**4. Git – HOL**

**Objectives:**

* Learn how to resolve conflicts during a merge.
* Practice creating a merge conflict and using a 3-way merge tool (P4Merge) to fix it.

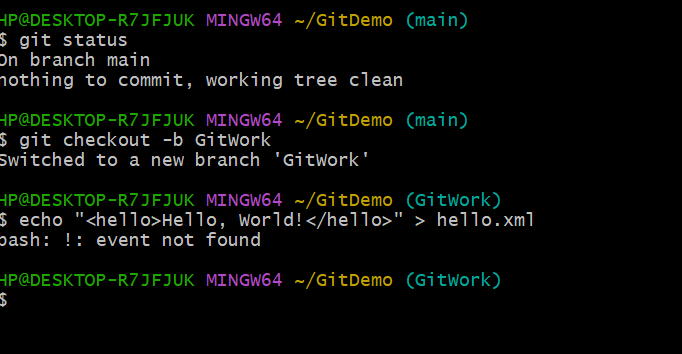
**Prerequisites:**

* A working Git environment in a local repository (like your GitDemo project).
* P4Merge is installed and configured as your difftool and mergetool.

**Hands-on Lab Steps (Detailed Break-down):**

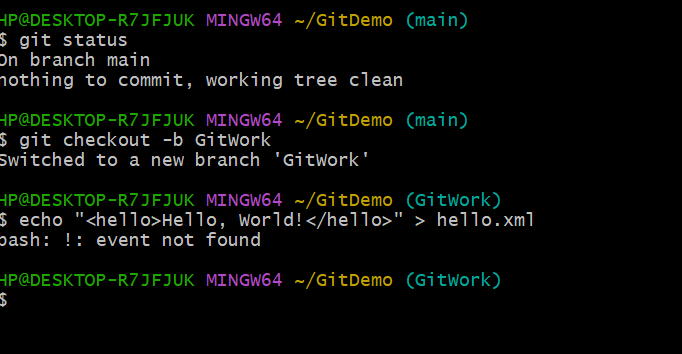
Let's assume you're in your GitDemo directory in the Git Bash shell.

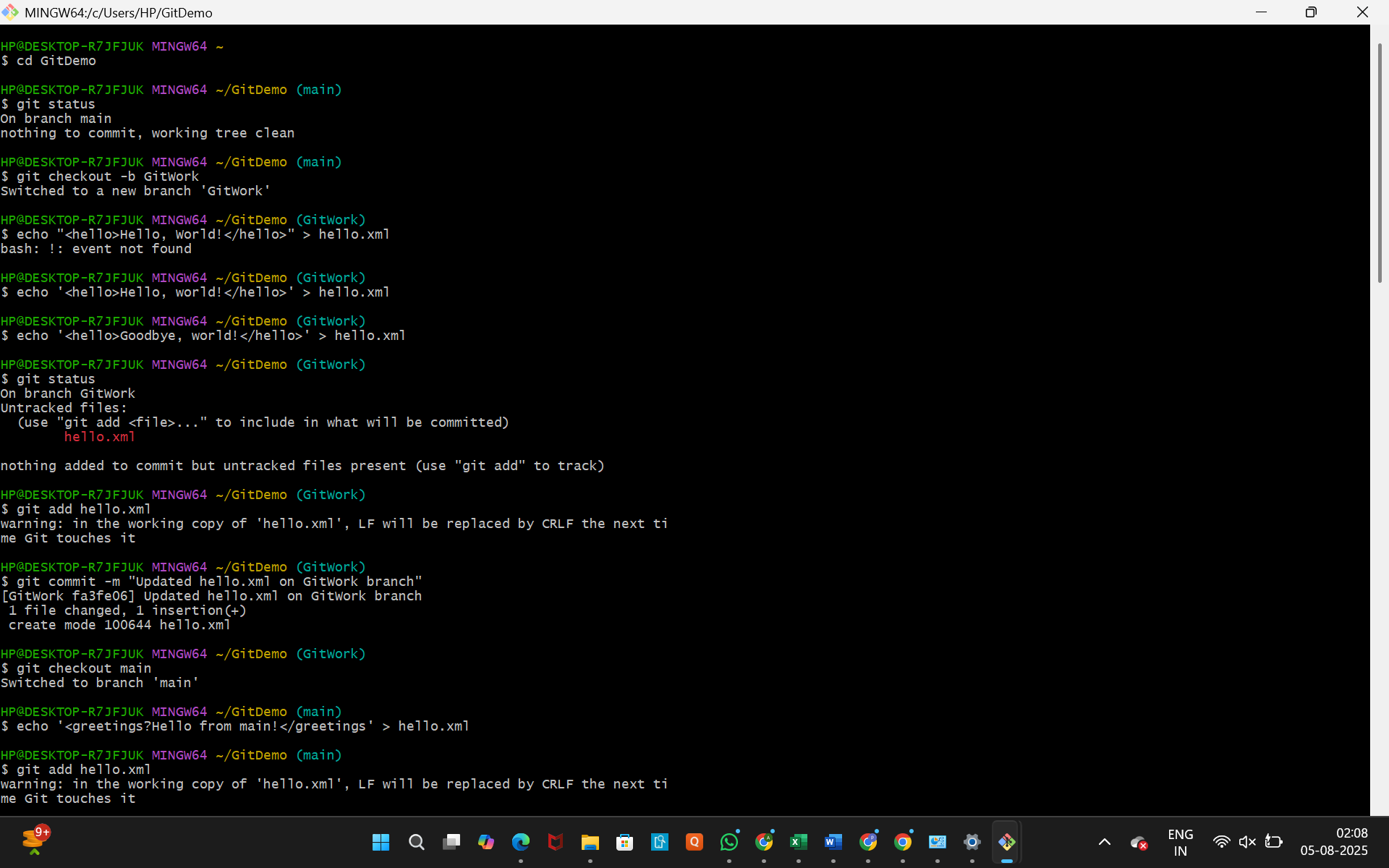
1. **Verify if master is in a clean state.**
   * **Command:**



* + **What to expect:** You should see On branch main and nothing to commit, working tree clean.

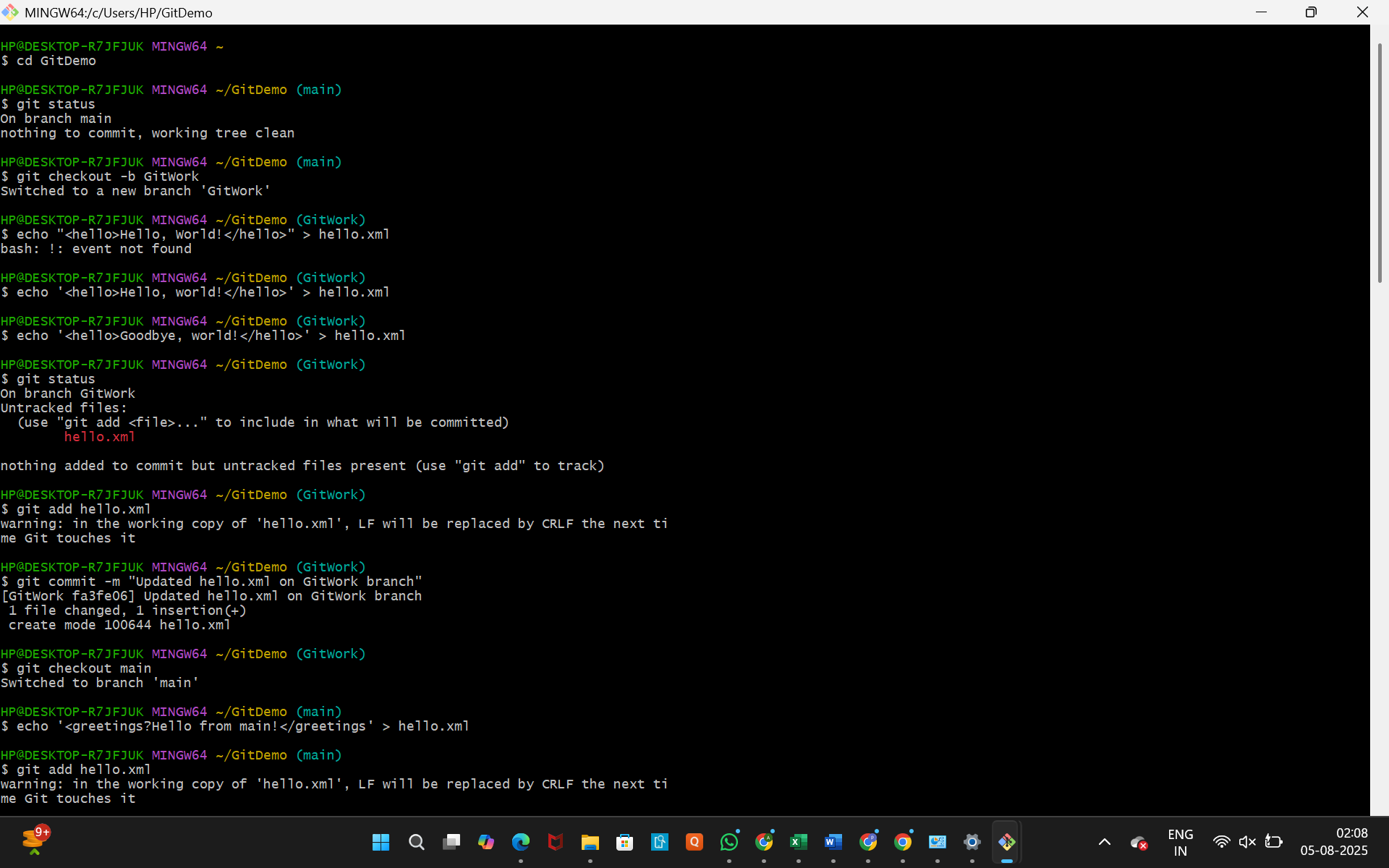
1. **Create a branch "GitWork". Add a file "hello.xml".**
   * **Command:**

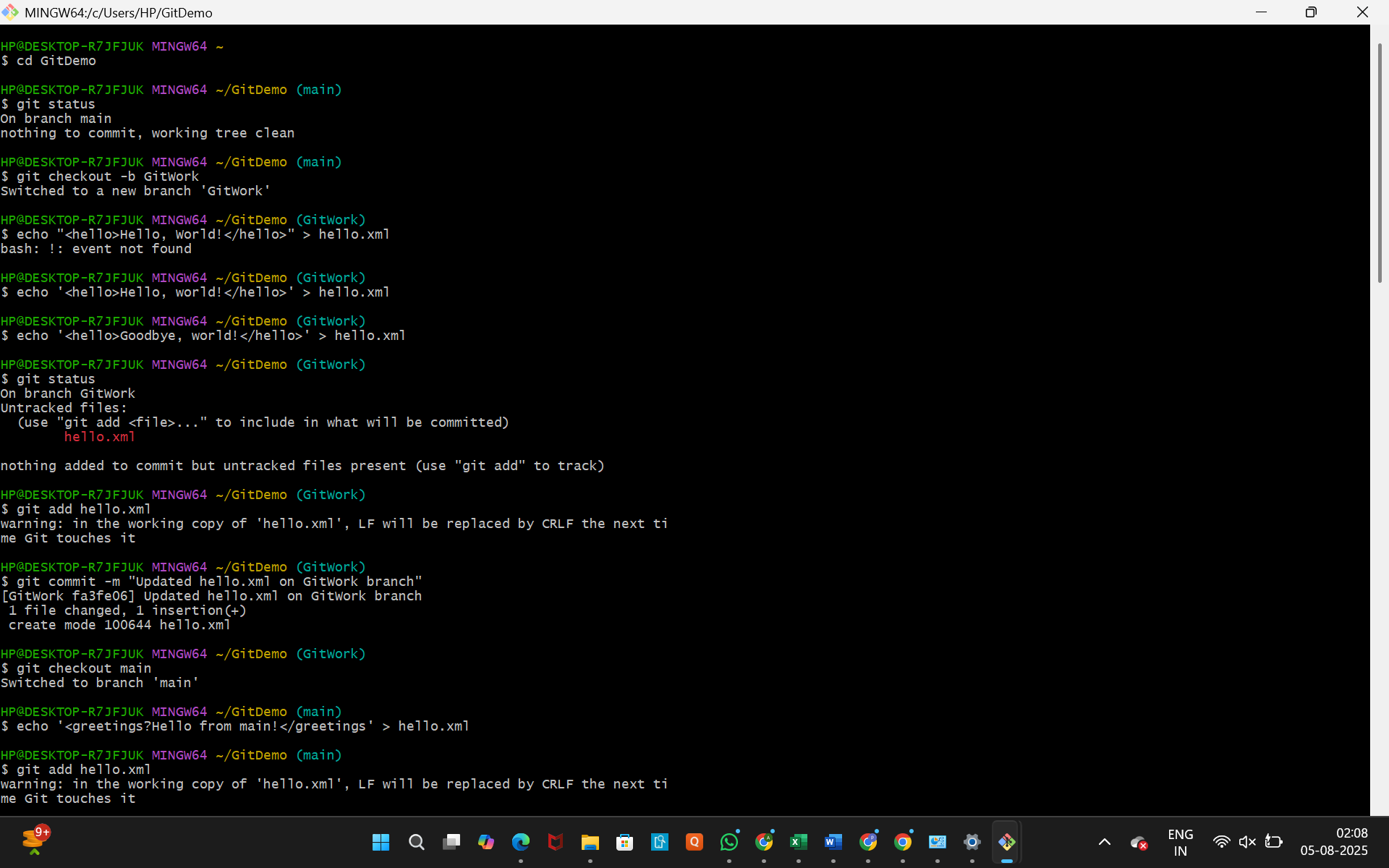




* + **Explanation:** This creates a new branch named GitWork, switches to it, and creates a new file.

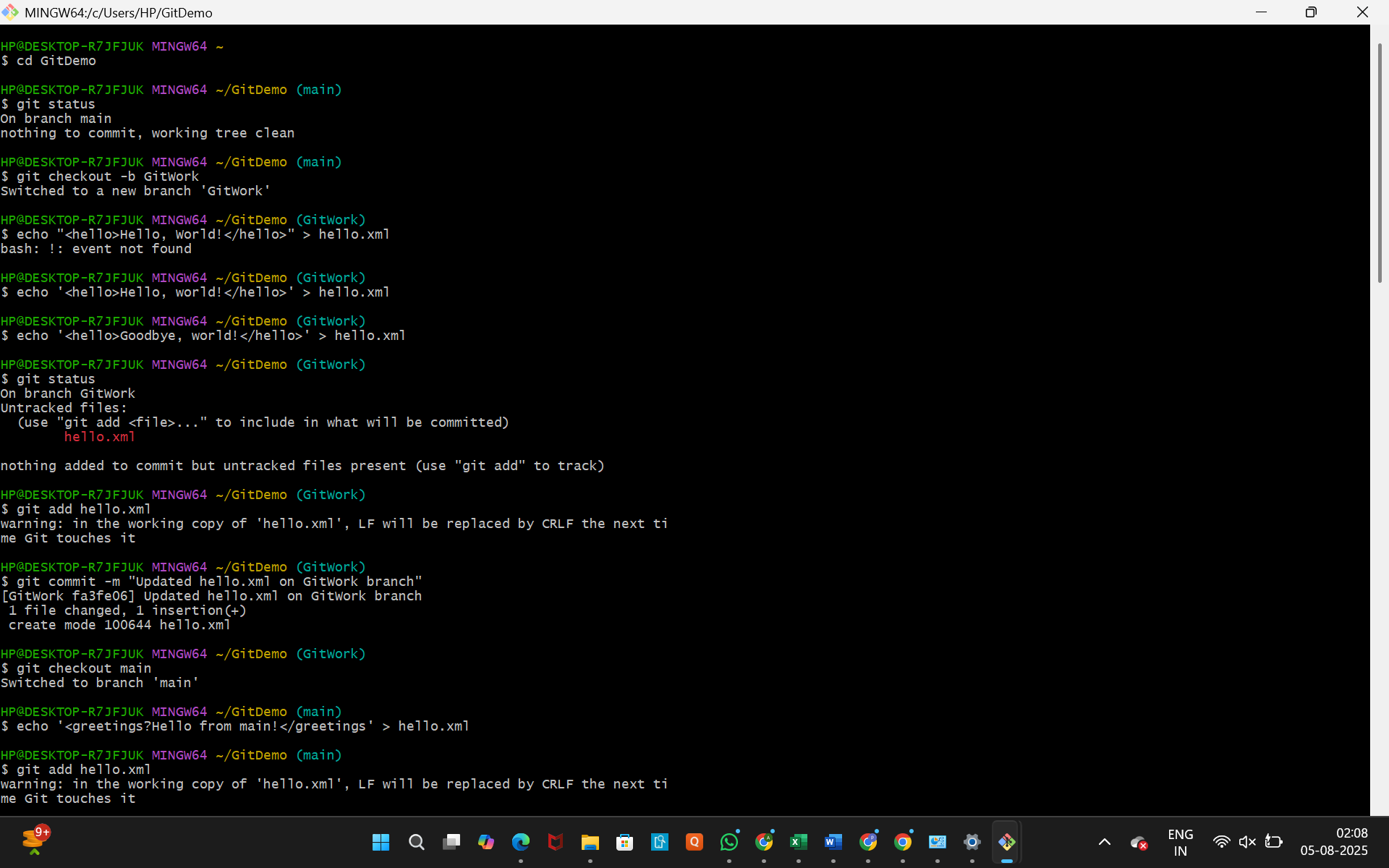
1. **Update the content of "hello.xml" and observe the status.**
   * **Command:**



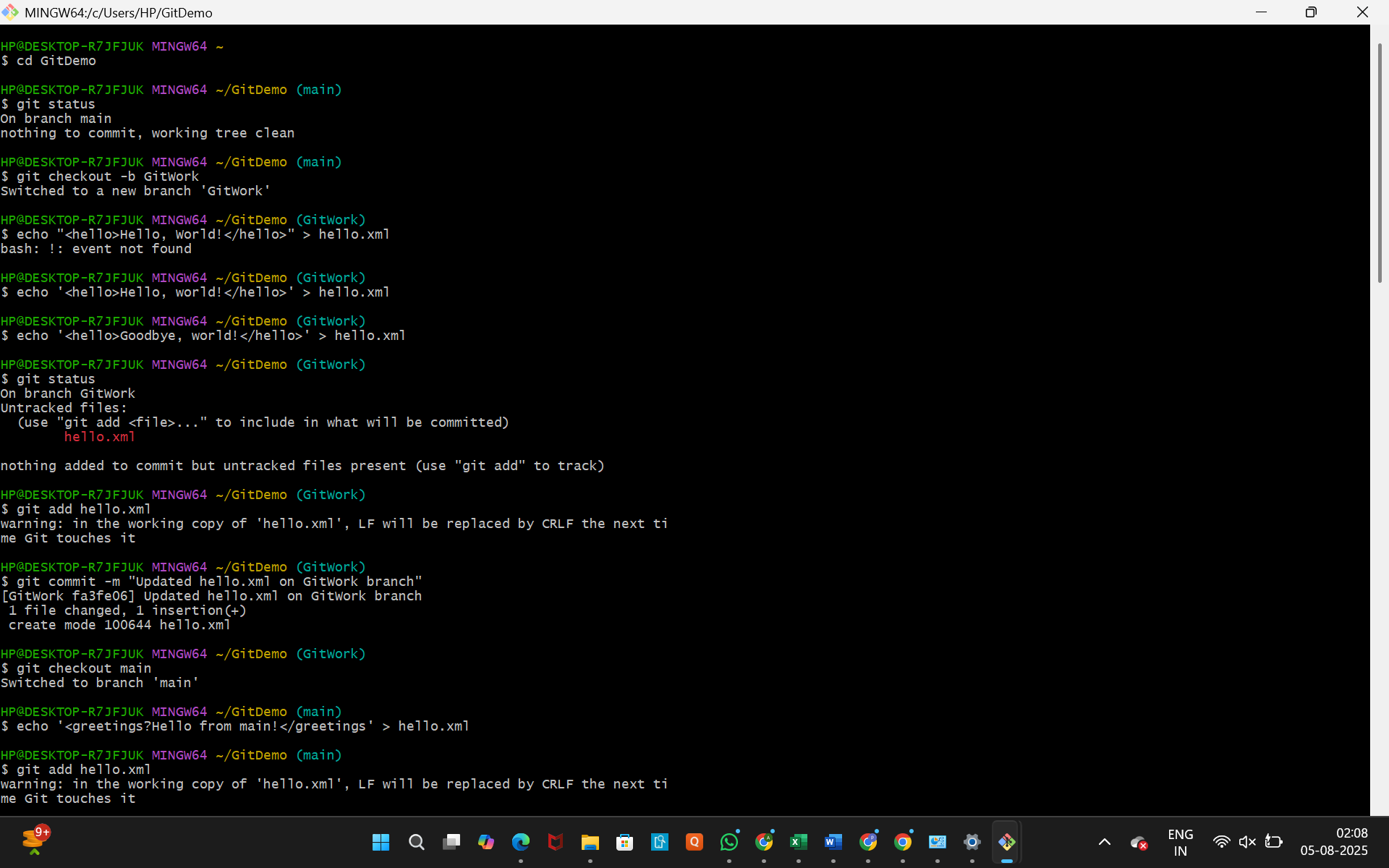


* + **Explanation:** This overwrites the file with different content. The git status command will show hello.xml as a "modified" file.

1. **Commit the changes to reflect in the branch.**
   * **Command:**

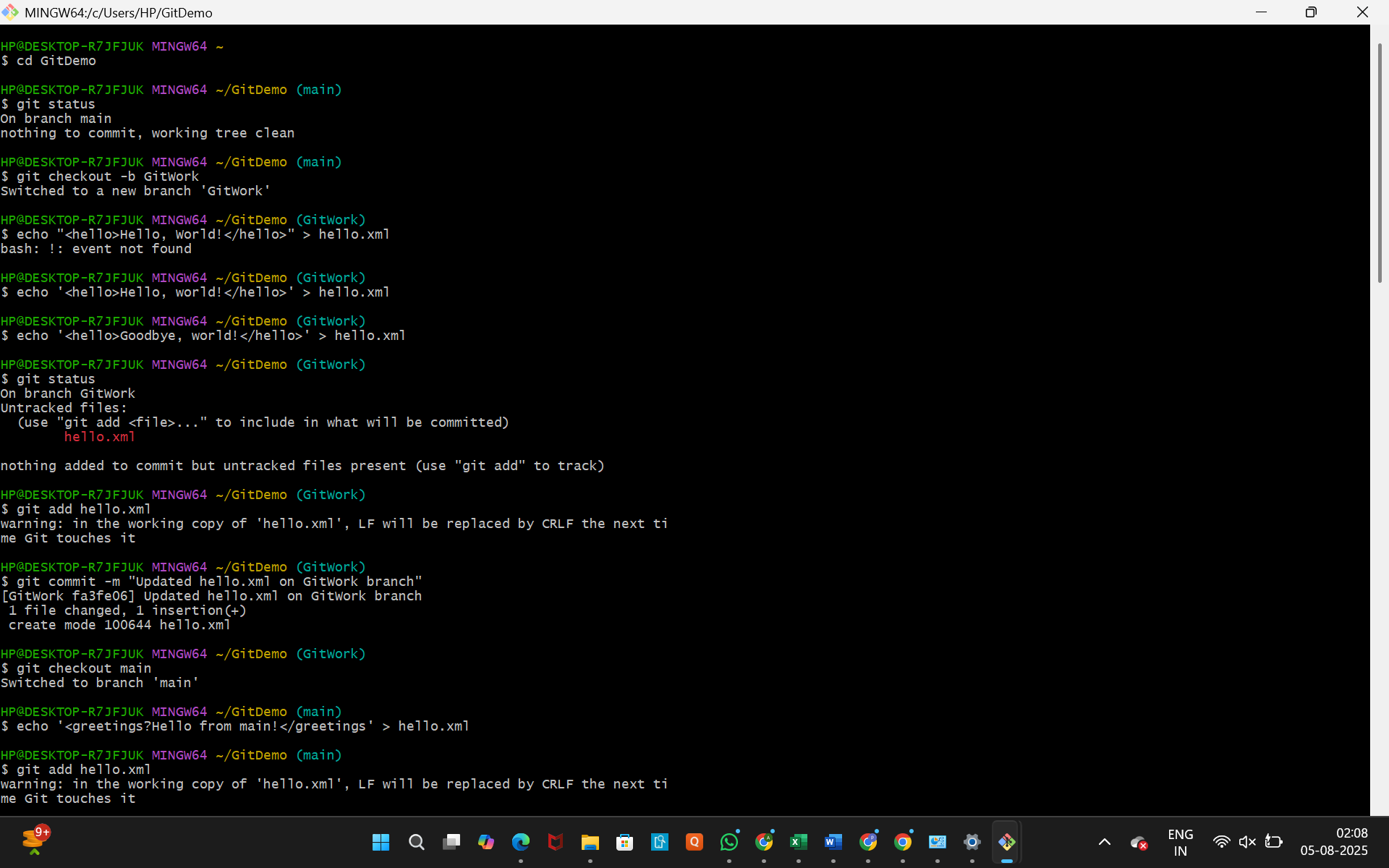


1. **Switch to master (your main branch).**
   * **Command:**



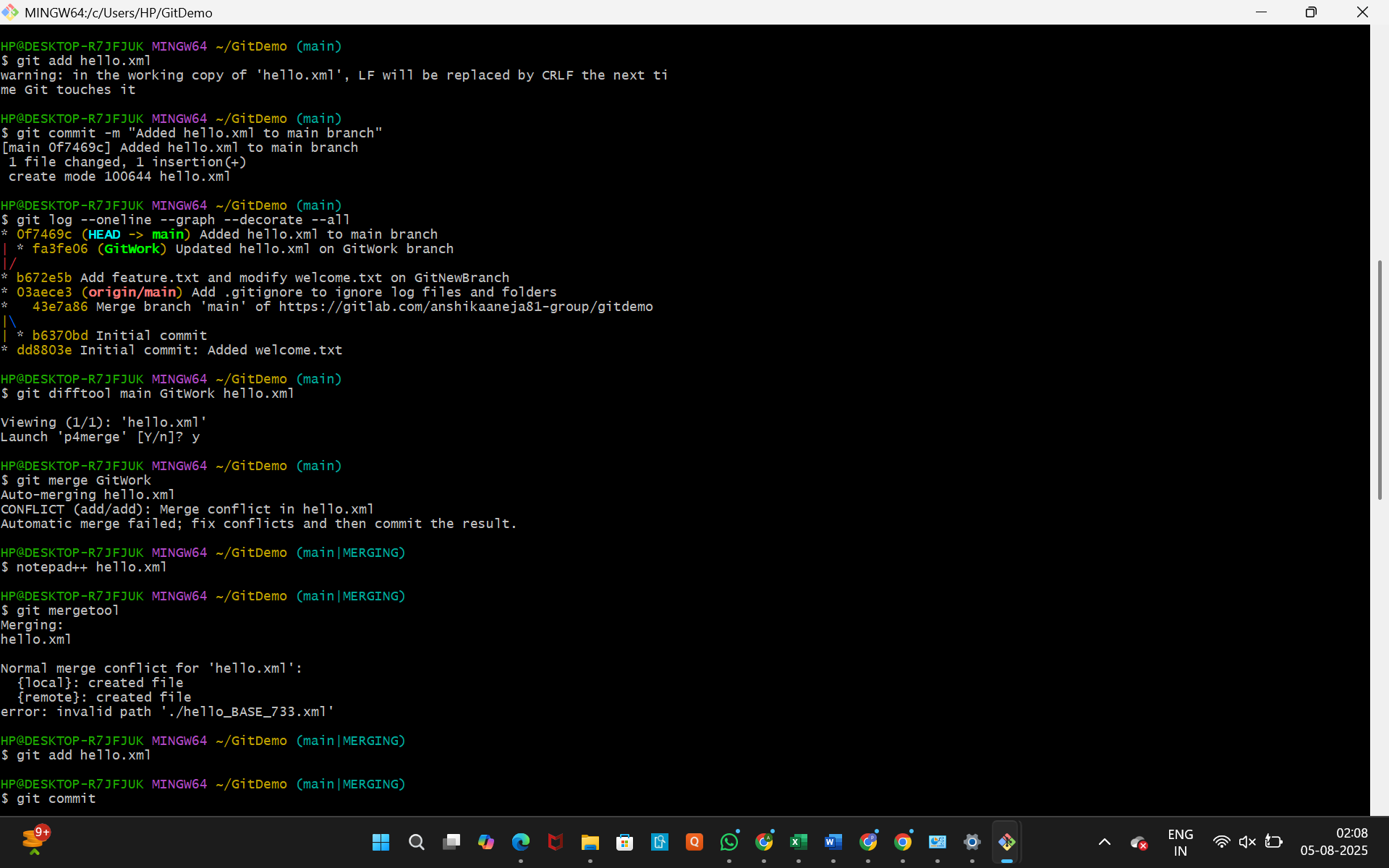
* + **What to expect:** The prompt will change to (main). If you type cat hello.xml, you will see an error because the file doesn't exist on this branch yet.

1. **Add a file "hello.xml" to main and add some different content than previous.**
   * **Command:**

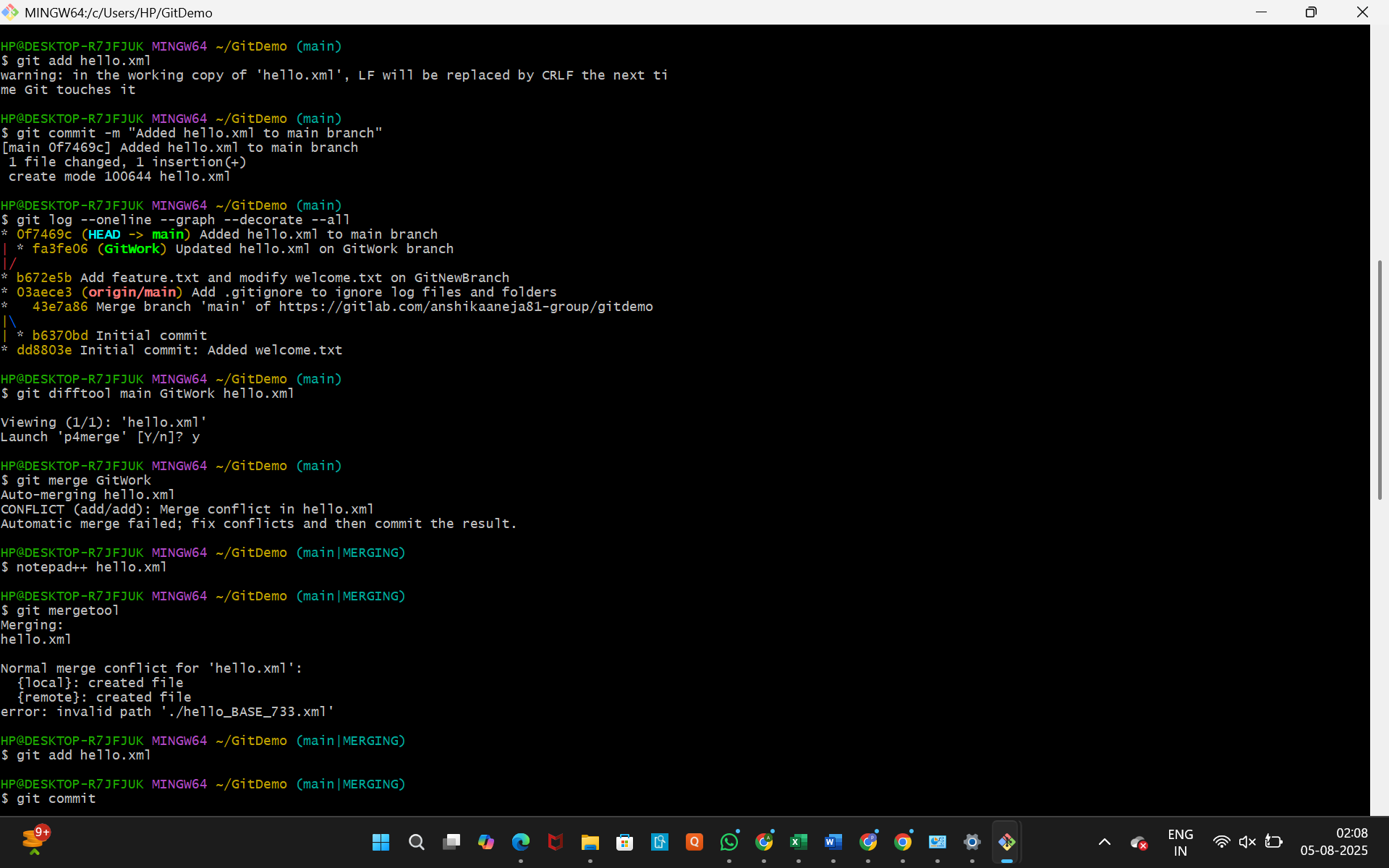


* + **Explanation:** This is the key step that sets up the merge conflict. Both branches now have a file with the same name, but with different content, and they were added independently.

1. **Commit the changes to the main branch.**
   * **Command:**

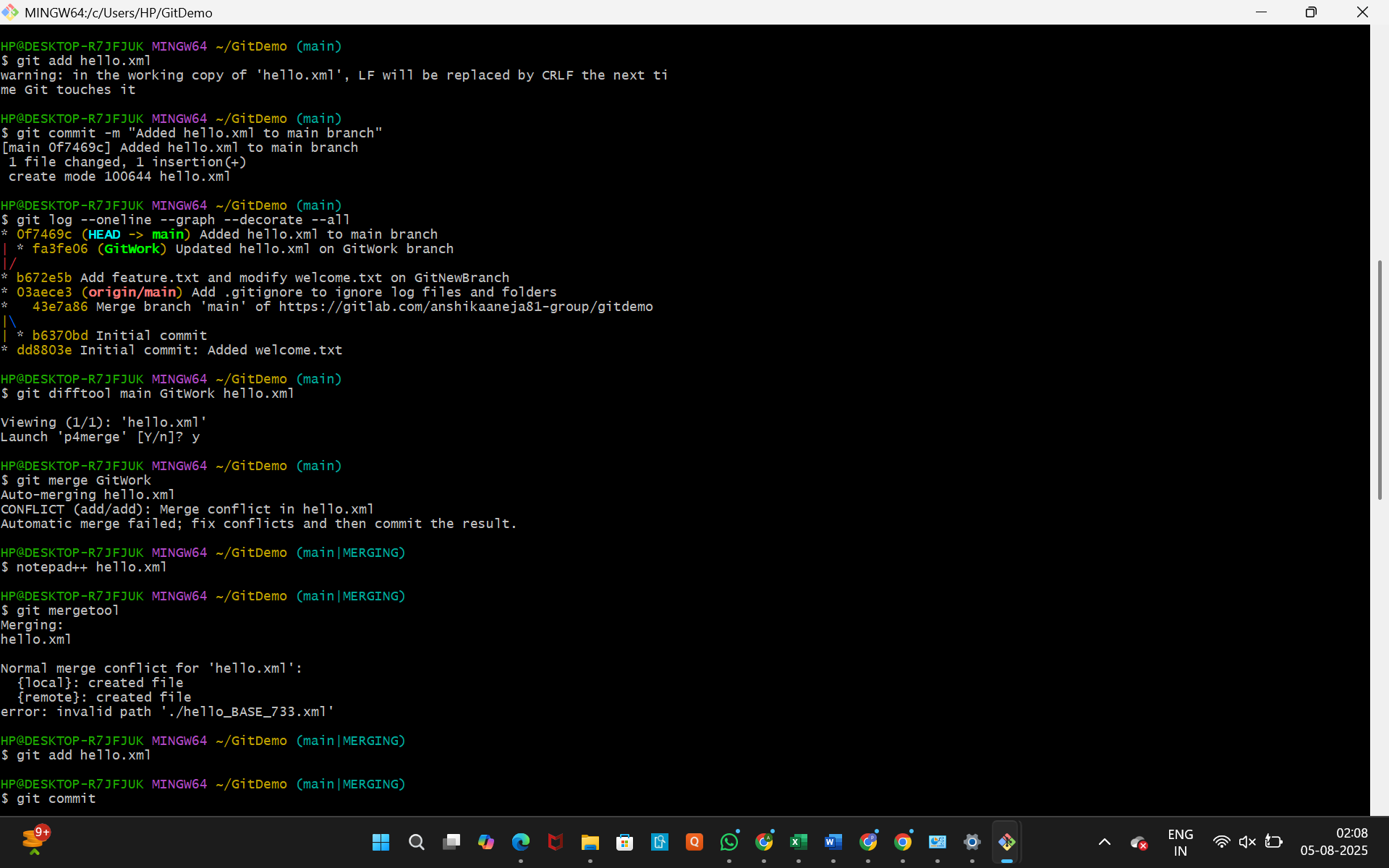


1. **Observe the log by executing git log --oneline --graph --decorate --all.**
   * **Command:**

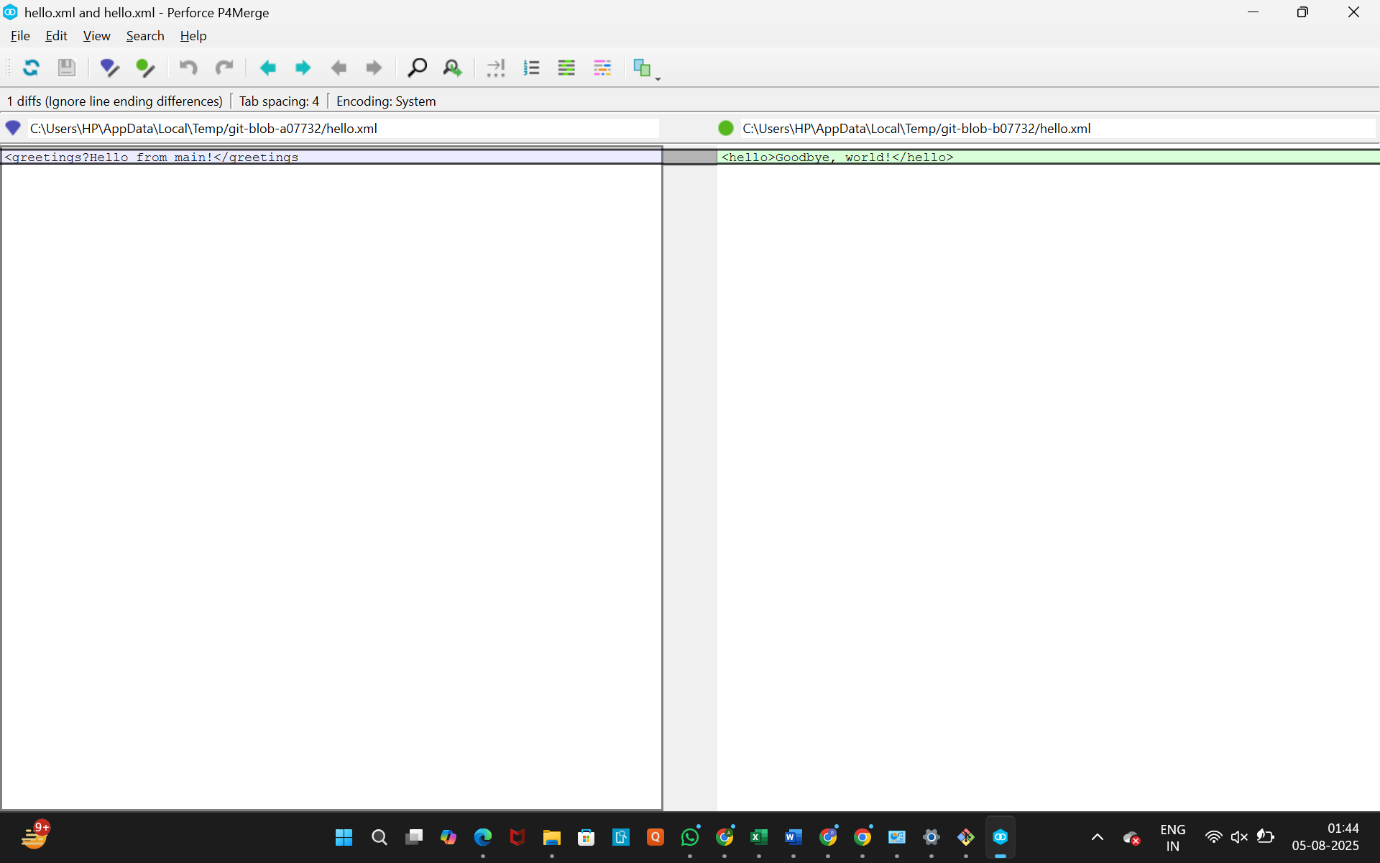


* + **Explanation:** This will show the commit history of both branches diverging from a common point. You'll see two separate lines of history.

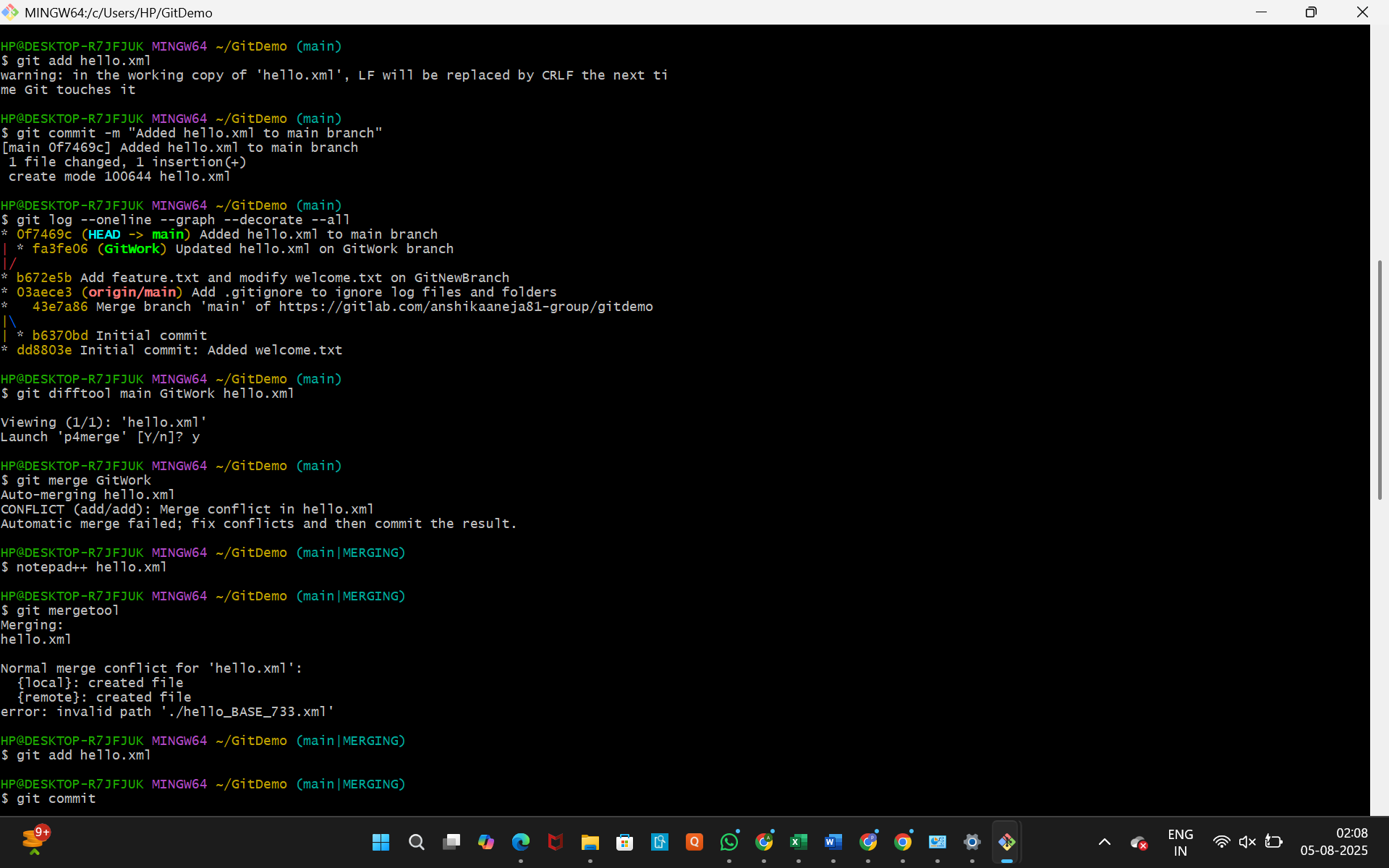
1. **Check the differences with Git diff tool.**
   * **Command:**



* + **Explanation:** This will launch P4Merge to show you the differences in the hello.xml file between the two branches, visually confirming that they have different content. Close P4Merge when you are done.

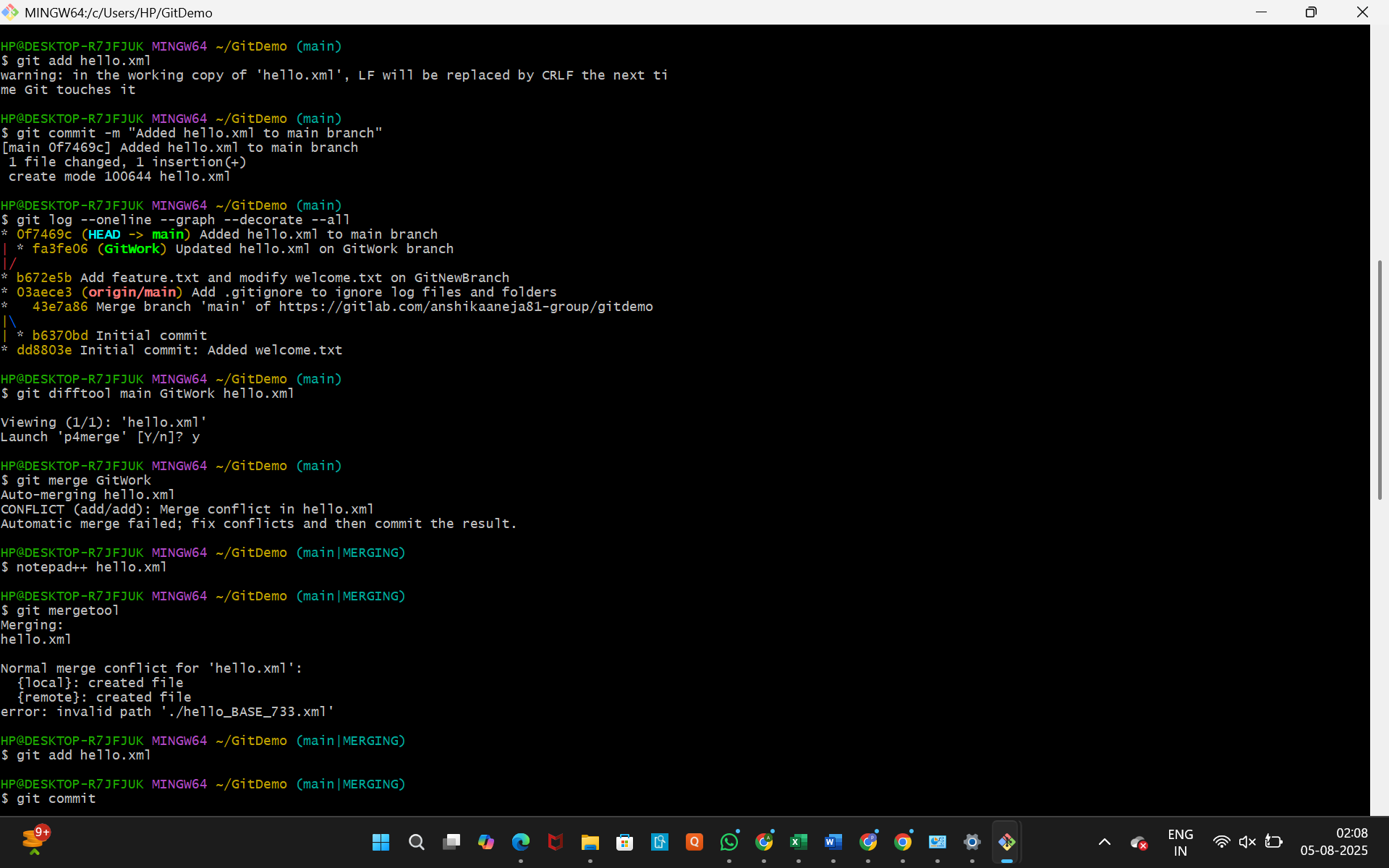


1. **(This step is a duplicate of the previous, we've already done it).**
2. **Merge the branch to the master.**
   * **Command:**

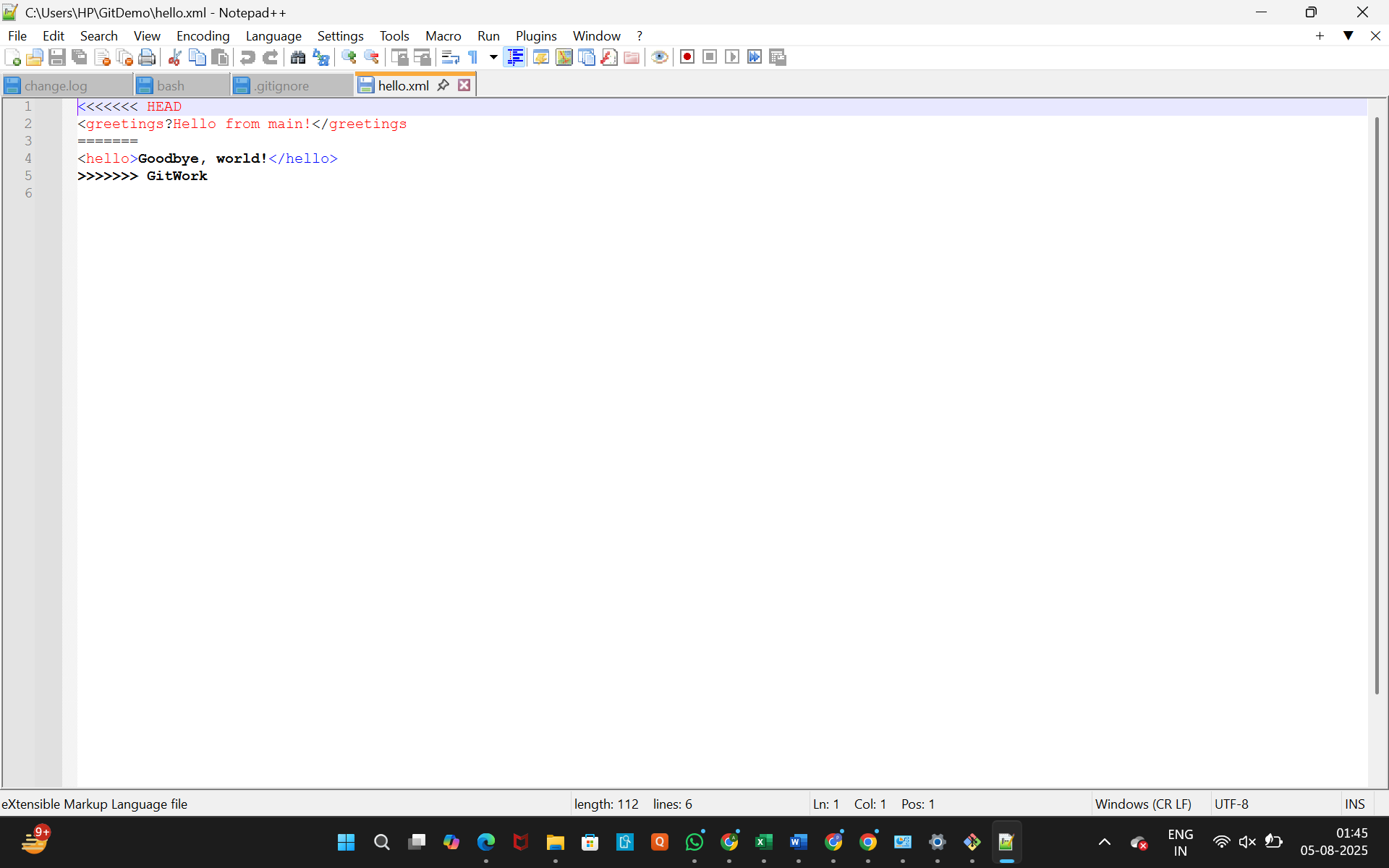


* + **What to expect:** This is where the conflict will occur. Git will halt the process and give you a message like CONFLICT (add/add): Merge conflict in hello.xml and Automatic merge failed; fix conflicts and then commit the result..

1. **Observe the git mark up.**
   * **Action:**
     + Open the conflicted file in Notepad++:

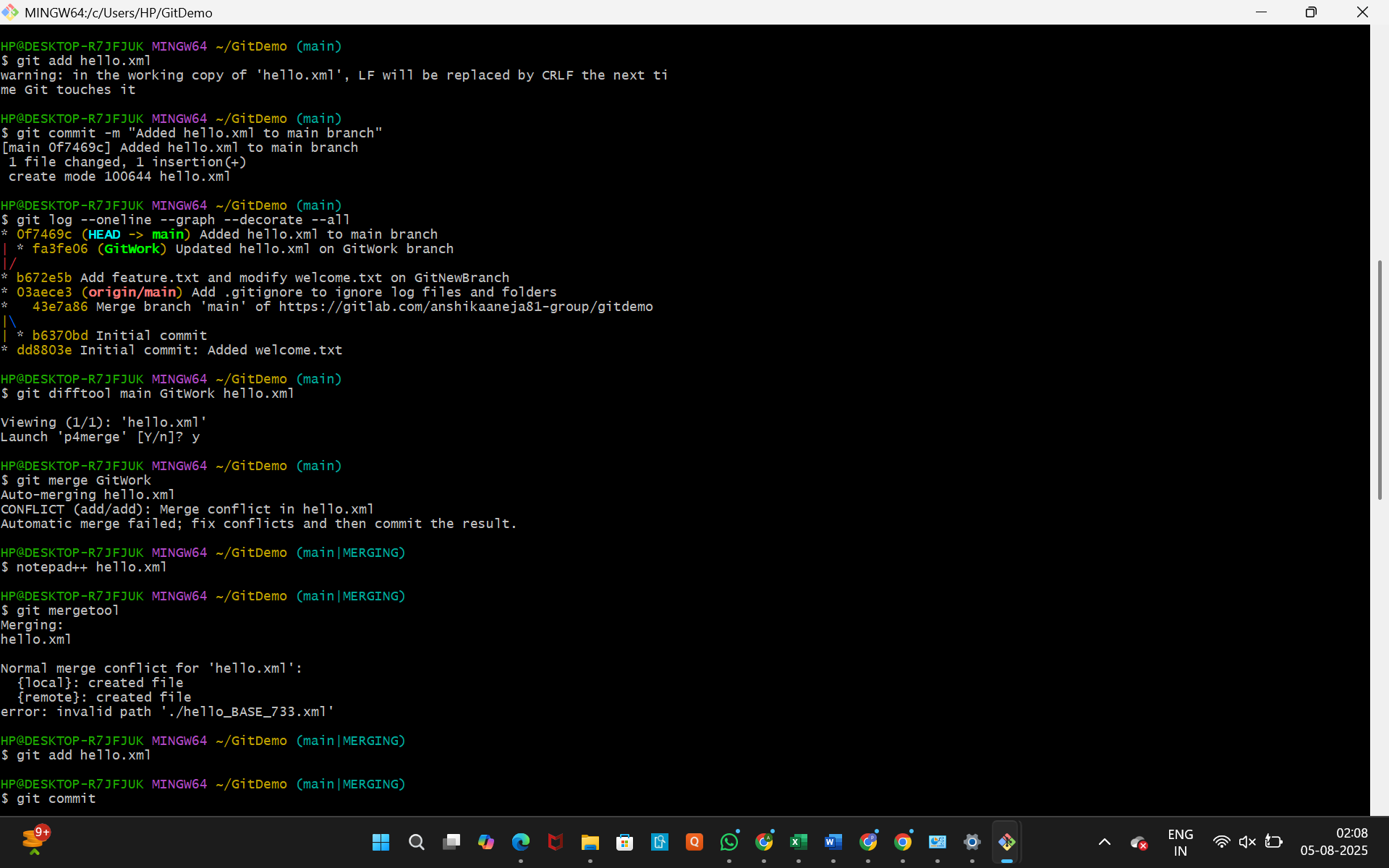


* + **What to expect:** You will see the conflict markers that Git adds to the file:
  + <<<<<<< HEAD
  + <greetings>Hello from main!</greetings>
  + =======
  + <hello>Goodbye, World!</hello>
  + >>>>>>> GitWork

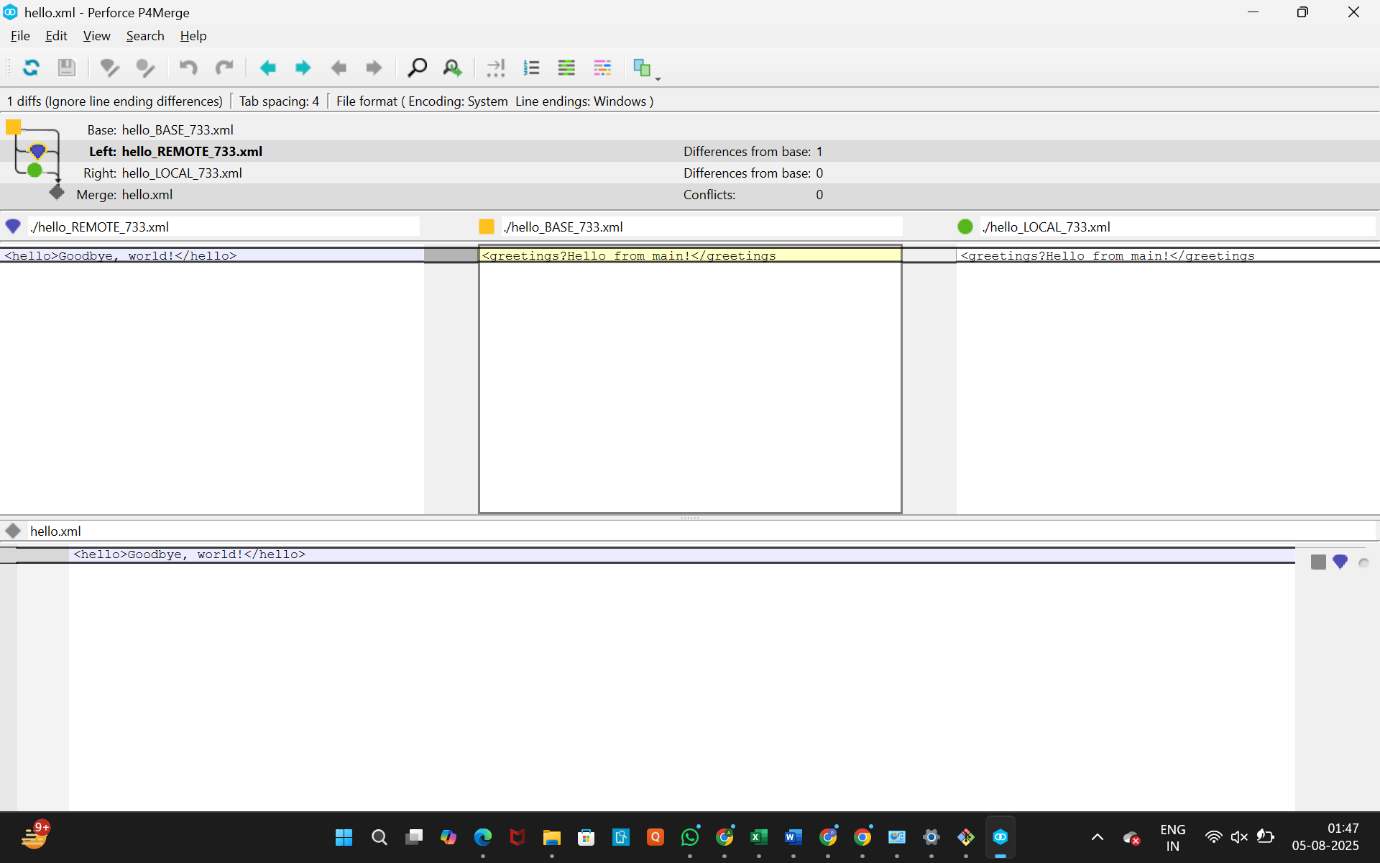


* + **Explanation:** HEAD is your current branch (main), and GitWork is the branch you are trying to merge. The lines between <<<<<<< HEAD and ======= are the changes from main, and the lines between ======= and >>>>>>> GitWork are the changes from GitWork.

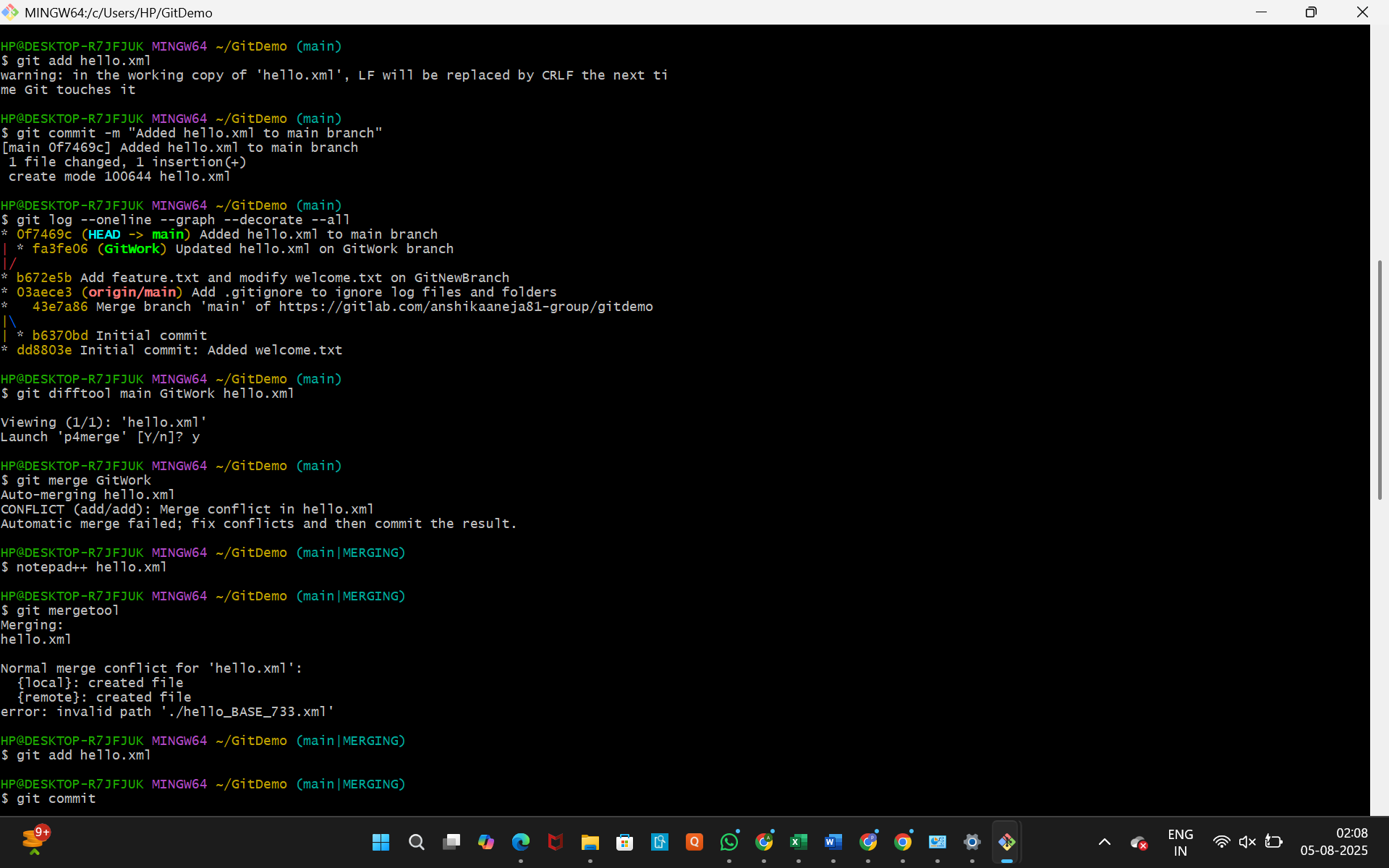
1. **Use 3-way merge tool to resolve the conflict.**
   * **Command:**

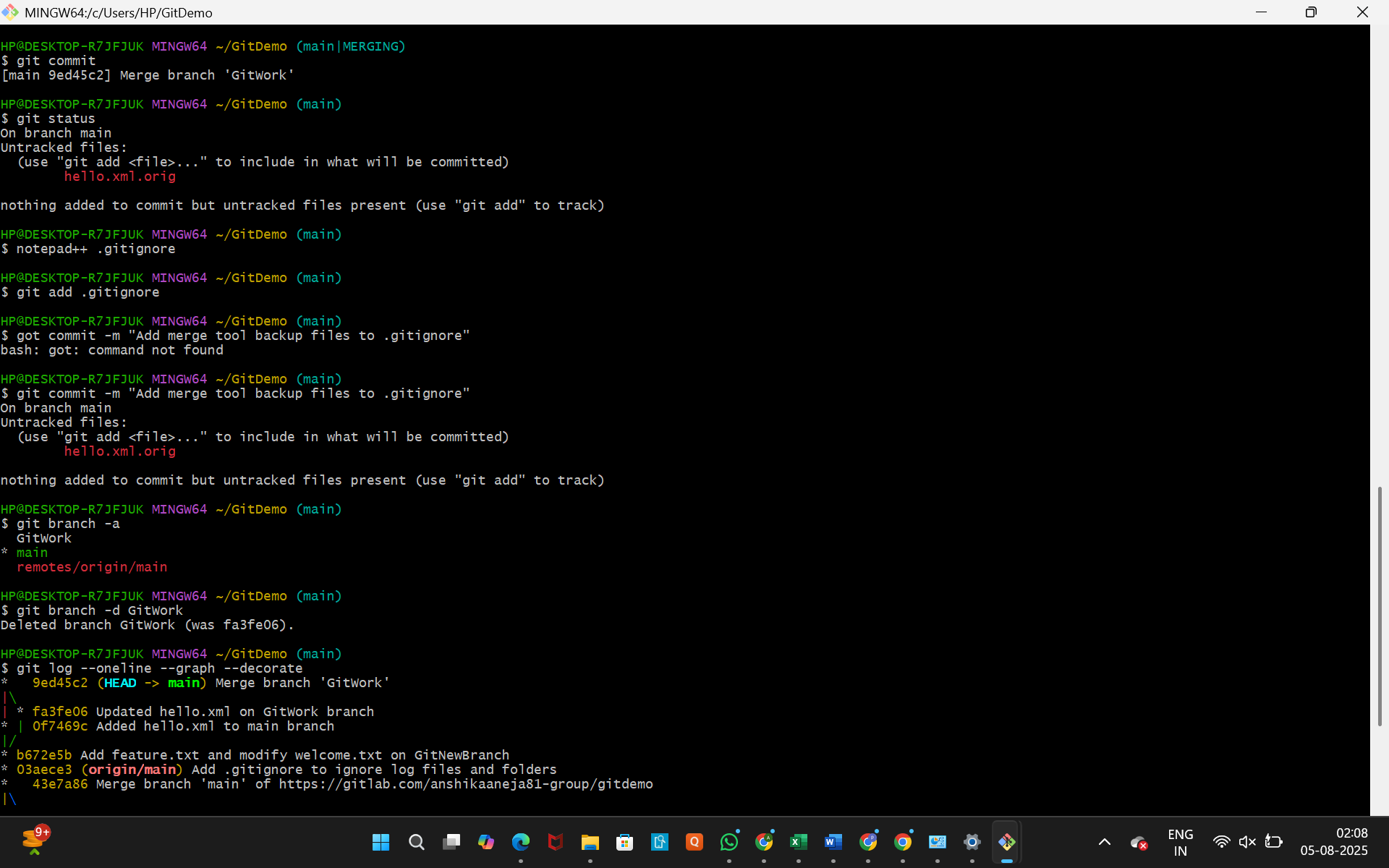


* + **What to expect:** P4Merge will open in a 3-way view. You will see:
    - Left pane: The file from the main branch.
    - Right pane: The file from the GitWork branch.
    - Bottom pane: The final, merged file.
  + **Action:** Manually edit the content in the bottom pane to what you want the final version to be (e.g., <hello>Hello, World!</hello> <greetings>Hello from main!</greetings>). Save the file and close P4Merge.

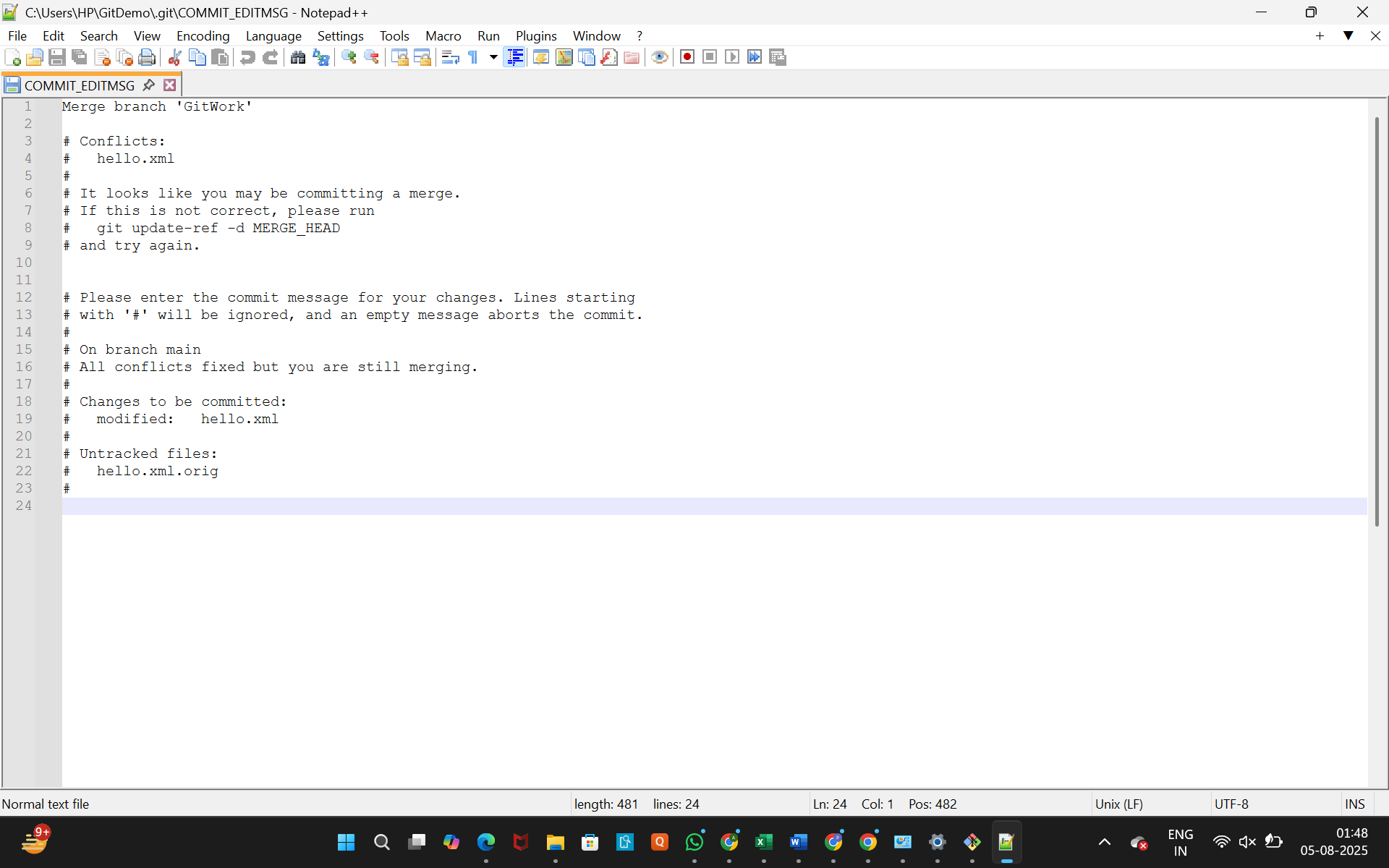


1. **Commit the changes to the master, once done with conflict.**
   * **Command:**

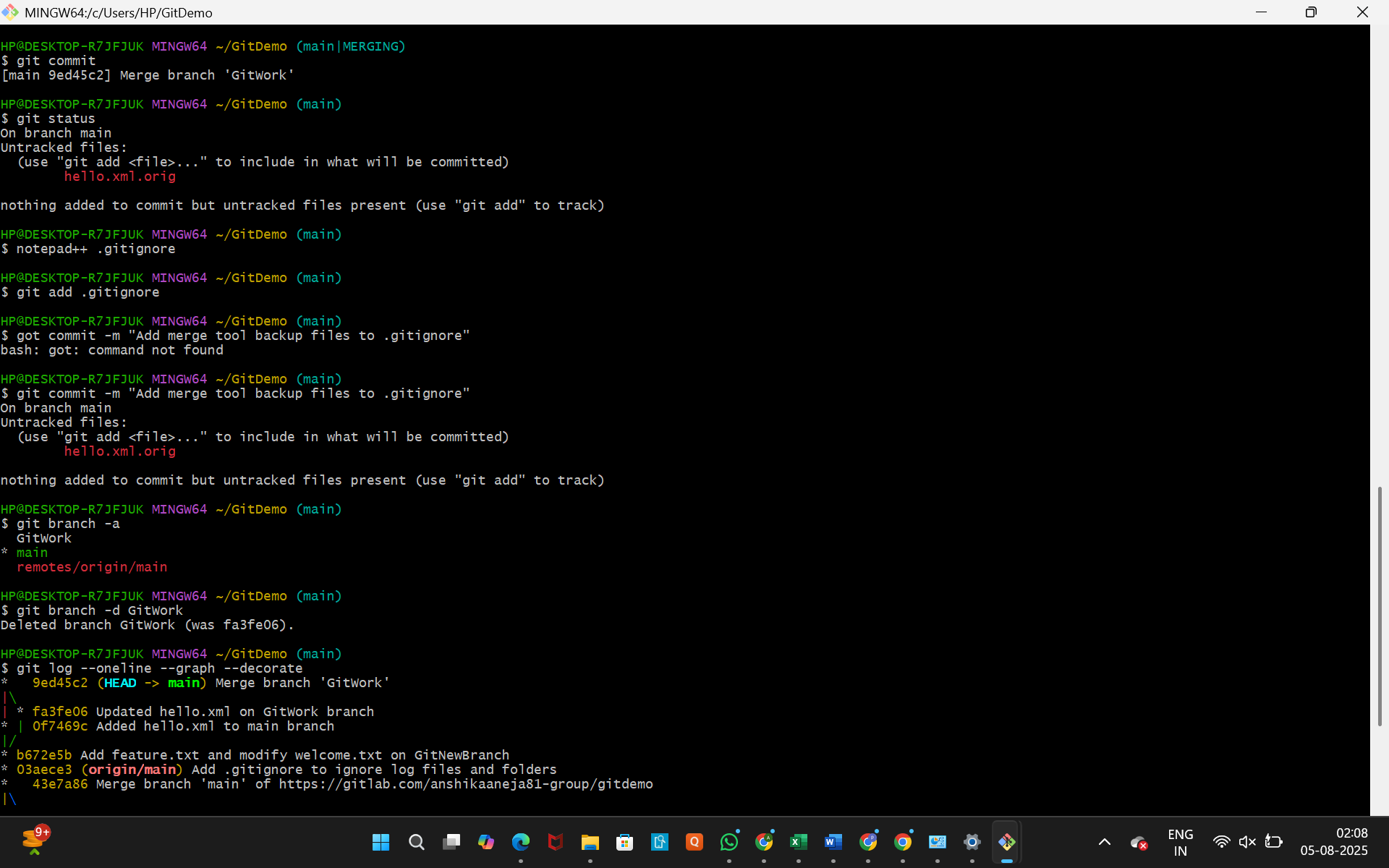




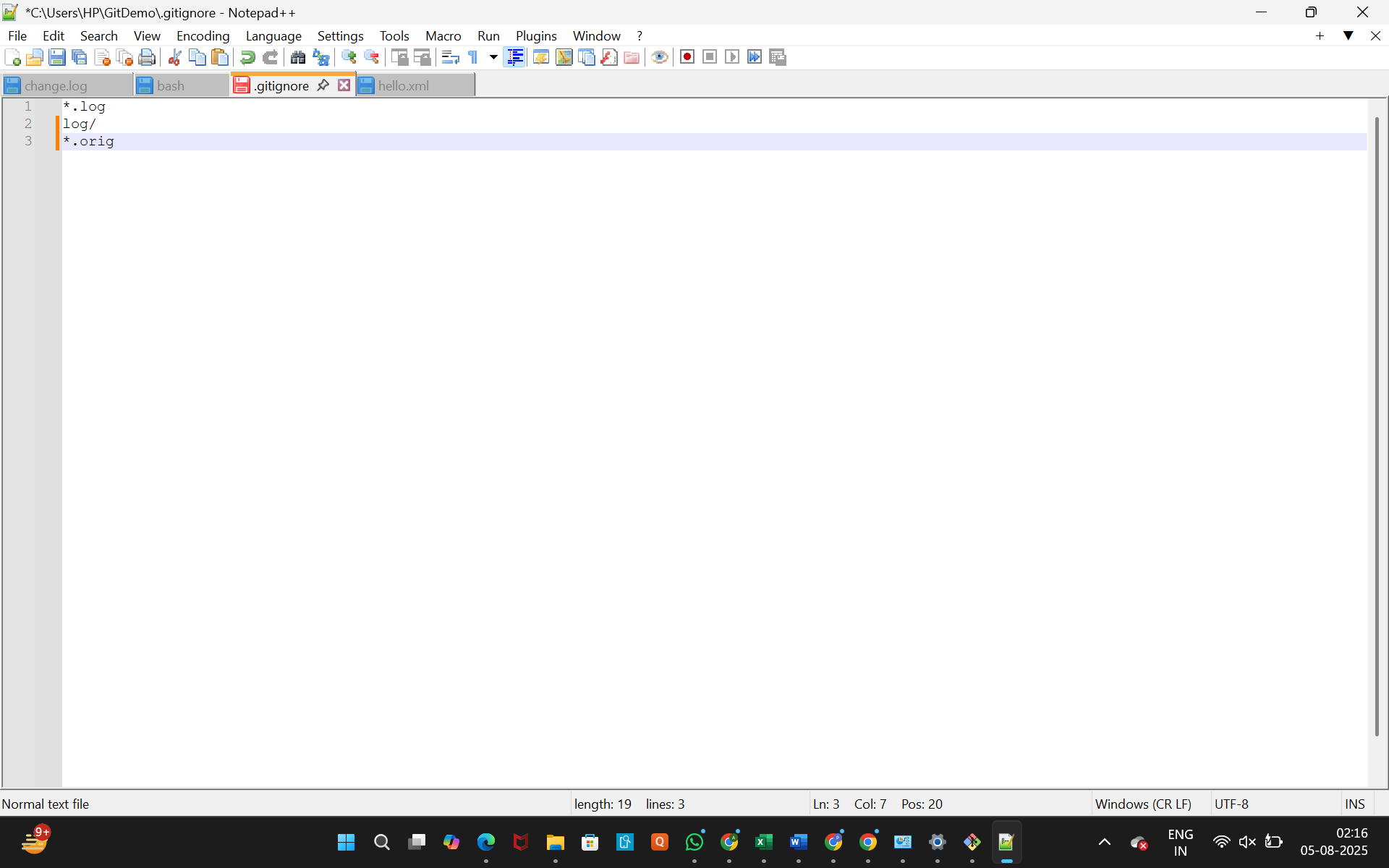
* + **Explanation:** git mergetool automatically resolves the conflict markers. The git add stages the now-resolved file. The git commit command will open your editor with a pre-filled merge commit message. Save and close the editor to complete the merge.



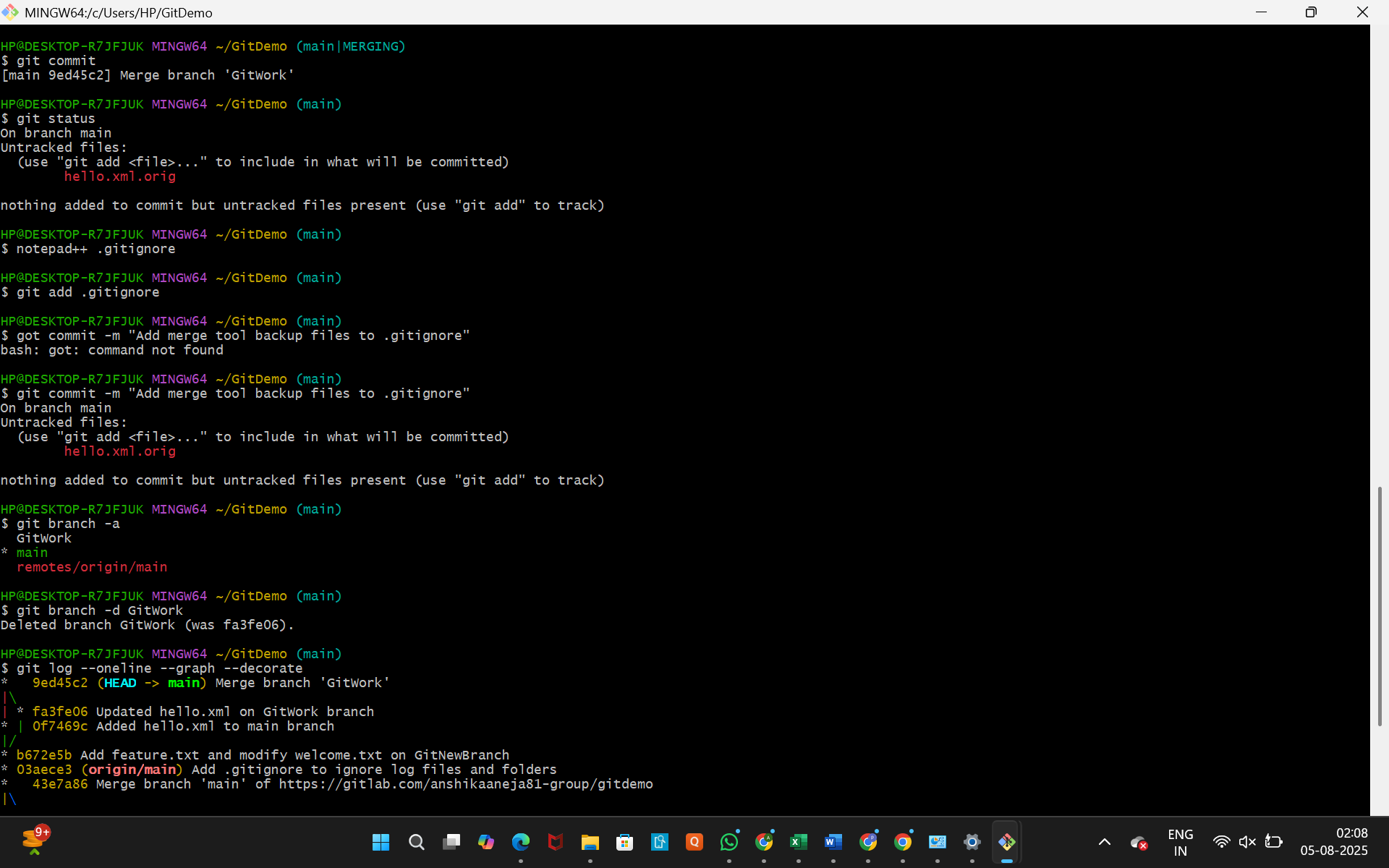
1. **Observe the git status and add backup file to the .gitignore file.**
   * **Command:**

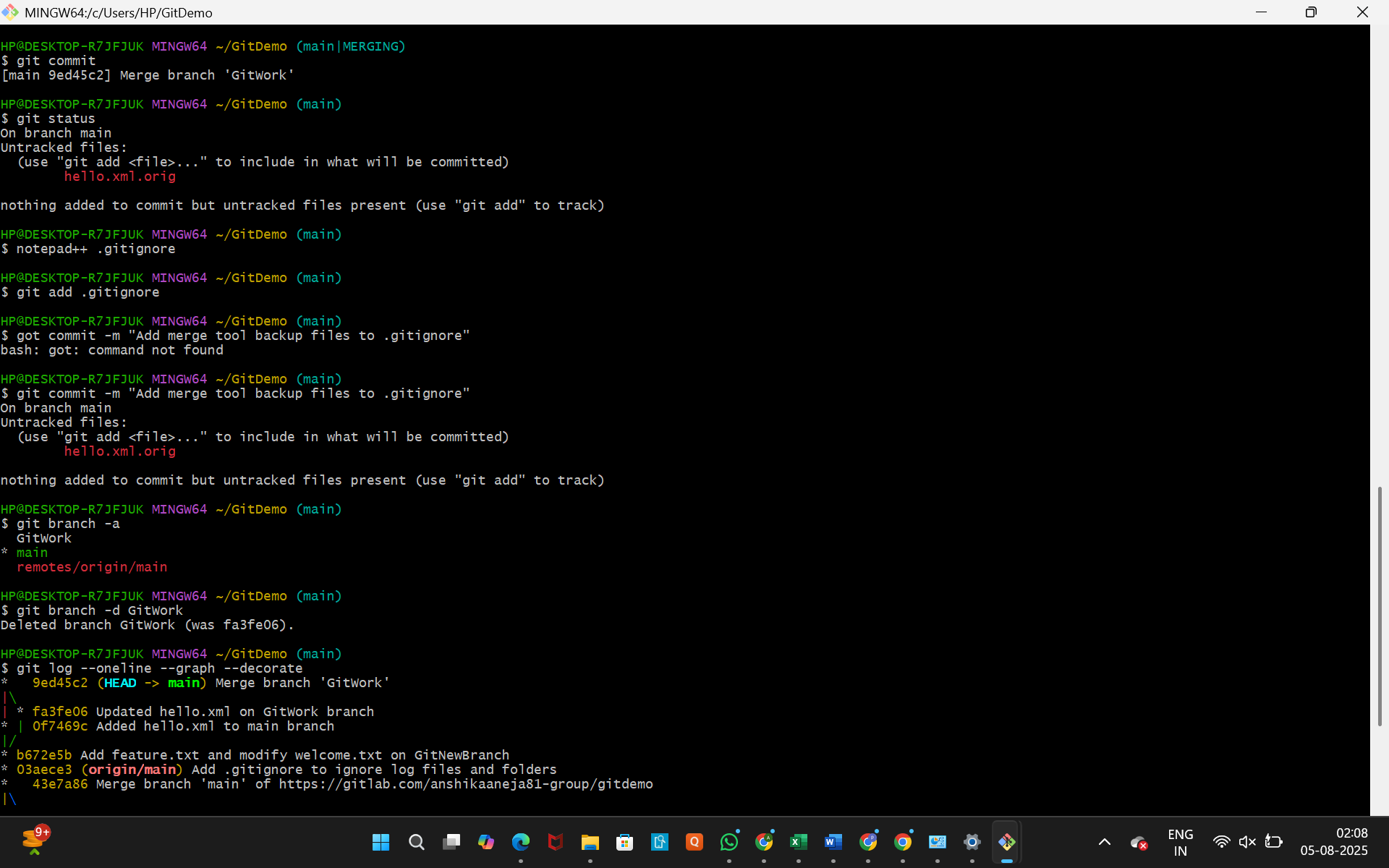


* + **Action:** The git status might show a .orig or .bak file created by the merge tool. In the .gitignore file, add a rule to ignore these files (e.g., \*.orig). Save and close Notepad++.

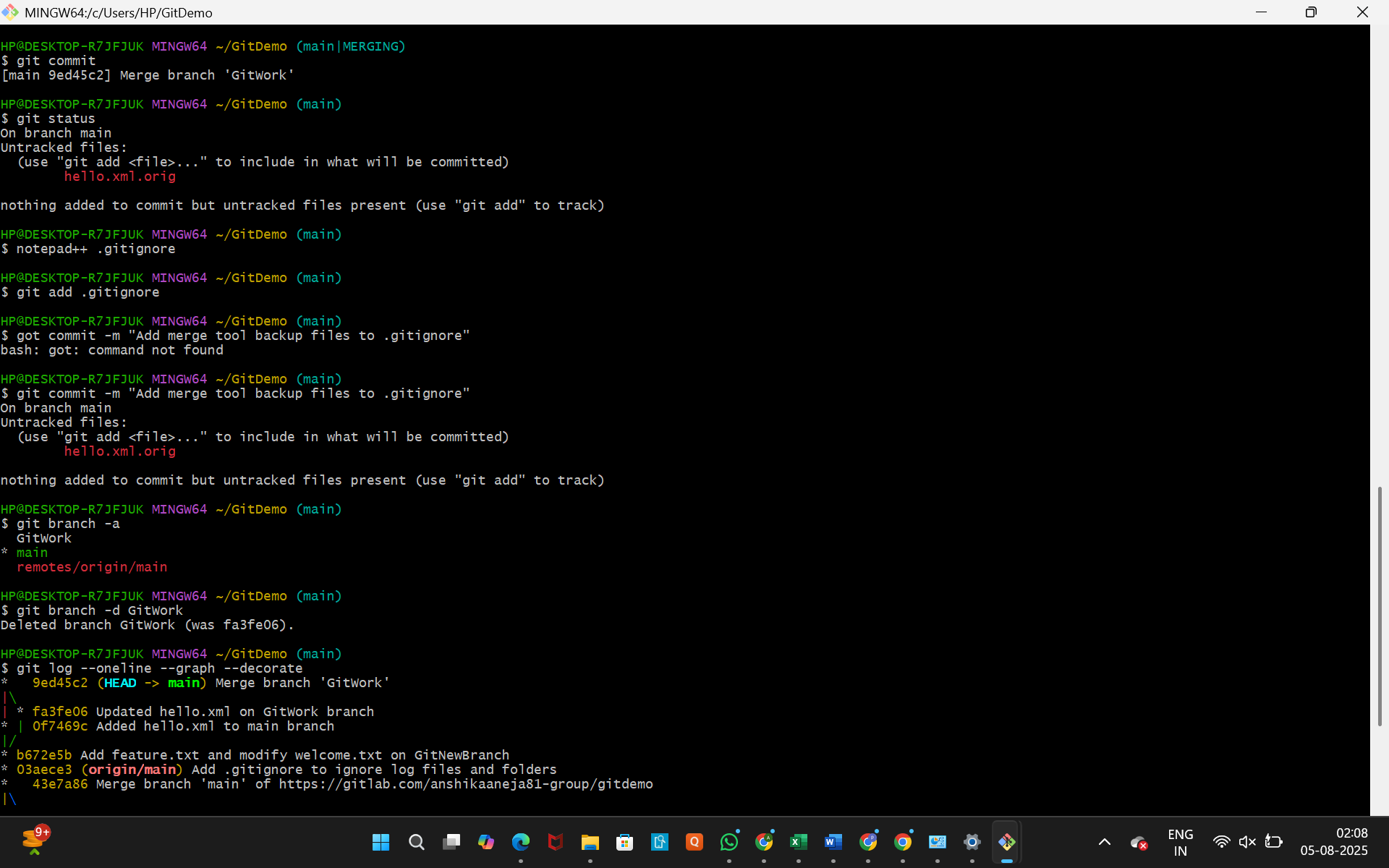


1. **Commit the changes to the .gitignore file.**
   * **Command:**

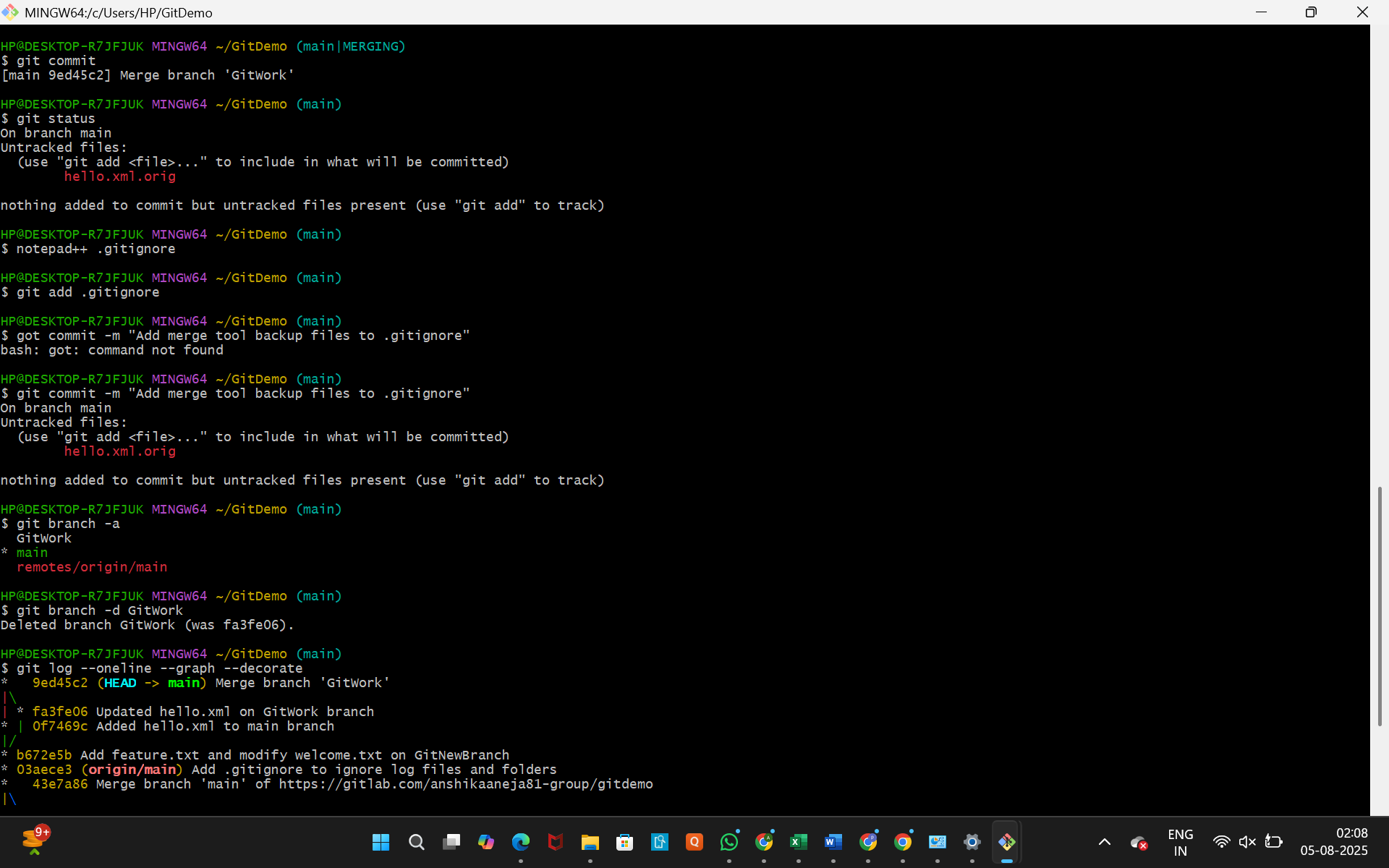




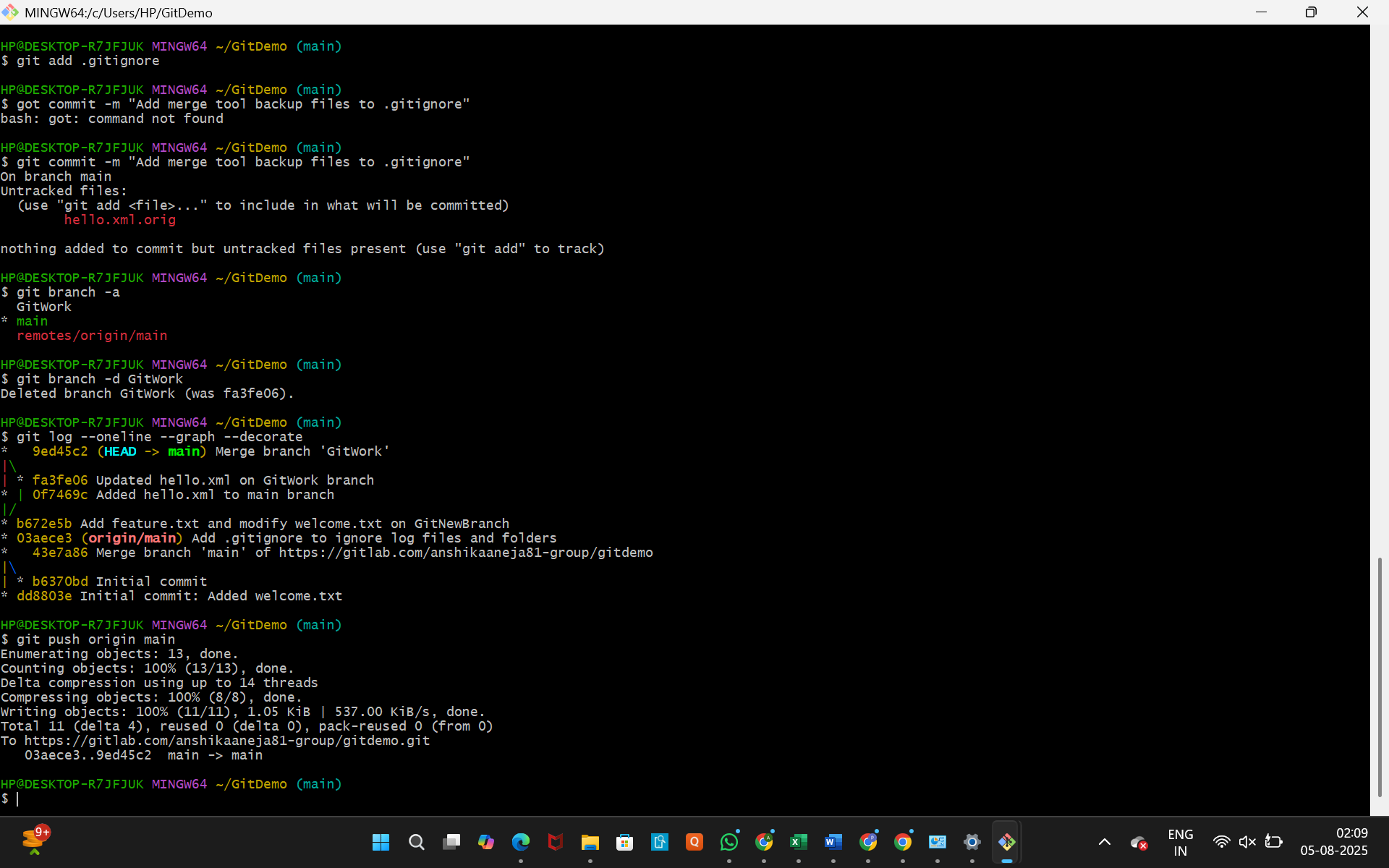
1. **List out all the available branches.**
   * **Command:**



1. **Delete the branch, which merged to master.**
   * **Command:**



1. **Observe the log by executing git log --oneline --graph --decorate.**
   * **Command:**



* + **Explanation:** This will show the final commit history, with a clear merge commit that brings the GitWork branch history into main.