**Explain how to resolve the conflict during merge**

When Git encounters a **merge conflict**, it's unable to automatically combine the changes from two different branches. This happens when the same lines of code are modified in both branches, or when one branch deletes a file that the other branch modified. Git will pause the merge process and ask you to resolve the conflict manually.

**Steps to Resolve a Merge Conflict**

1. **Identify the Conflict**: When you run git merge and a conflict occurs, Git will tell you which files have conflicts. The merge process is now paused. You can use the command git status to see the list of files with conflicts, which are marked as "unmerged."
2. **Open the Conflicted Files**: Open the files listed in the git status output. Git adds special markers to the code to highlight the conflicting sections. These markers look like this:
3. <<<<<<< HEAD
4. This is the change in your current branch (e.g., main).
5. =======
6. This is the change from the branch you are merging (e.g., feature-branch).
7. >>>>>>> feature-branch
8. **Edit the File**: Manually edit the conflicted section. You need to decide which changes to keep, or if you want to combine them. Delete the conflict markers (<<<<<<<, =======, >>>>>>>) and edit the code to the final, correct version.
9. **Stage the Resolved File**: After you've resolved all the conflicts in a file, you need to tell Git that you're done. You do this by staging the file with git add.

Bash

git add <file-name>

1. **Complete the Merge**: Once all conflicted files have been resolved and staged, you can finalize the merge with a commit.

Bash

git commit -m "Merge branch 'feature-branch' into main"

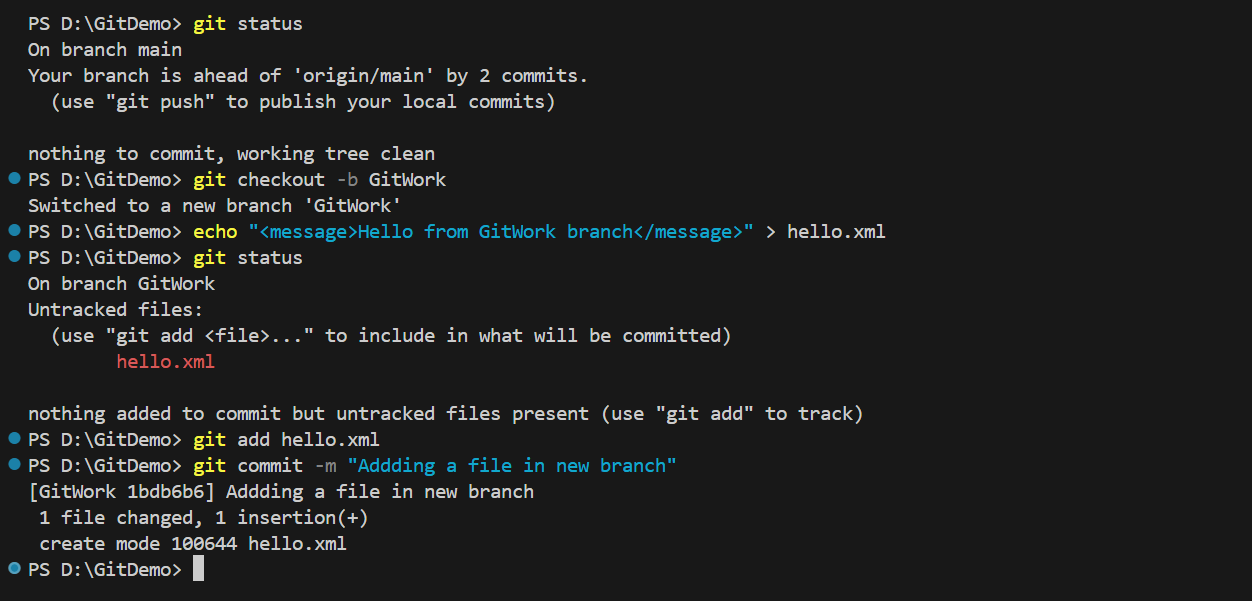
Git often pre-fills a merge commit message for you, which you can use or edit as needed. This commit officially completes the merge and integrates the changes from both branches.

**1. Verify if master is in clean state.**

**2. Create a branch “GitWork”. Add a file “hello.xml”.**

**3. Update the content of “hello.xml” and observe the status**

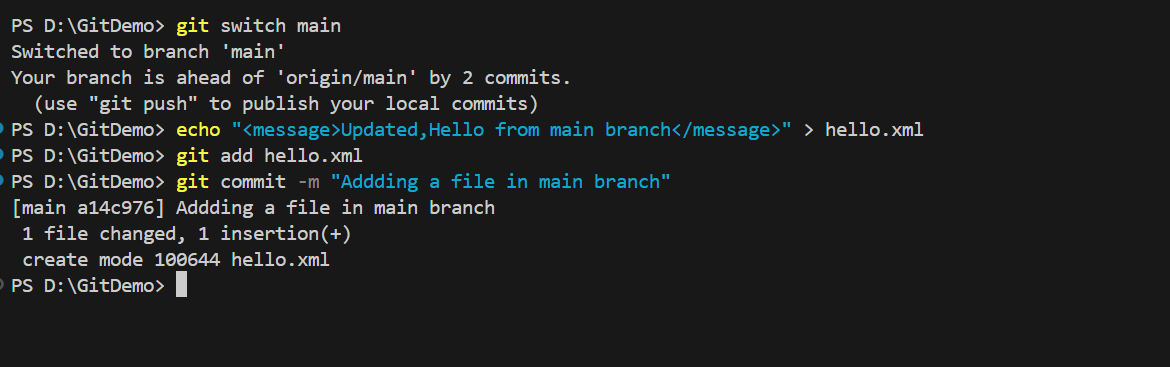
**4. Commit the changes to reflect in the branch**

****

**Switch to master.**

**6. Add a file “hello.xml” to the master and add some different content than previous.**

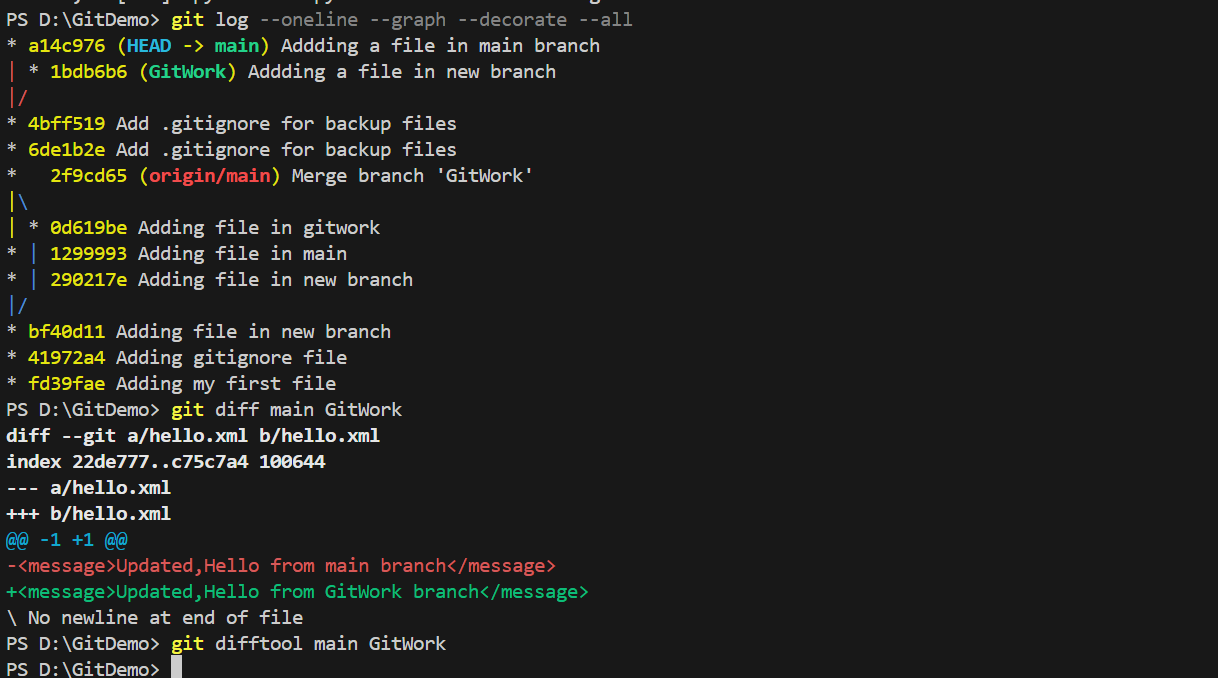
**7. Commit the changes to the master**

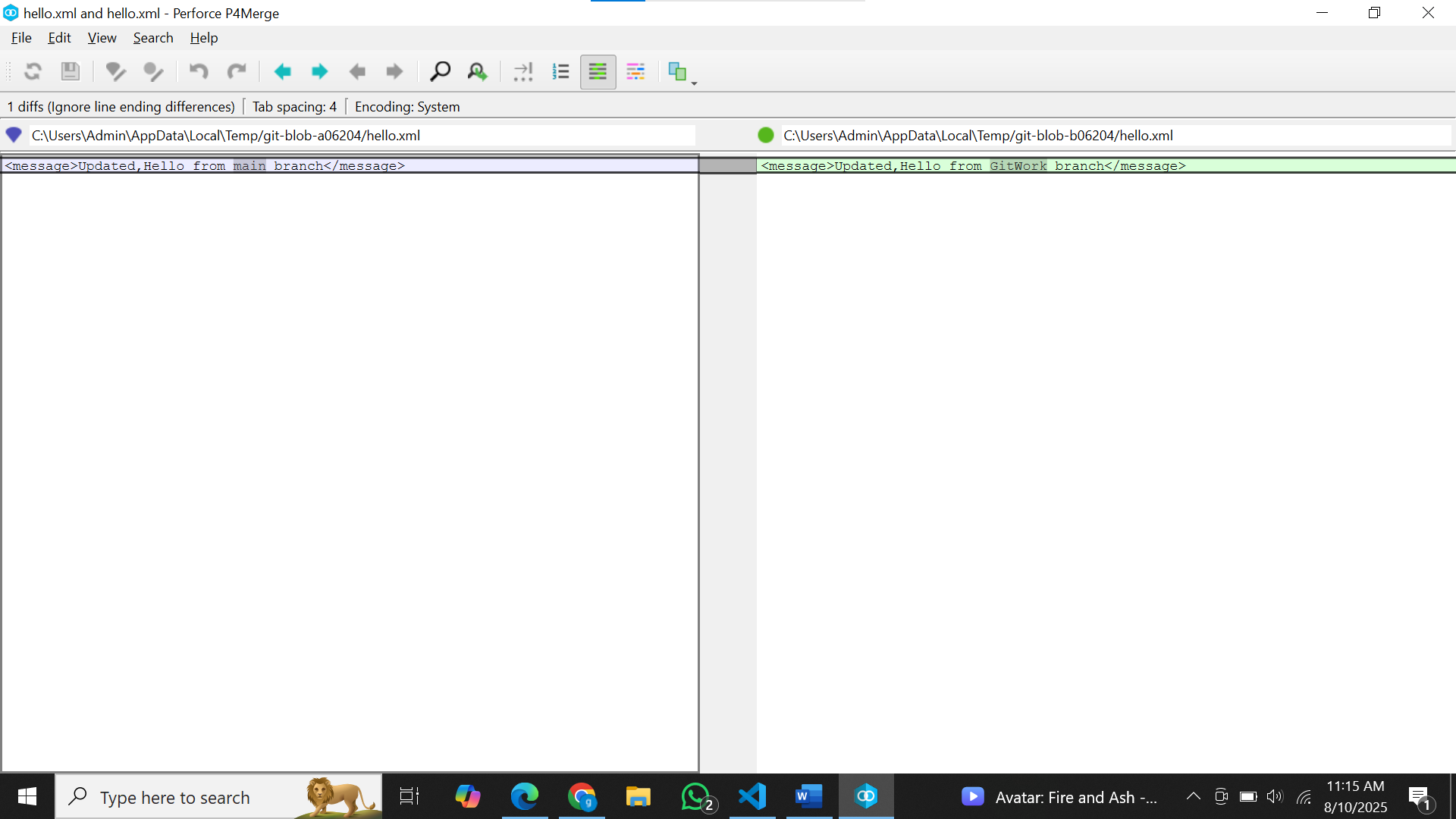
****

**Observe the log by executing “git log –oneline –graph –decorate –all”**

**9. Check the differences with Git diff tool**

**10. For better visualization, use P4Merge tool to list out all the differences between master and branch**

****

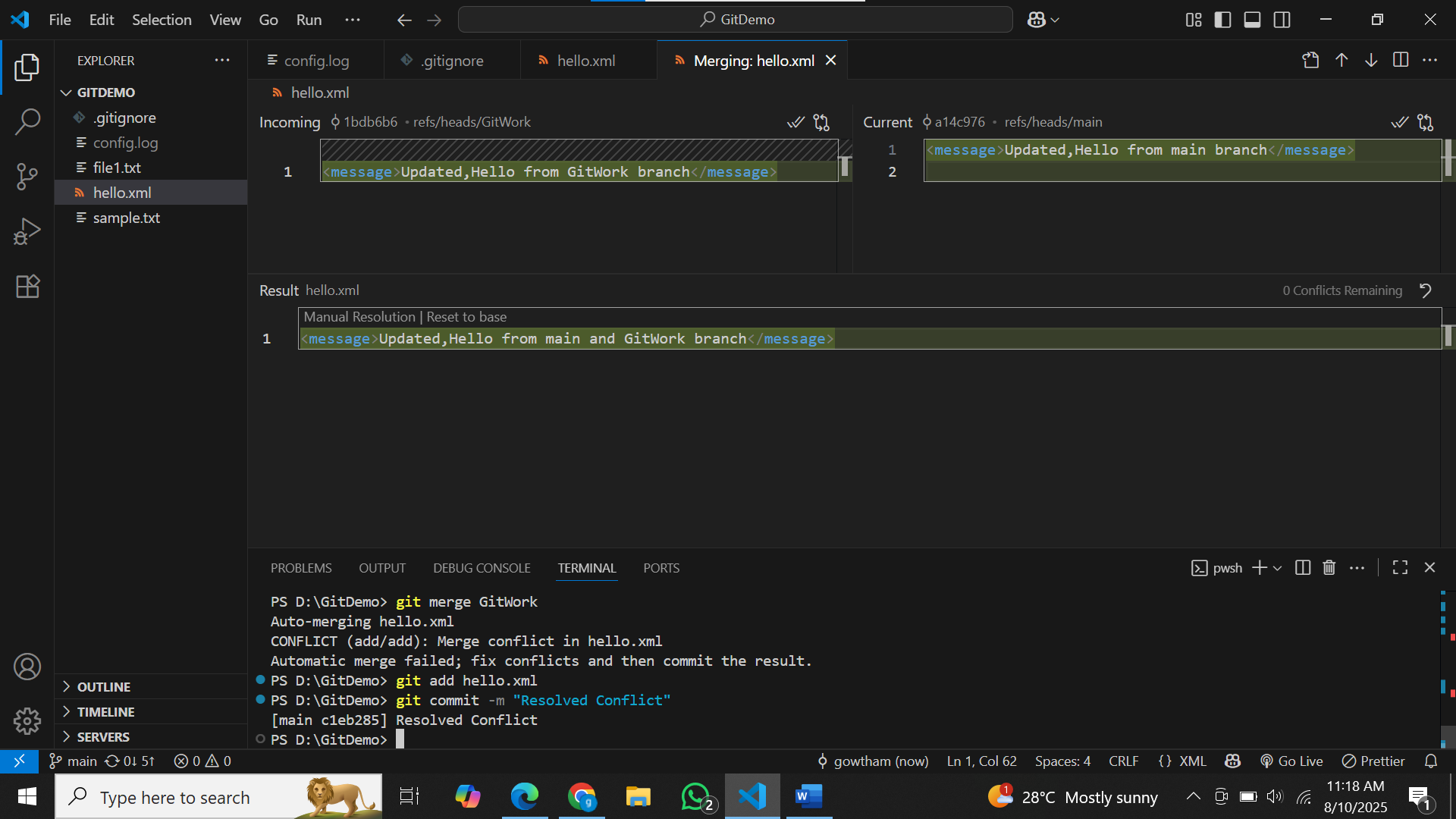
****

**Merge the bran to the master**

**12. Observe the git mark up.**

**13. Use 3-way merge tool to resolve the conflict**

**14. Commit the changes to the master, once done with conflict**

****

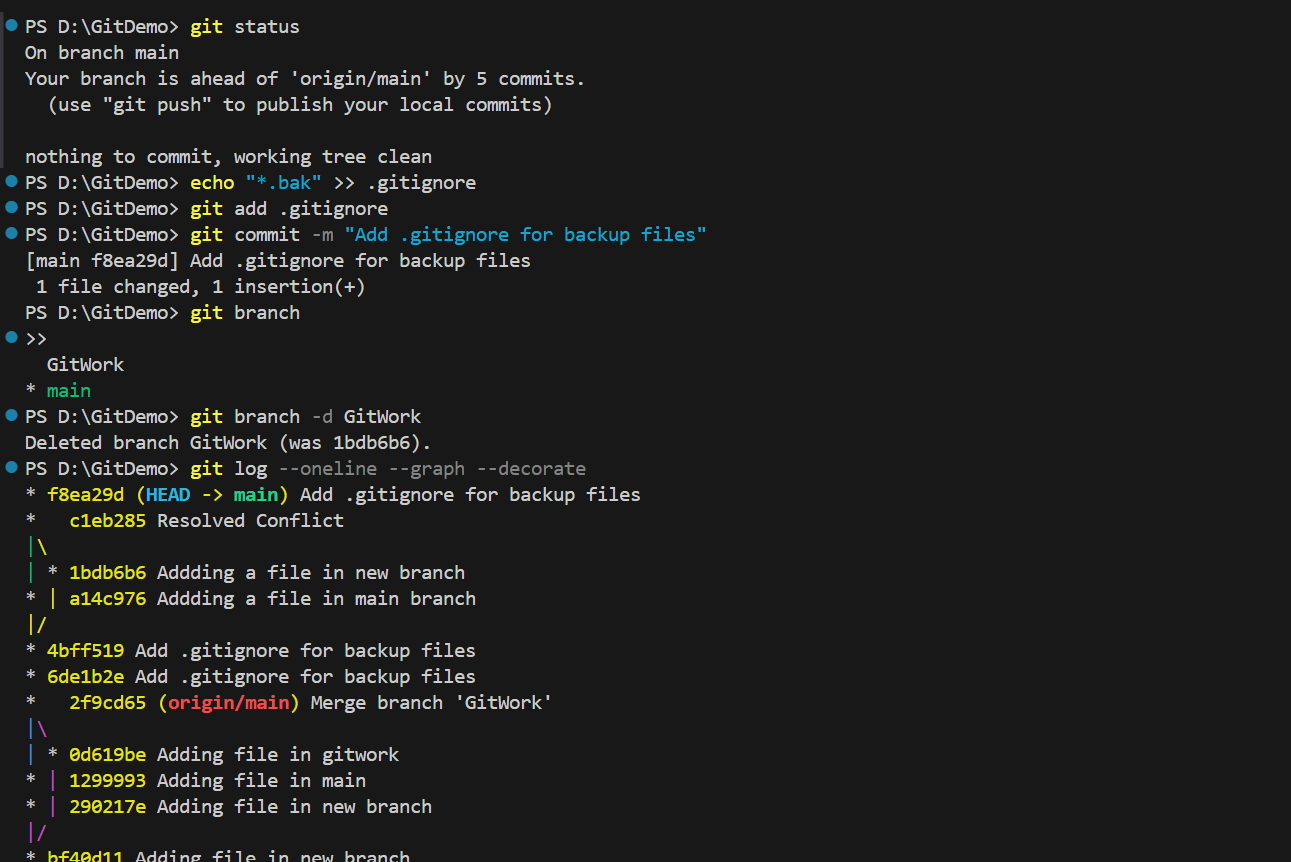
**15. Observe the git status and add backup file to the .gitignore file.**

**16. Commit the changes to the .gitignore**

**17. List out all the available branches**

**18. Delete the branch, which merge to master.**

**19. Observe the log by executing “git log –oneline –graph –decorate”**

****