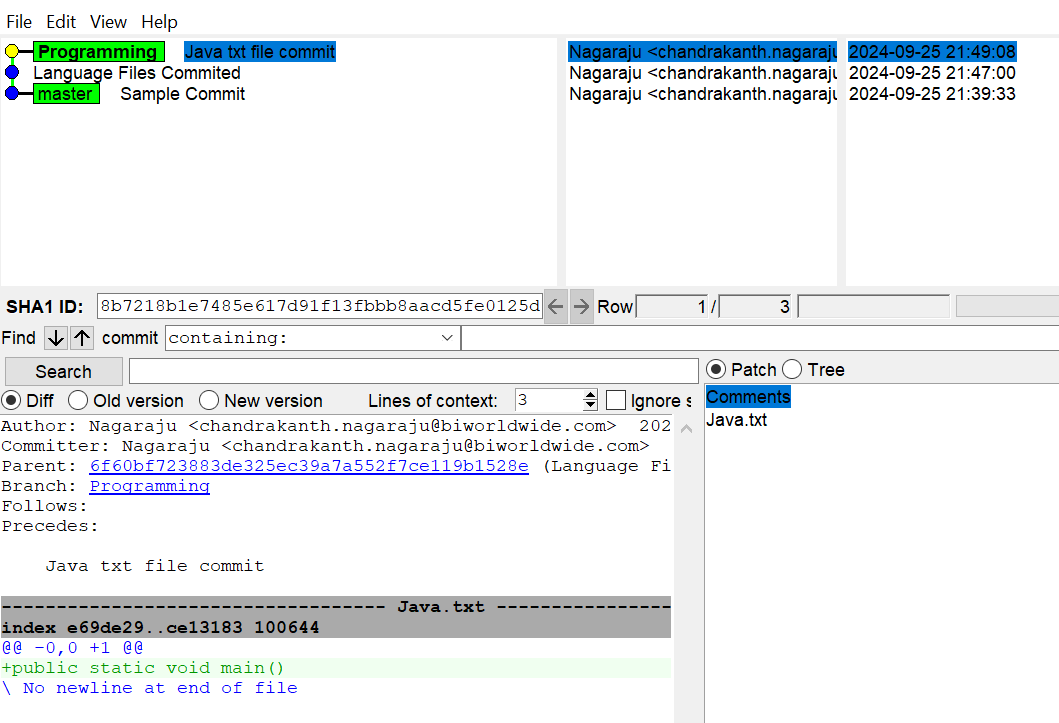
contents.



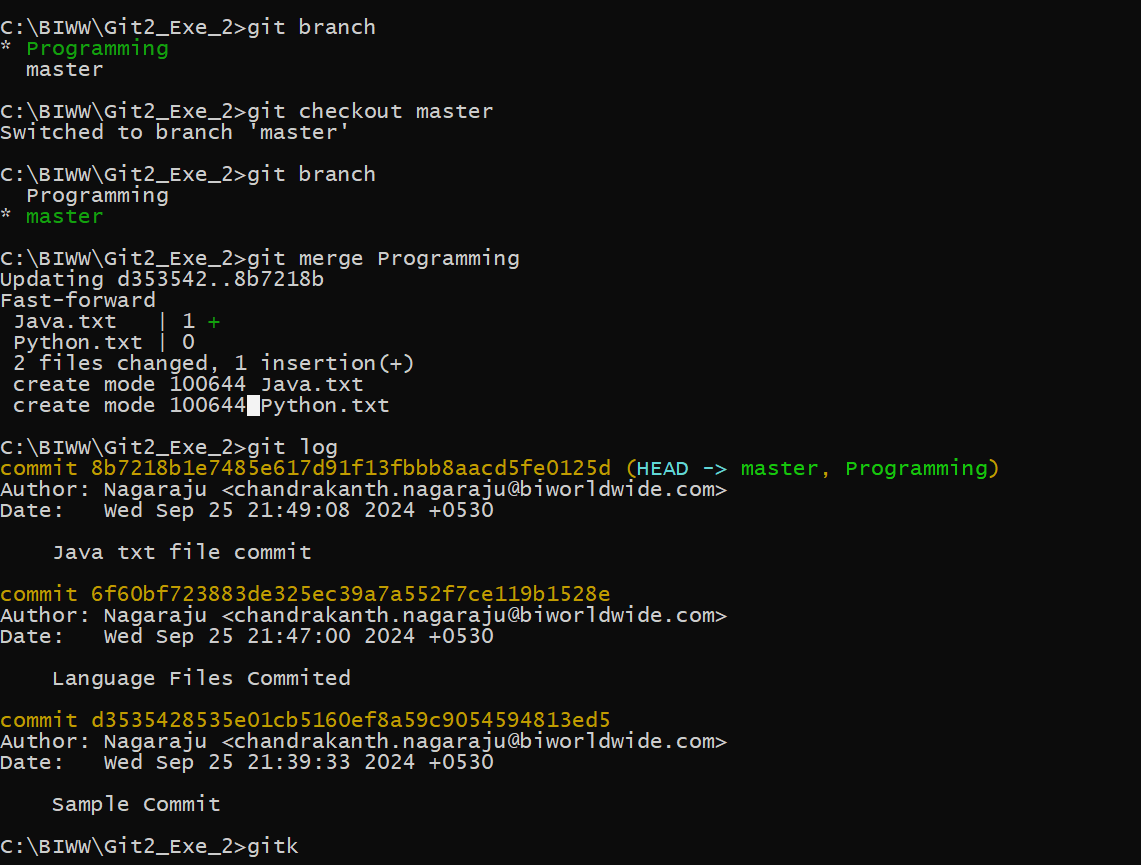
7. Use the checkout command to switch back to your branch. Use gitk to take a look at the

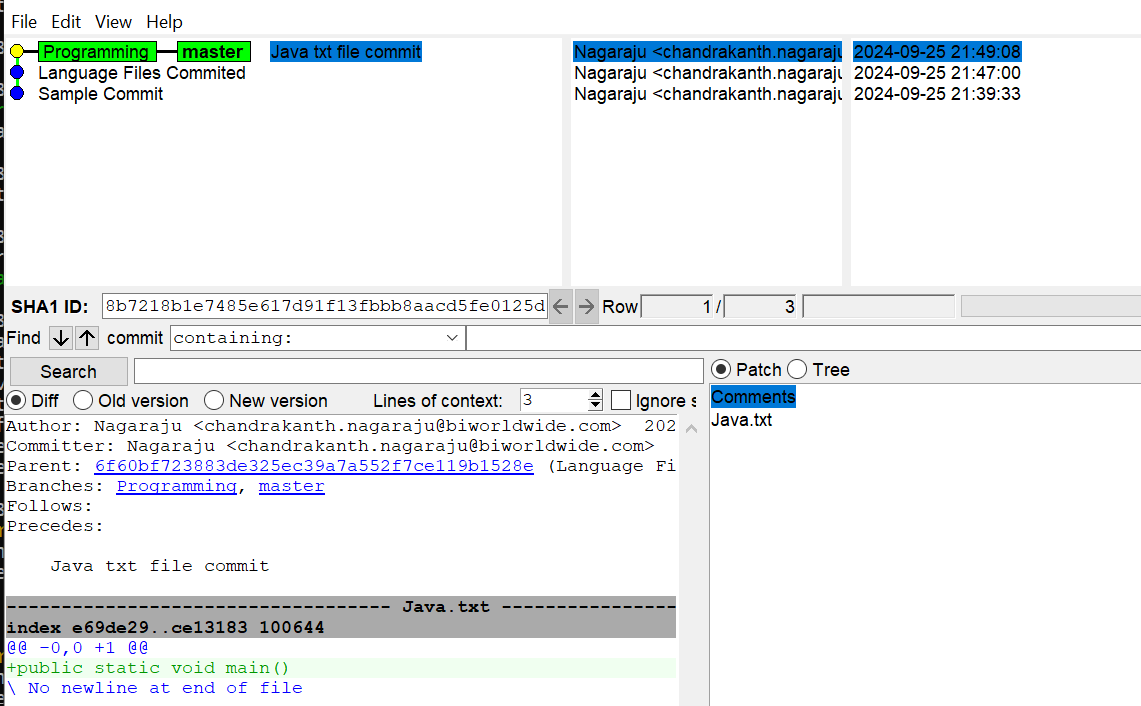
commit graph; notice it’s linear.





8. Now checkout the master branch again. Use the merge command to merge your branch into it. Look for information about it having been a fast-forward merge. Look at git log, and see that there is no merge commit. Take a look in gitk and see how the DAG is linear.





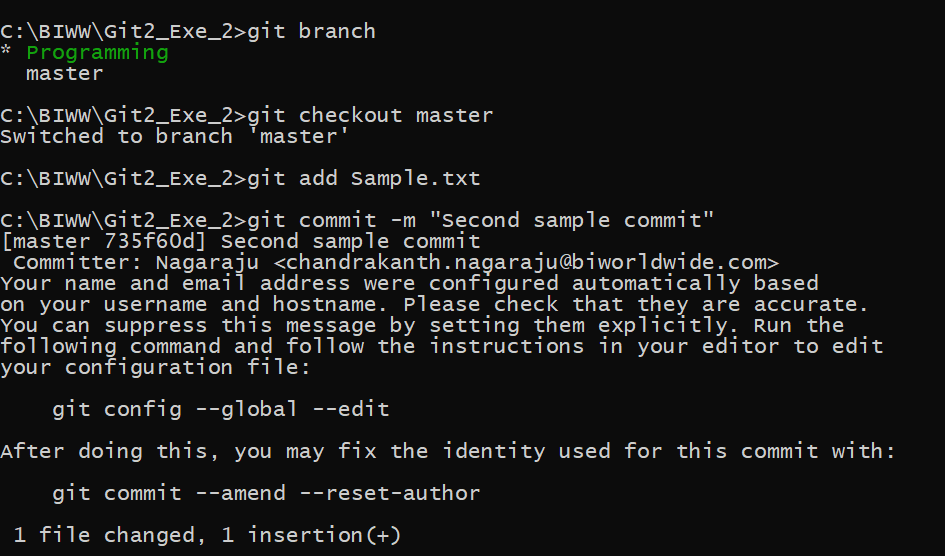
9. Switch back to your branch. Make a couple more commits.

A screenshot of a computer program

Description automatically generated

10. Switch back to master. Make a commit there, which should edit a different file from the

ones you touched in your branch – to be sure there is no conflict.



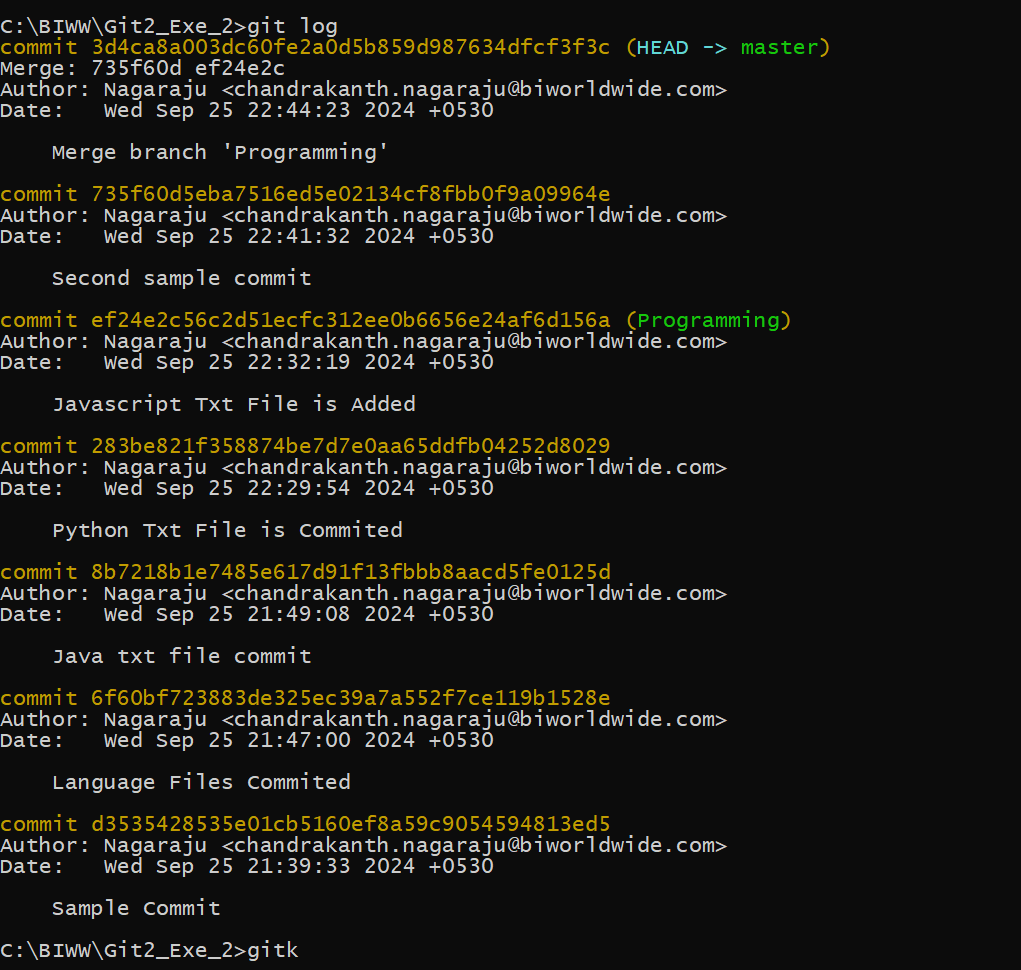
11. Now merge your branch again. (Aside: you don’t need to do anything to inform Git that you only want to merge things added since your previous merge. Due to the way Git works, that kind of issue simply does not come up, unlike in early versions of Subversion.)

A screen shot of a computer

Description automatically generated

12. Look at git log. Notice that there is a merge commit. Also look in gitk. Notice the DAG

now shows how things forked, and then were joined up again by a merge commit.



A screenshot of a computer

Description automatically generated

**Exercise 3**

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 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.