**3. Git – HOL**

**Objectives:**

* Explain branching and merging.
* Explain about creating a branch request in GitLab.
* Explain about creating a merge request in GitLab.

**In this hands-on lab, you will learn how to:**

* Construct a branch.
* Make some changes in the branch.
* Merge it with your main (or master/trunk) branch.

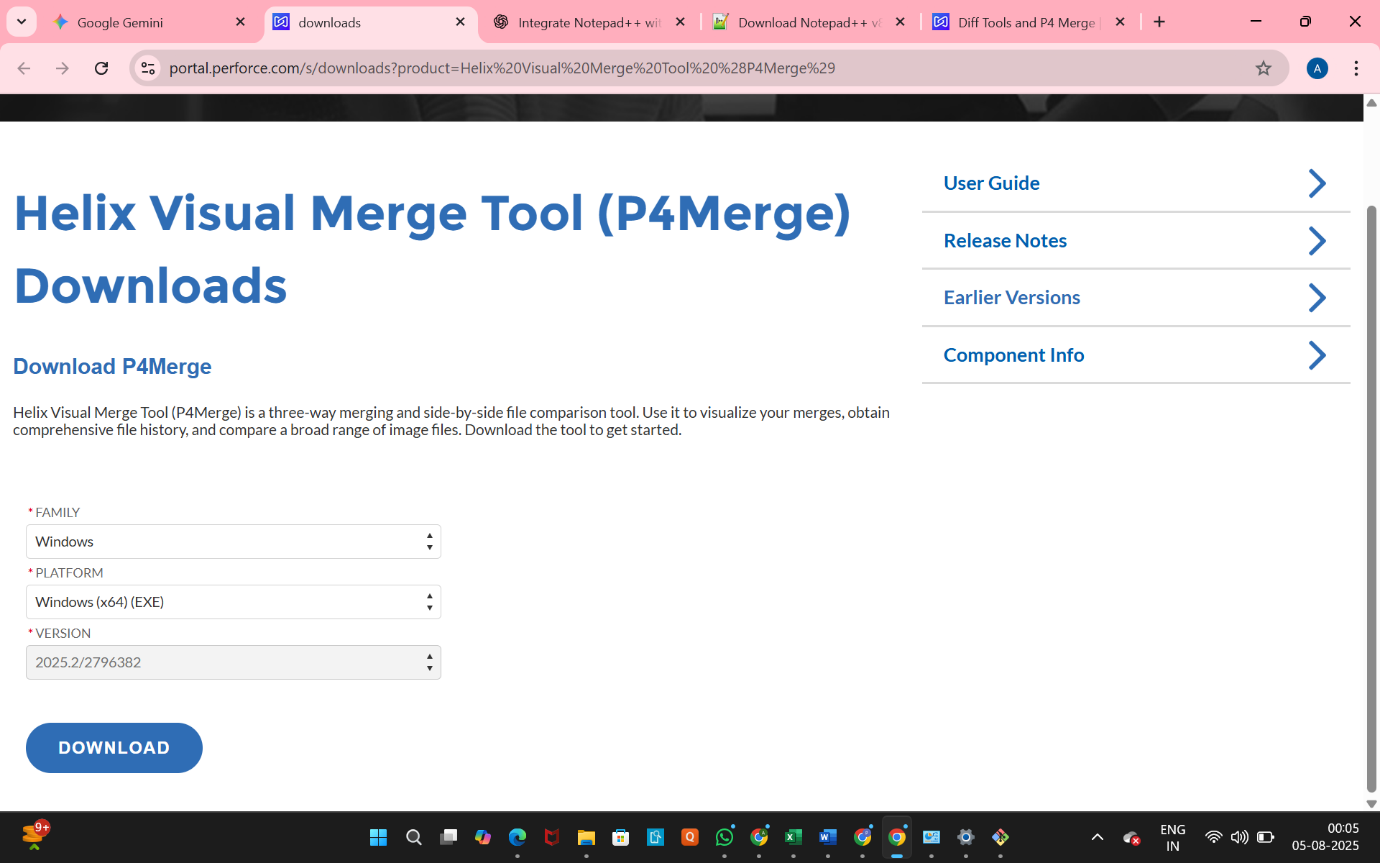
**Prerequisites:**

* Setting up Git environment with P4Merge tool for Windows: This is important for a specific step. If you haven't installed P4Merge and configured it with Git yet, you'll need to do so. I'll provide the configuration steps below.
* A free account in GitHub: As mentioned in previous labs, this is a general prerequisite for remote interactions, though this lab focuses on local branching/merging first.

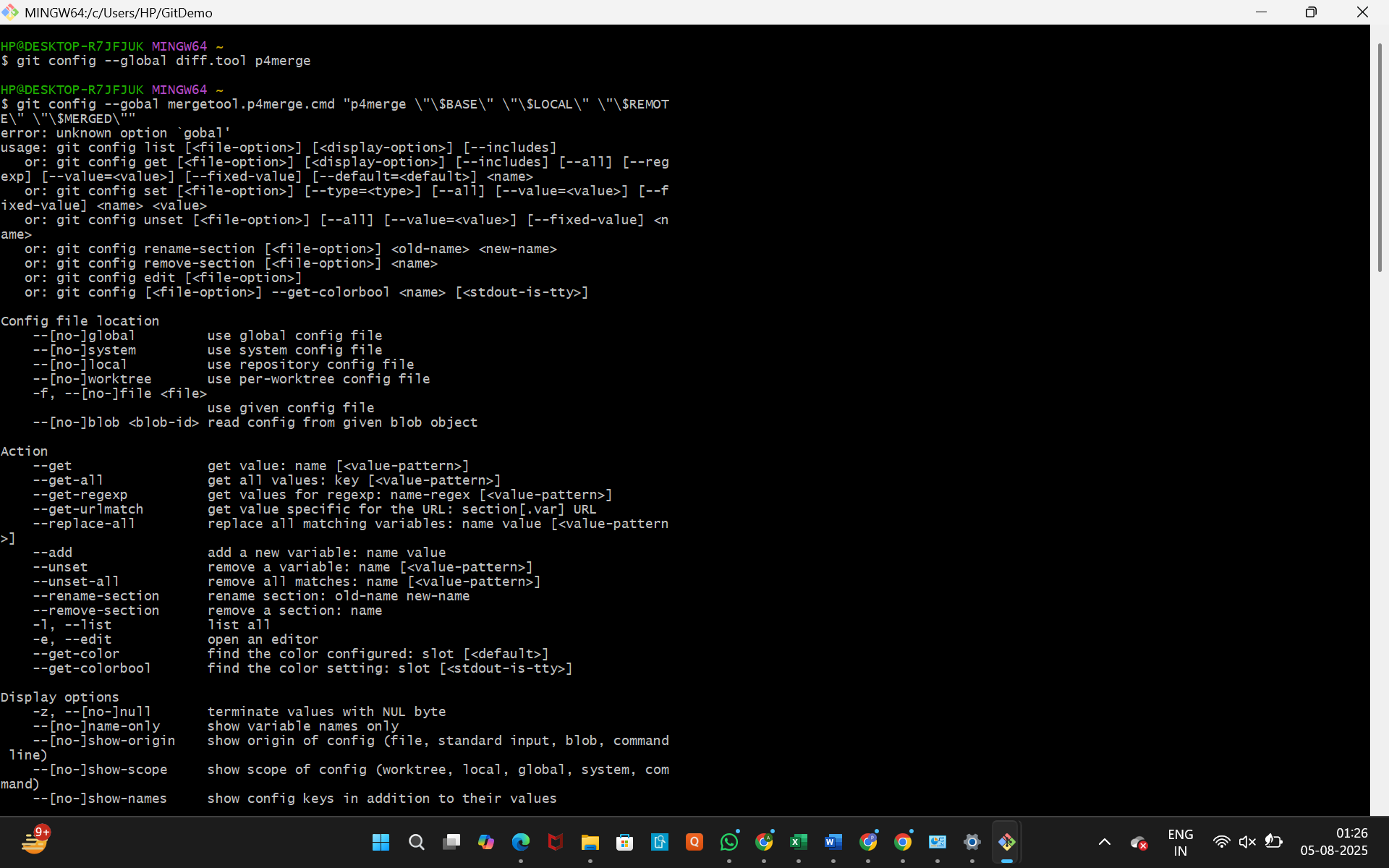
**Setup for P4Merge (If not already done):**

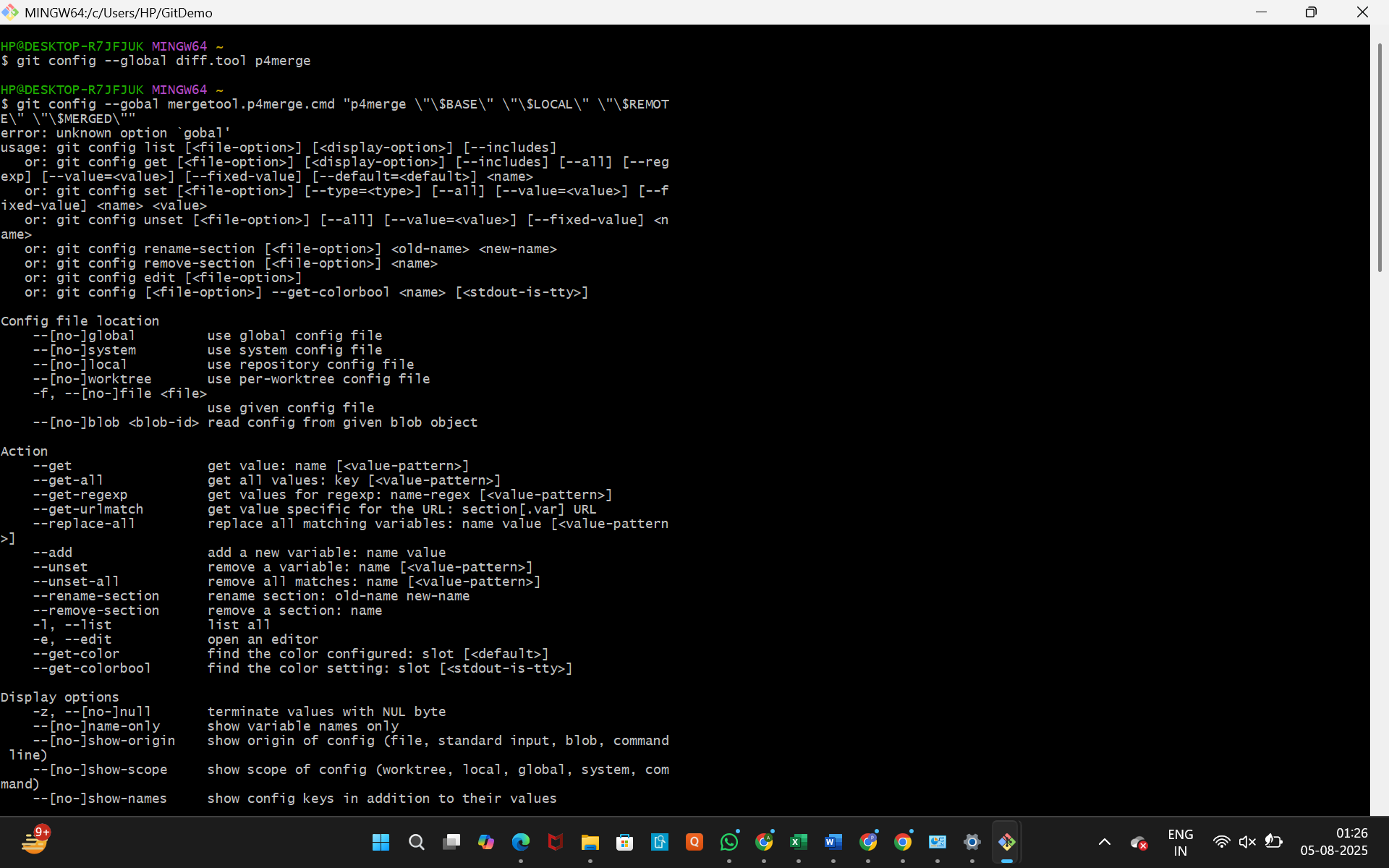
If P4Merge is not yet configured as your Git diff and merge tool, you need to do this first.

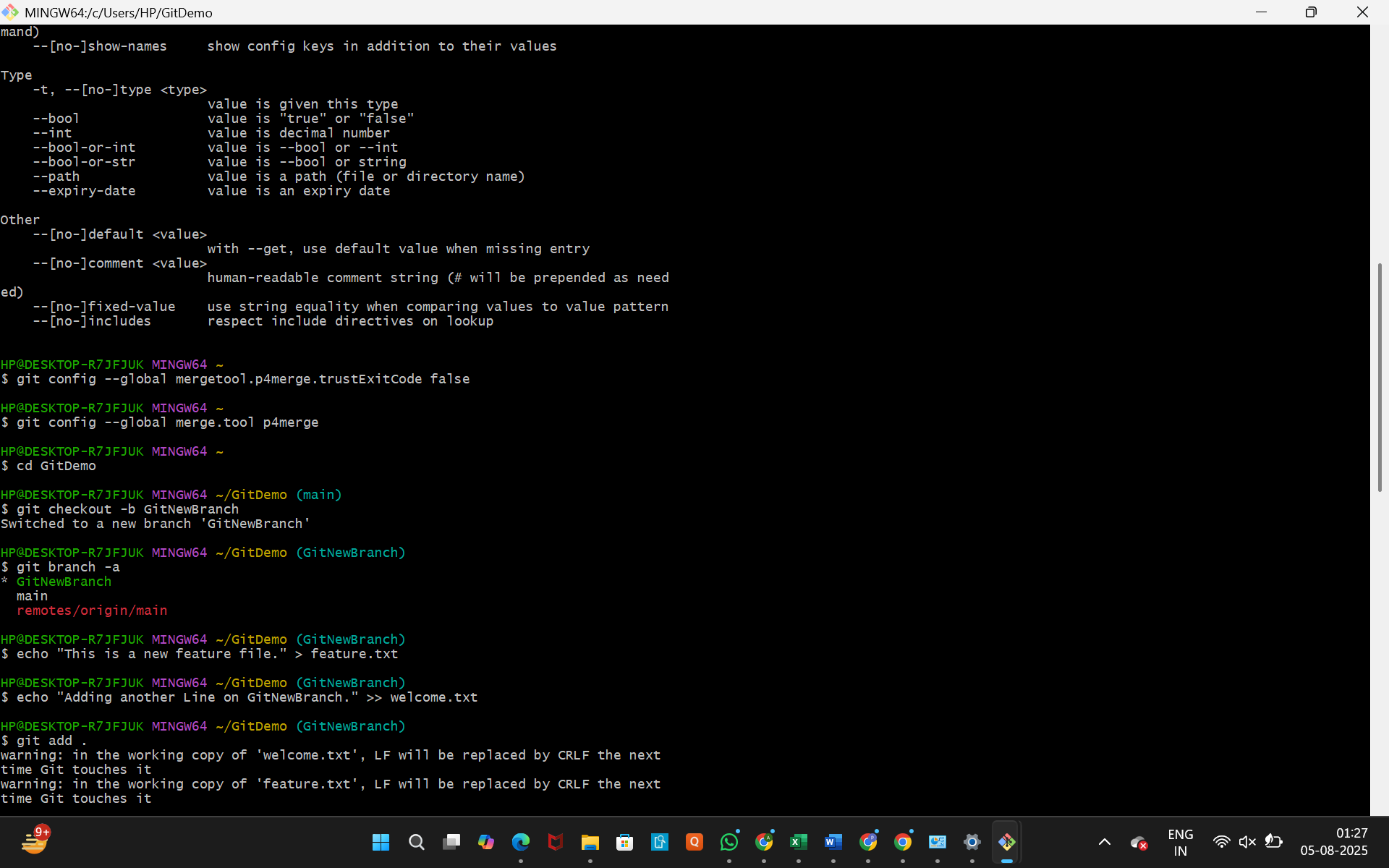
1. **Download and Install P4Merge:**
   * Go to the Perforce website (search for "P4Merge download").
   * Download and install P4Merge for Windows. Follow the installer prompts.

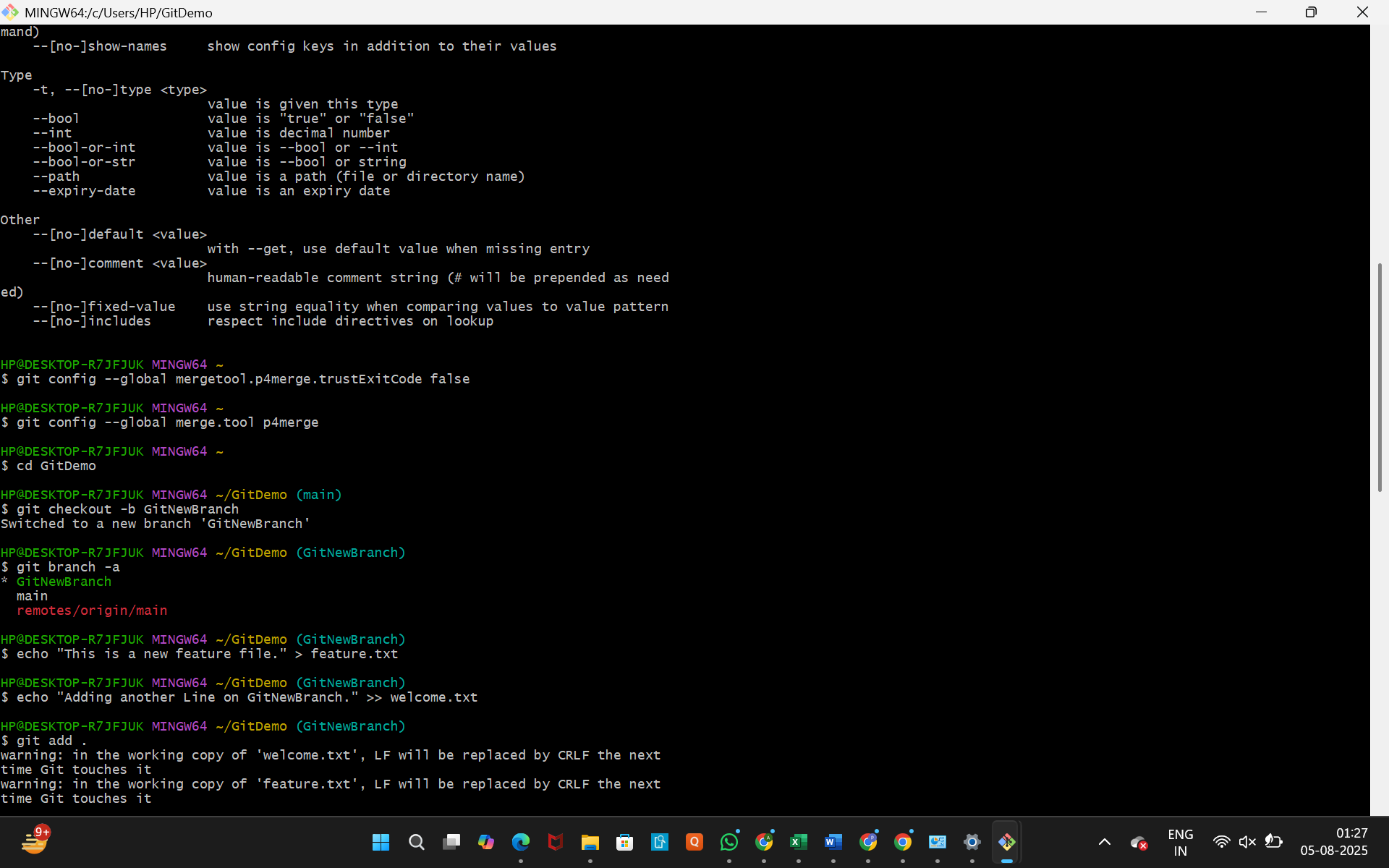


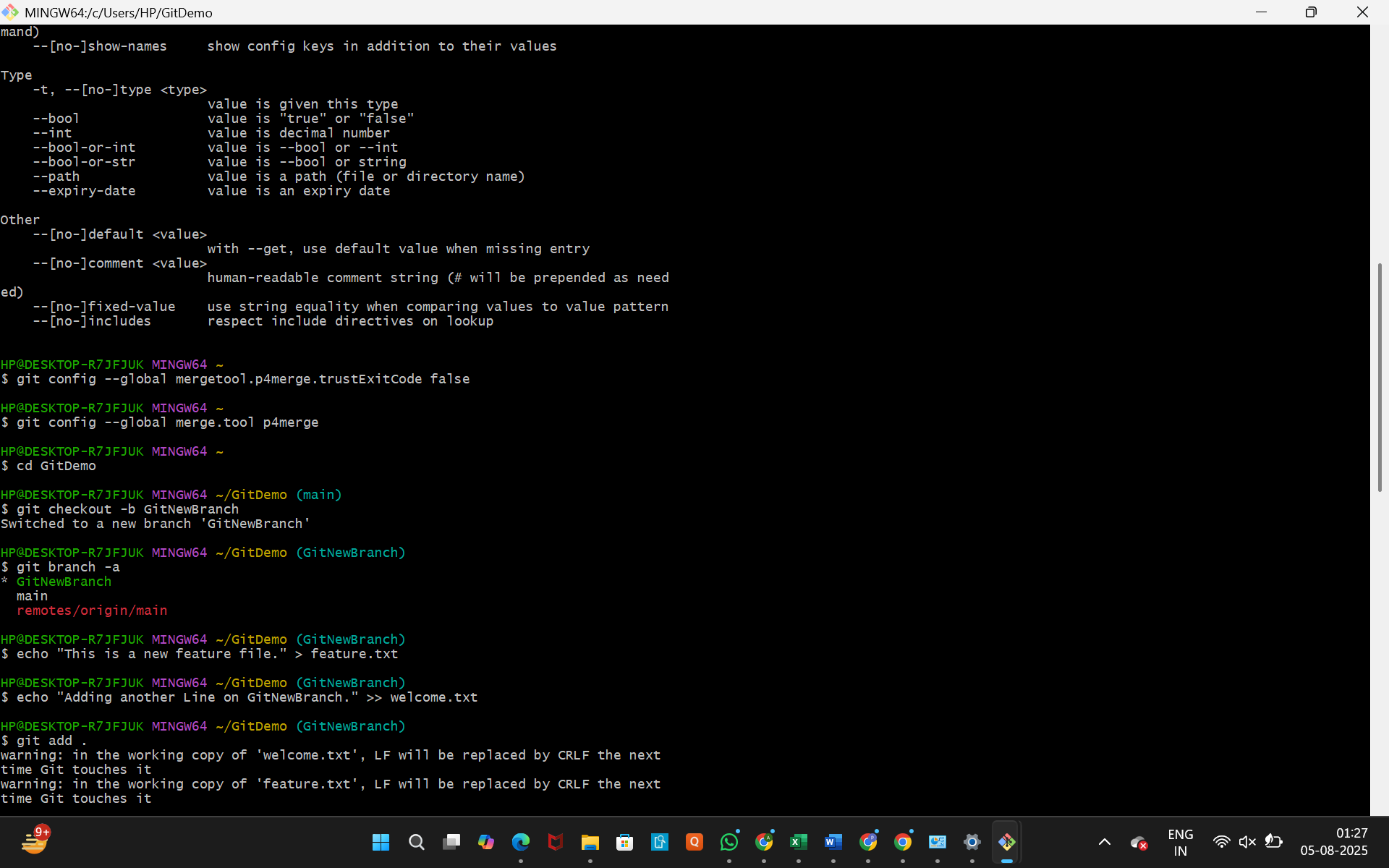
1. **Configure Git to use P4Merge (in Git Bash):**
   * Open your Git Bash shell.
   * Execute the following commands to tell Git to use P4Merge for diffing and merging:











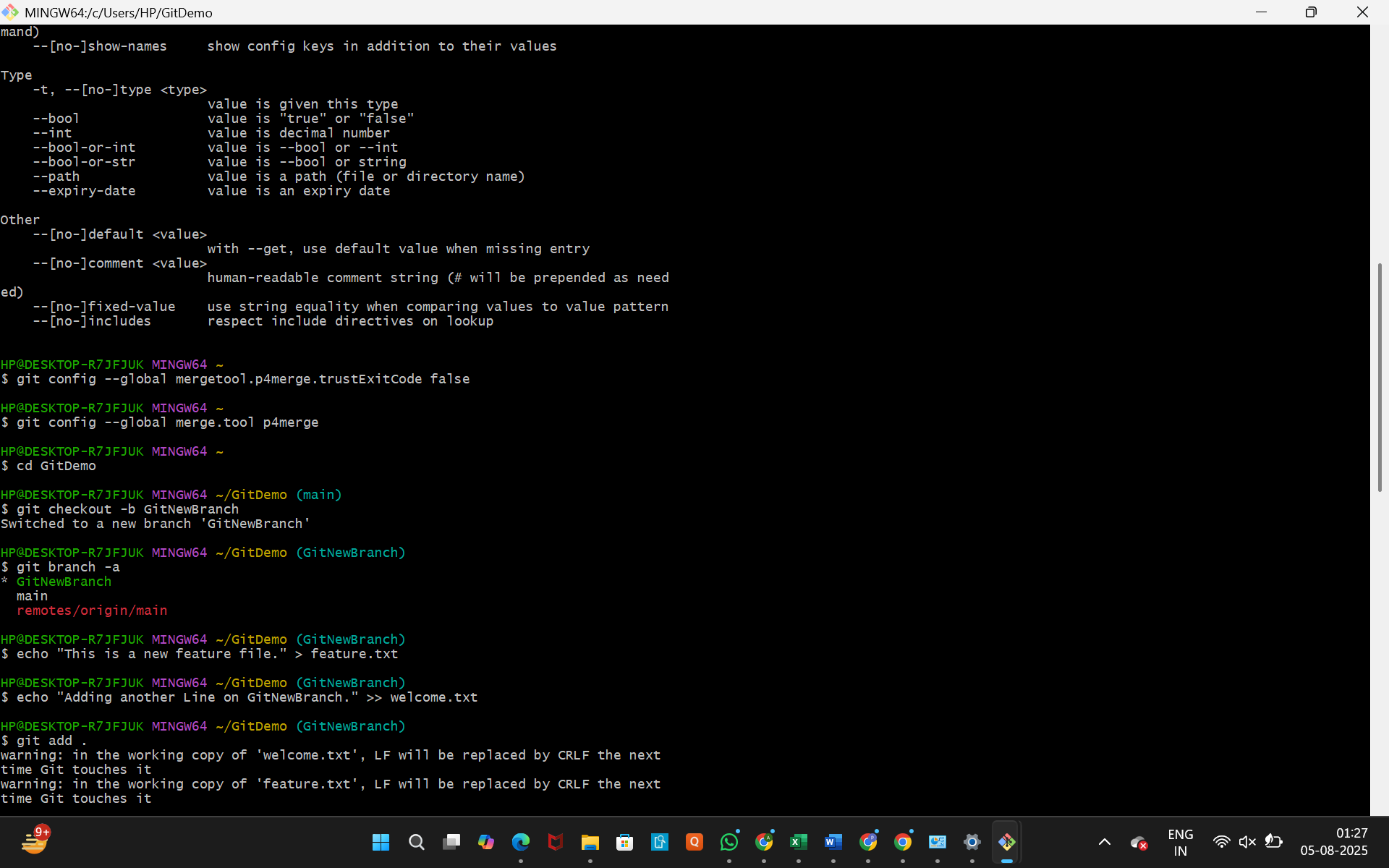
* + You can verify your configuration by running git config --global --list and looking for the diff and mergetool sections.

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

We will assume you are starting in your GitDemo project directory from the previous lab.

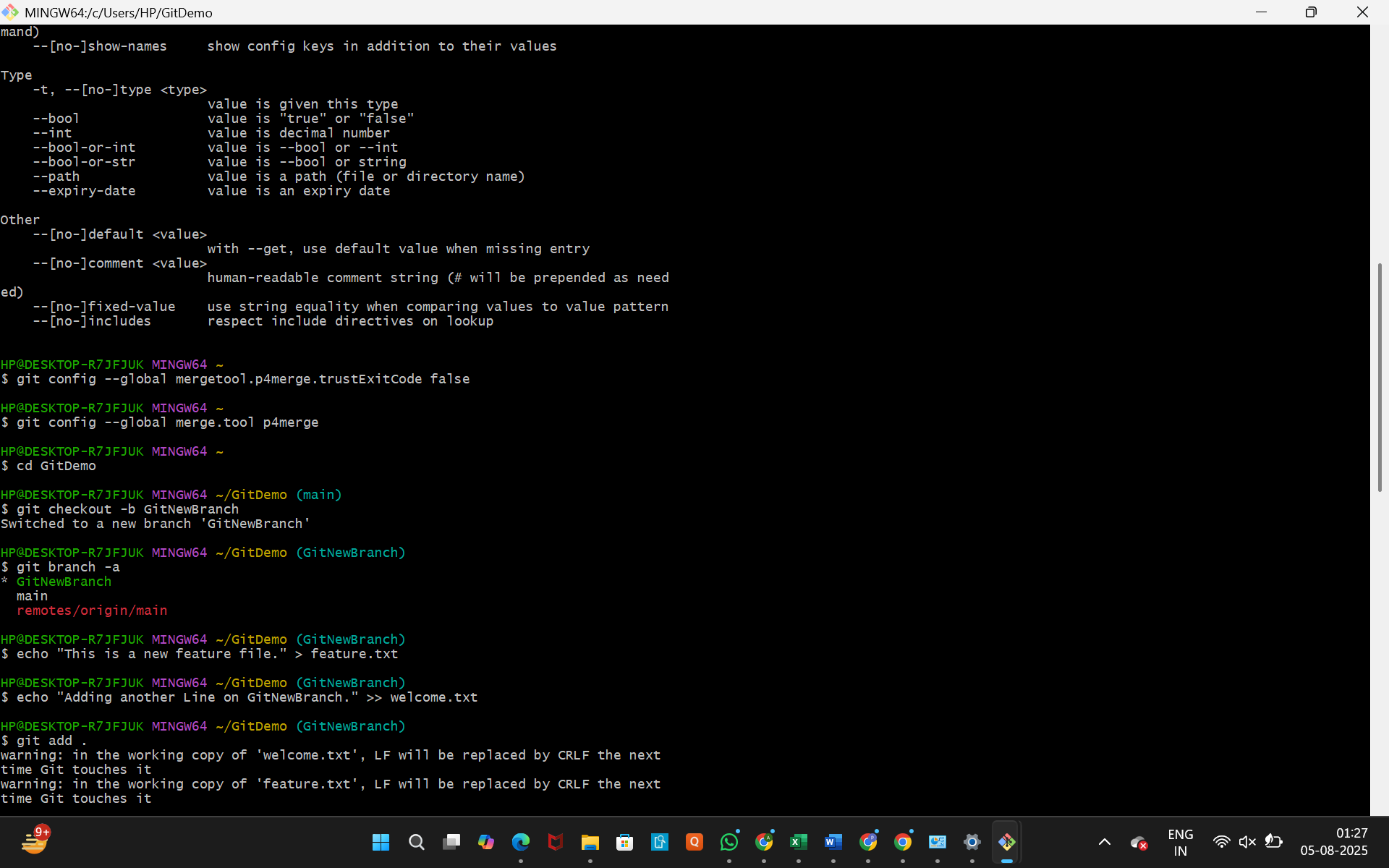
**Branching:**

1. **Create a new branch "GitNewBranch".**
   * **Purpose:** To create a new, separate line of development where you can make changes without affecting your main (or master) branch.
   * **Command:** In your Git Bash shell, navigate to your GitDemo project directory (e.g., cd GitDemo) and execute:



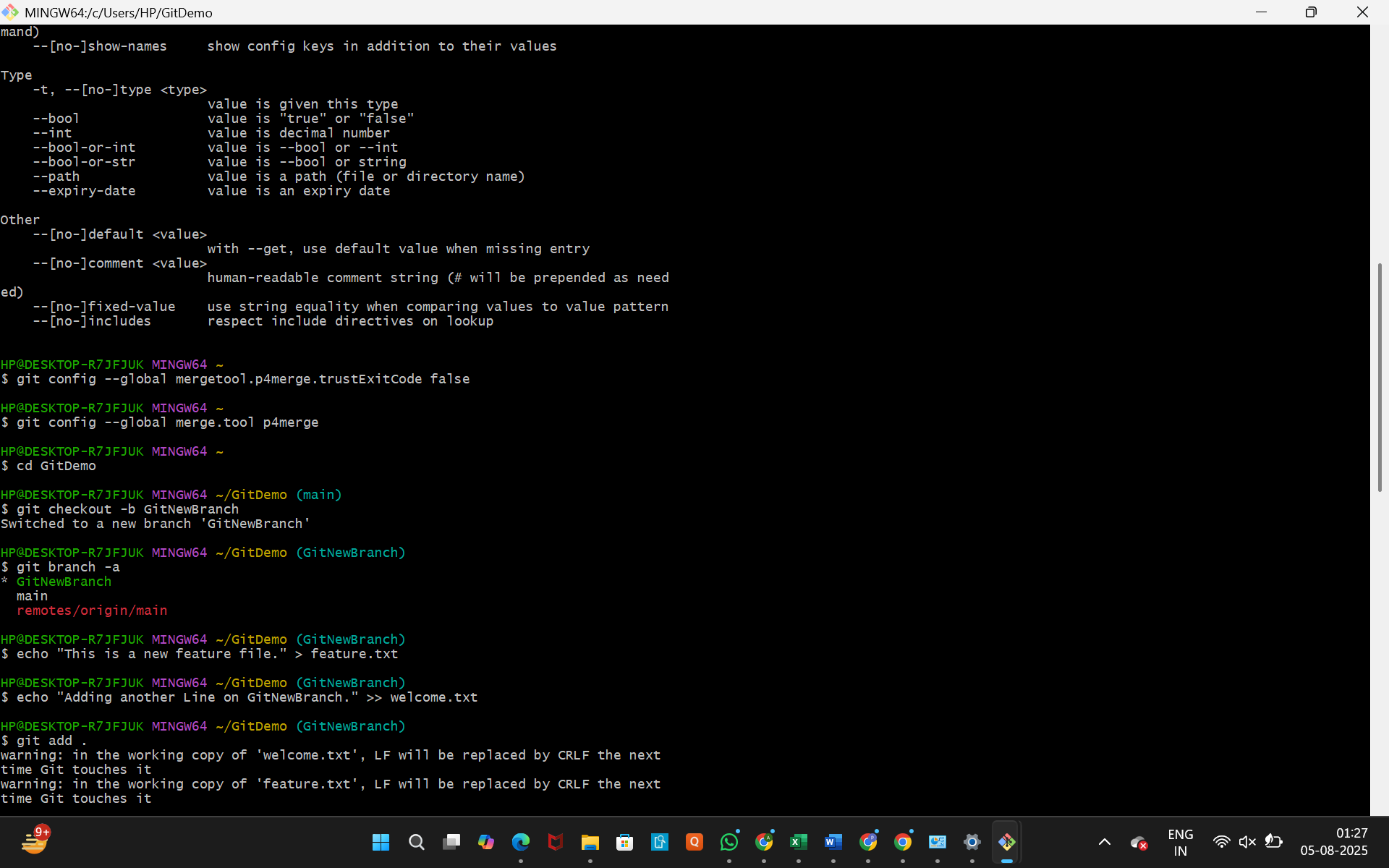
* + **Explanation:** git checkout -b is a convenient command that both creates a new branch (-b) and immediately switches your working directory to that new branch.

1. **List all the local and remote branches available in the current trunk. Observe the "\*" mark which denote the current pointing branch.**
   * **Purpose:** To see all branches in your repository and confirm you are on GitNewBranch.
   * **Command:**

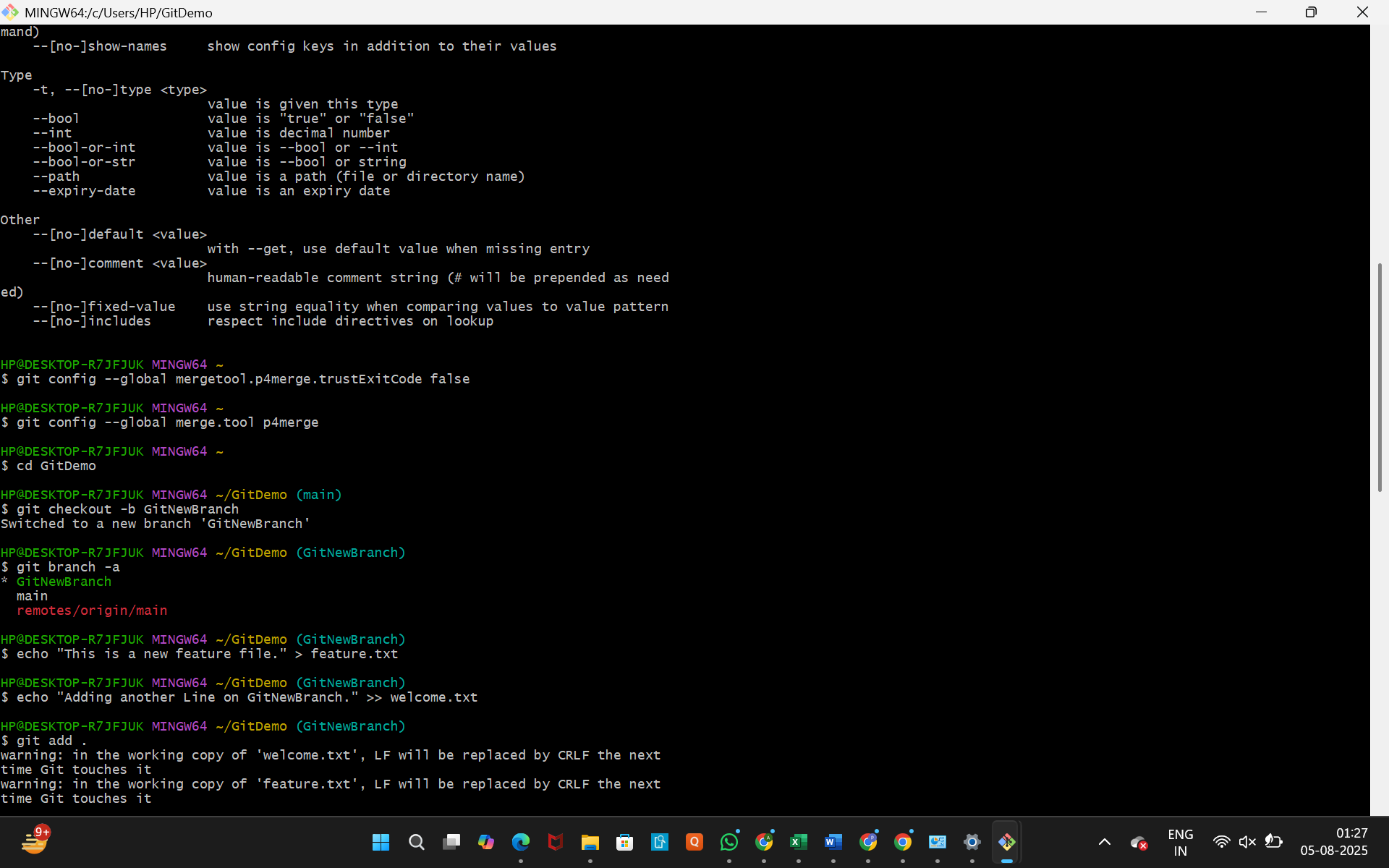


* + **What to expect:** You will see a list of your local branches (e.g., main, GitNewBranch) and possibly remote-tracking branches (e.g., remotes/origin/main). The branch you are currently on will have an asterisk (\*) next to it. You should see \* GitNewBranch.

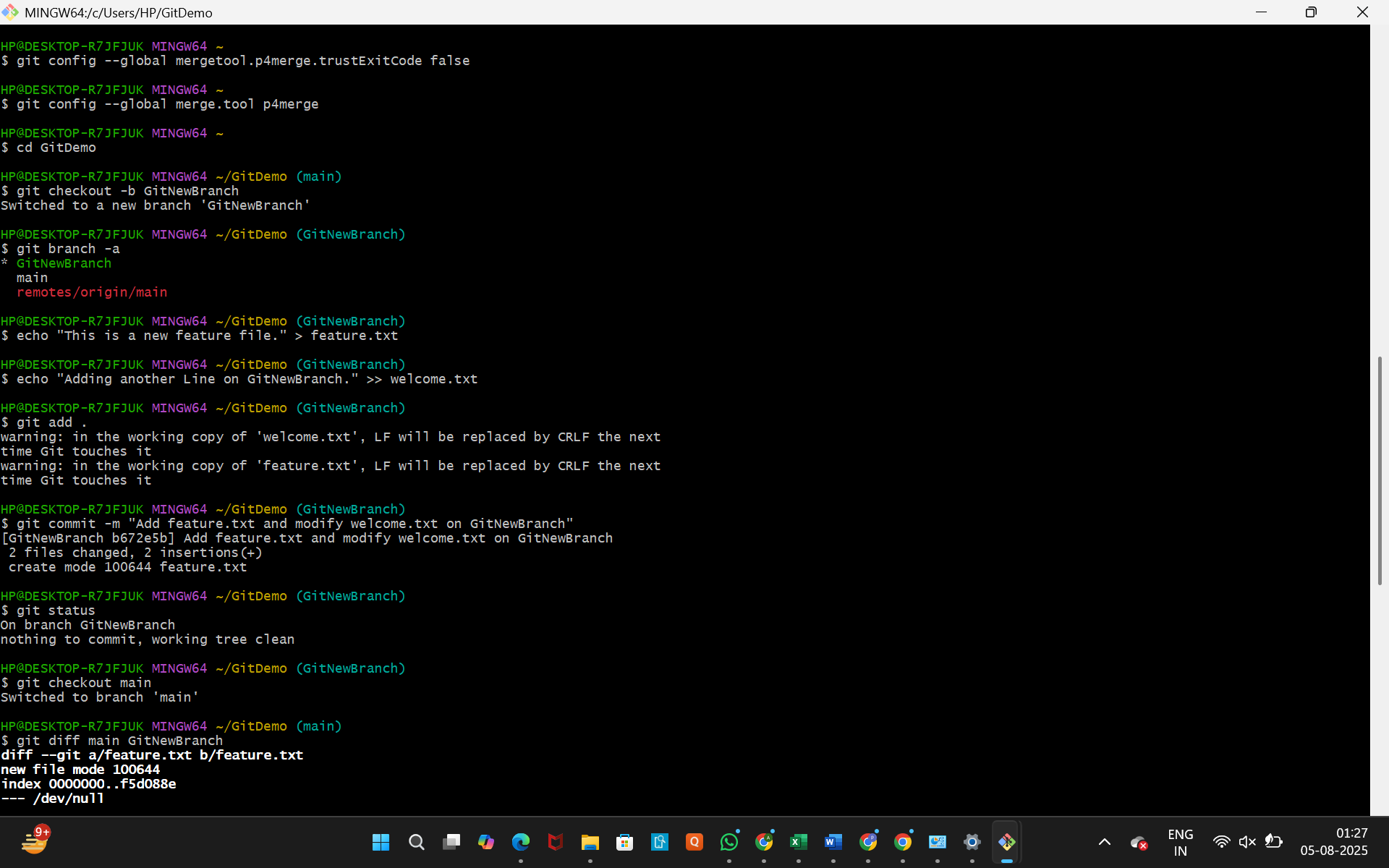
1. **Switch to the newly created branch. Add some files to it with some contents.**
   * **Purpose:** To demonstrate making changes specifically on your new branch.
   * **Action (already done):** You are already on GitNewBranch if you used git checkout -b in step 1. If you just used git branch GitNewBranch, you would need git checkout GitNewBranch first.
   * **Action (Add files):** Create a new file (e.g., feature.txt) in your GitDemo directory with some content:



* + **Action (Modify existing, optional):** You could also modify welcome.txt or any other file:

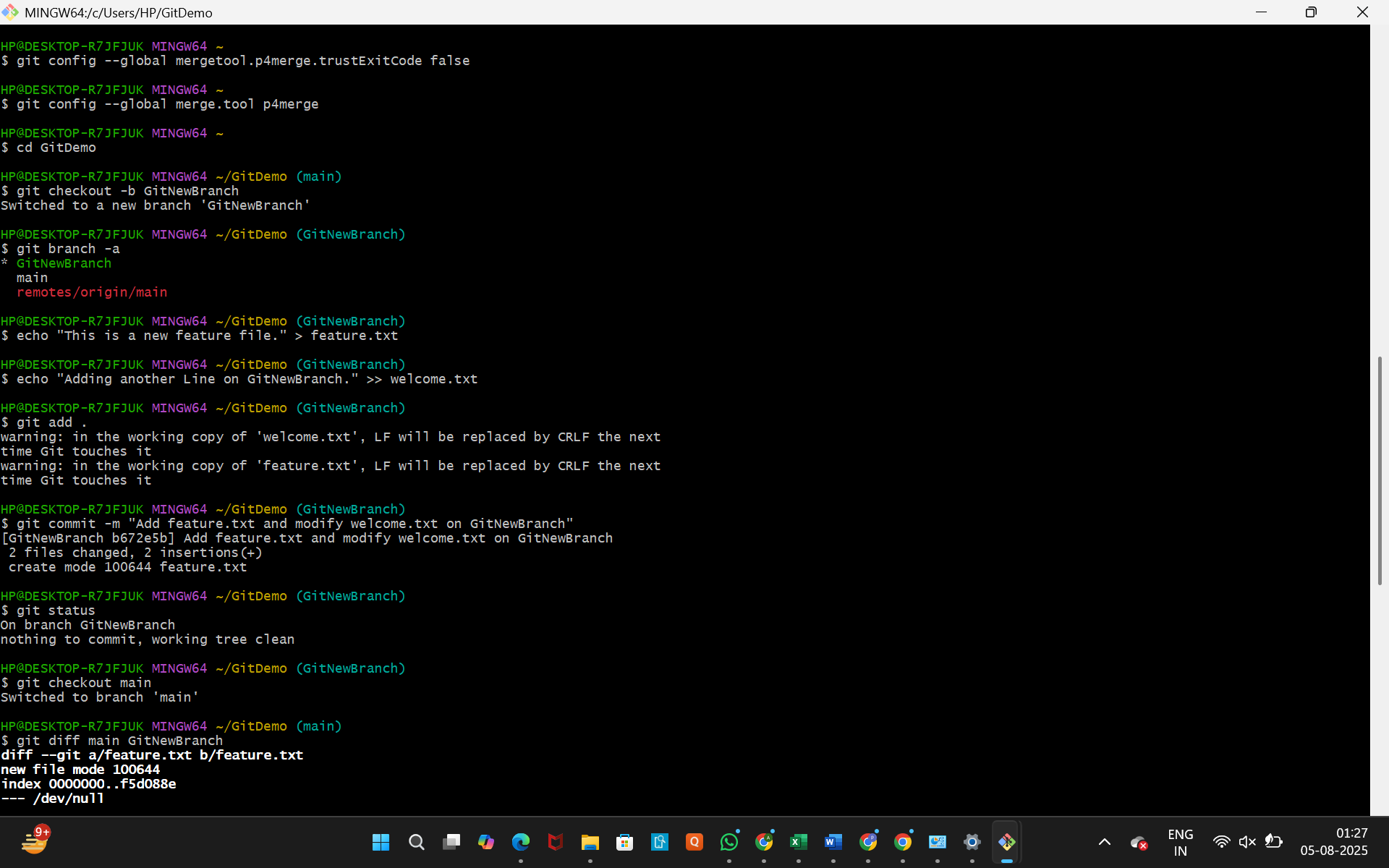


1. **Commit the changes to the branch.**
   * **Purpose:** To save the changes you made in GitNewBranch into that branch's history.
   * **Command (Stage changes):**



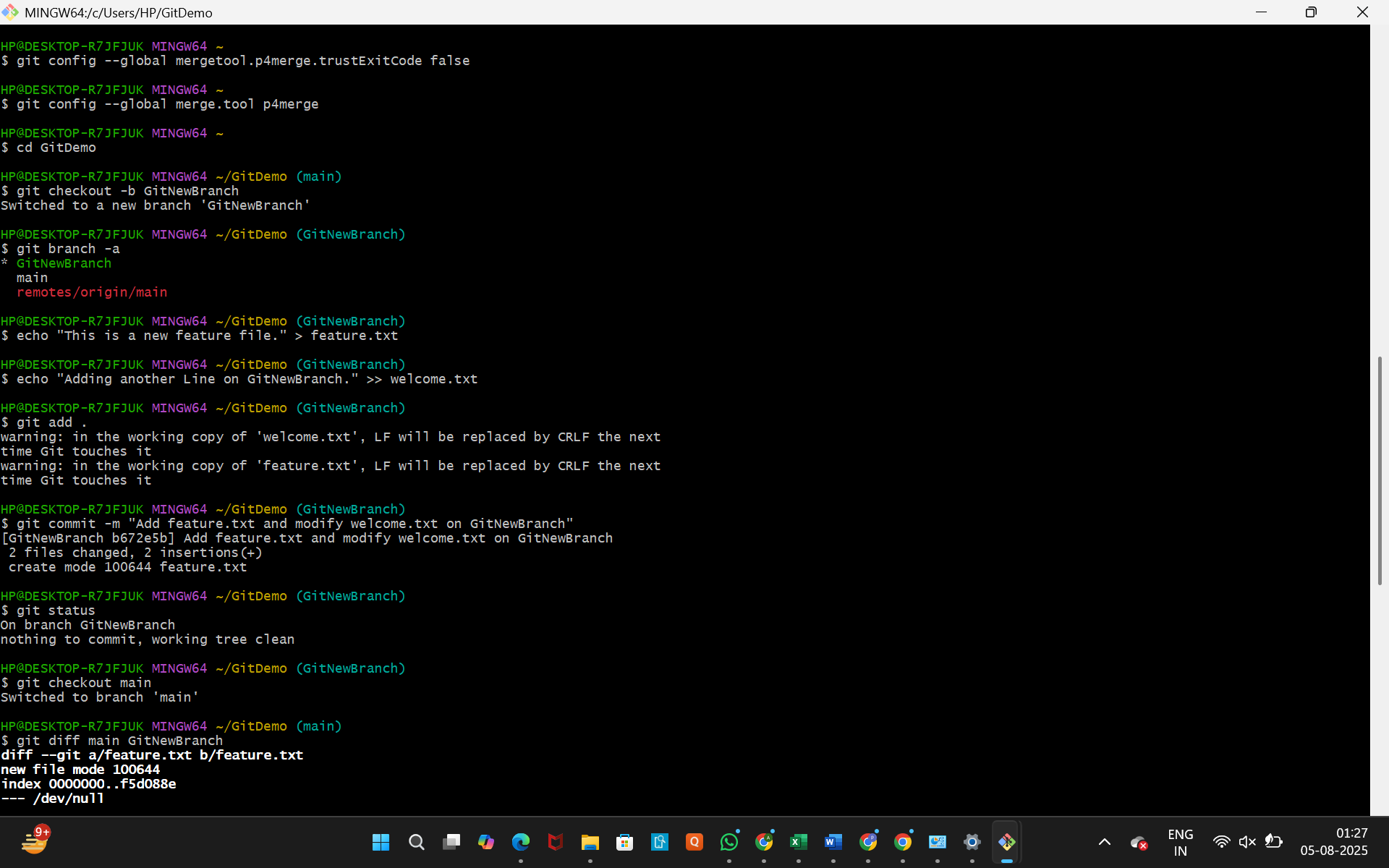
(This stages all new/modified files. You could also do git add feature.txt welcome.txt individually).

* + **Command (Commit changes):**



(Use a descriptive message.)

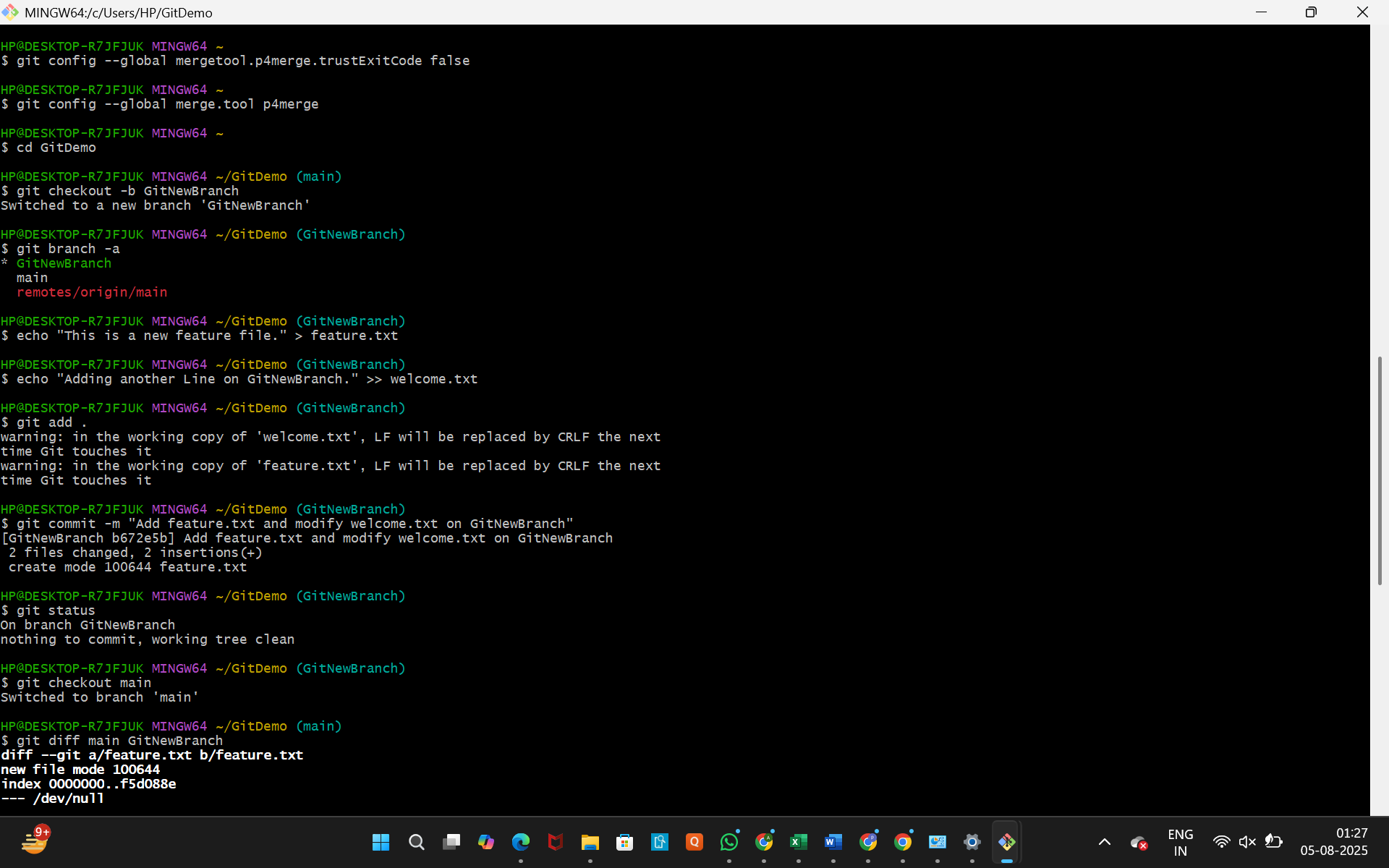
1. **Check the status with "git status" command.**
   * **Purpose:** To confirm that your working directory is clean and all changes are committed on GitNewBranch.
   * **Command:**



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

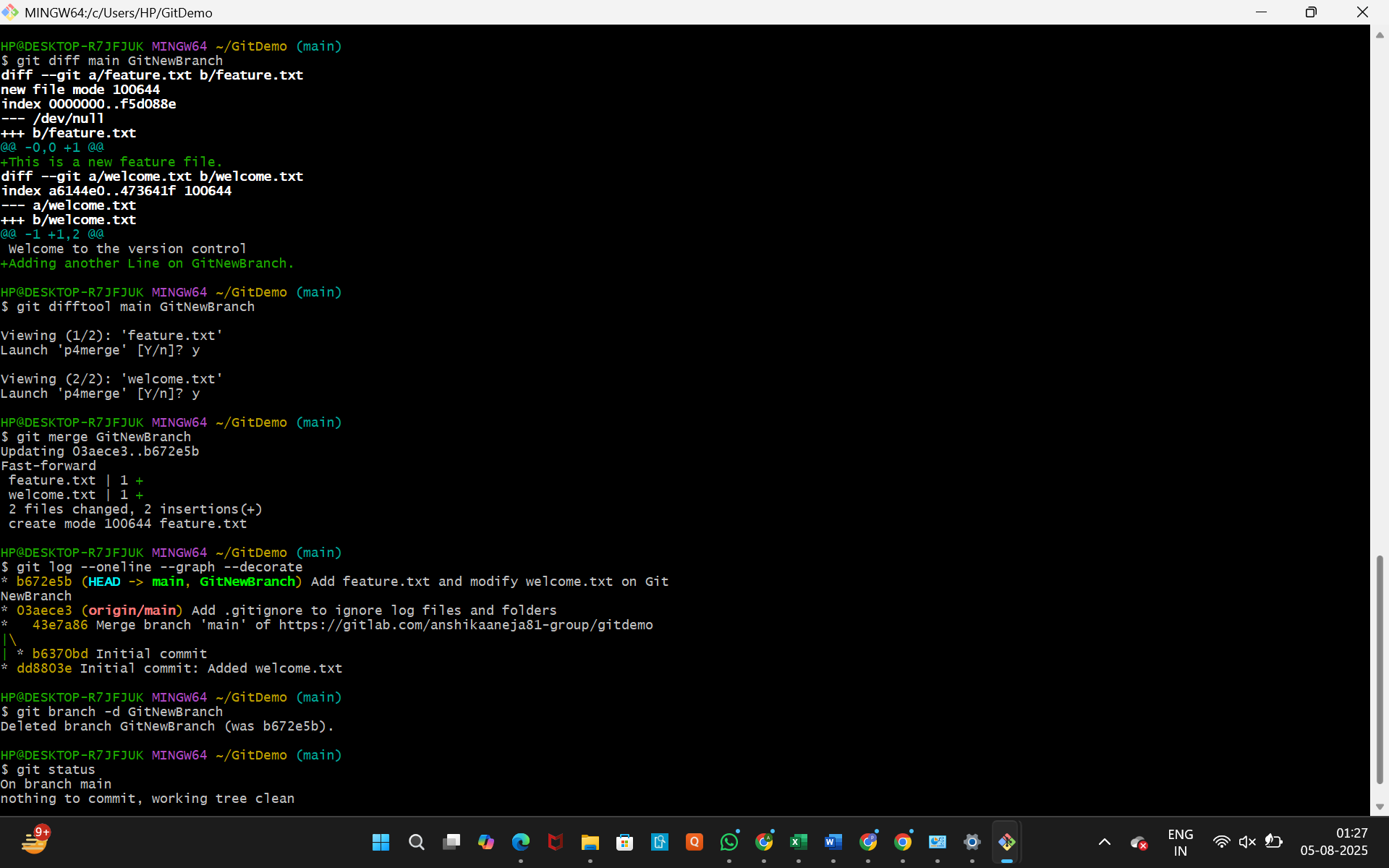
**Merging:**

1. **Switch to the main (or master/trunk) branch.**
   * **Purpose:** You need to be on the branch where you want to *incorporate* the changes from GitNewBranch.
   * **Command:**



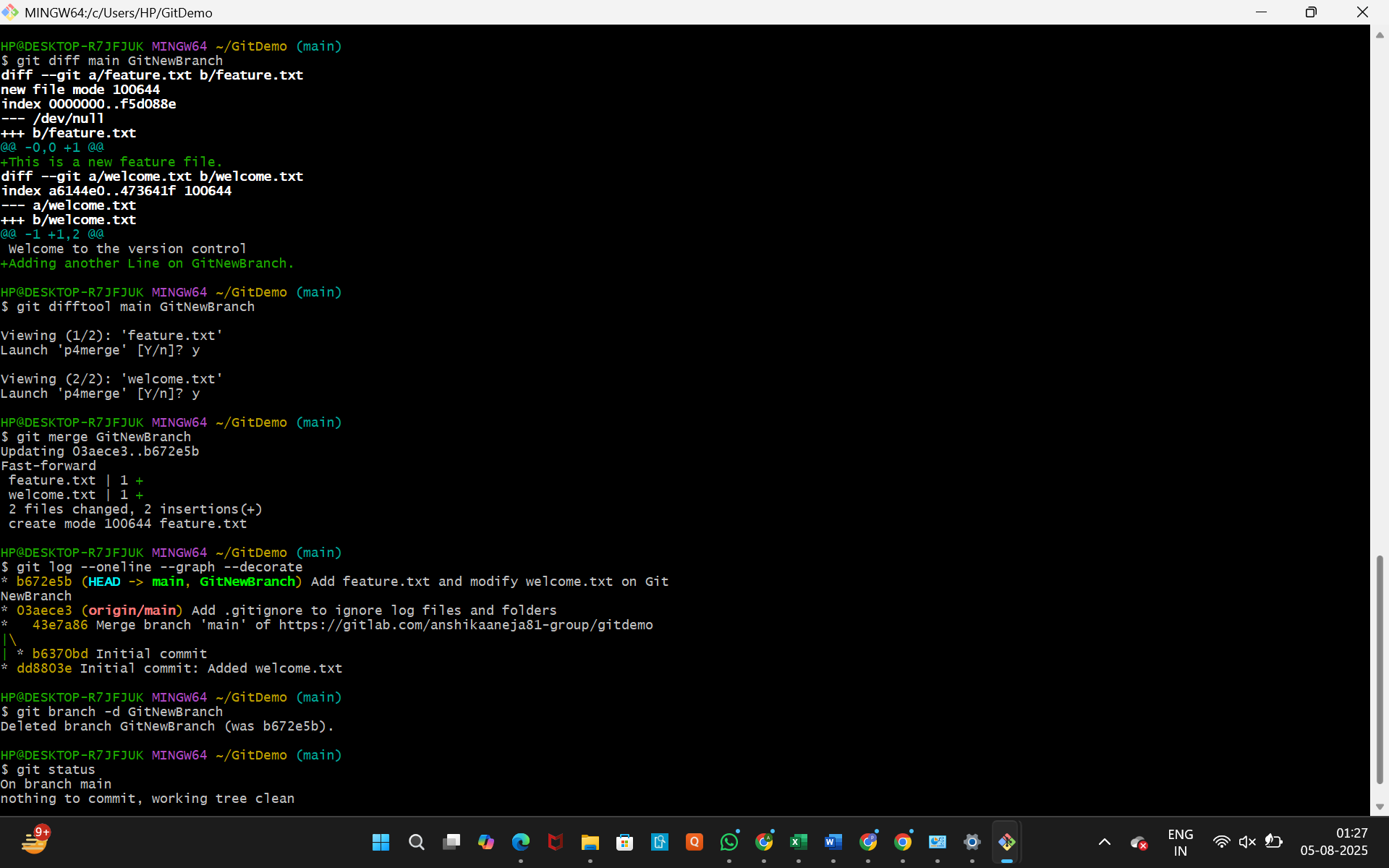
* + **What to expect:** Your Git Bash prompt will change to (main), and if you type ls, you will notice that feature.txt is *not* present in your working directory yet, because it only exists in GitNewBranch.

1. **List out all the visual differences between main and GitNewBranch. These provide the differences in command line interface.**
   * **Purpose:** To see a text-based comparison of the changes made on GitNewBranch relative to main.
   * **Command:**

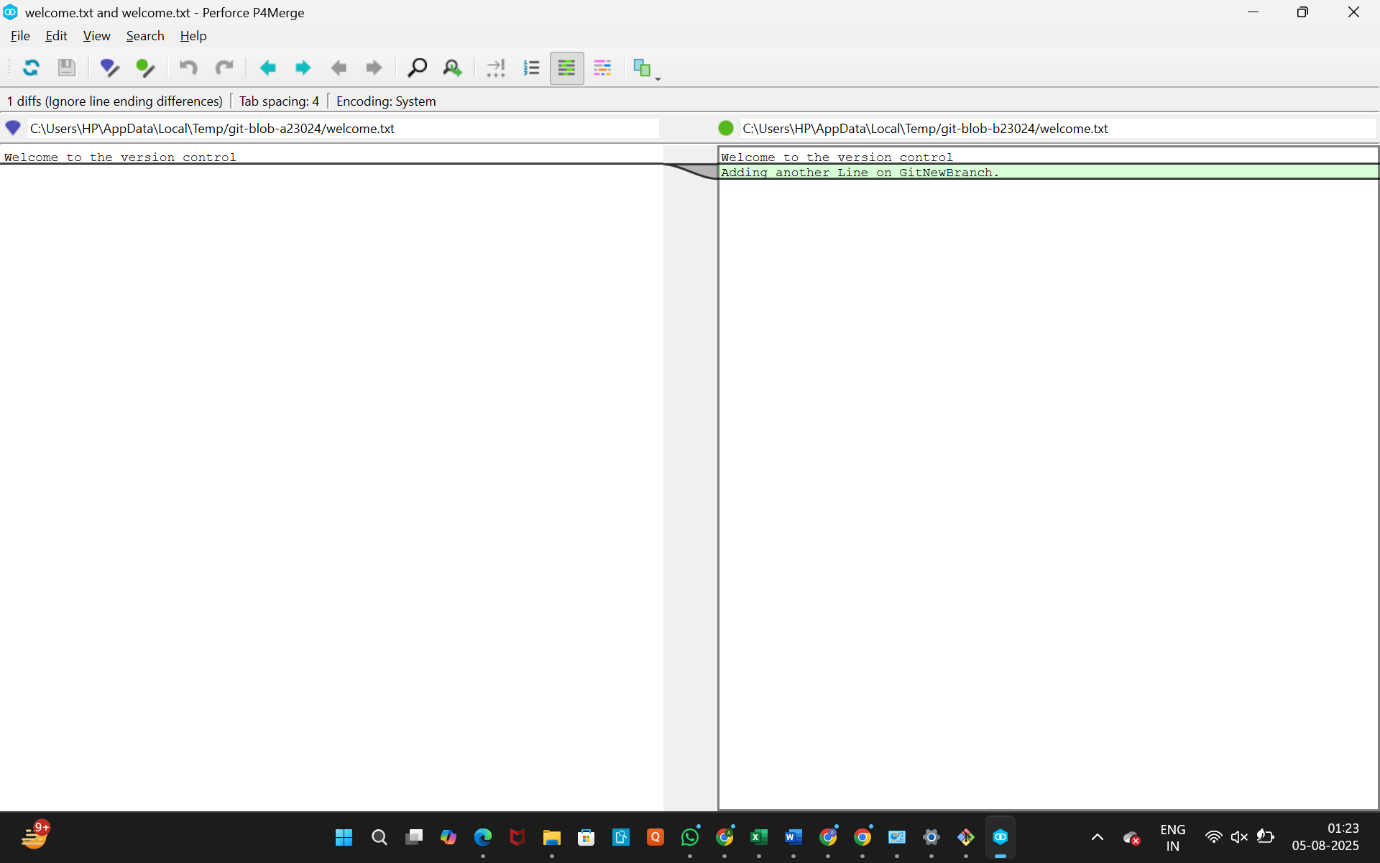


* + **What to expect:** Git will show you the lines that were added (like feature.txt) or changed in GitNewBranch compared to main.

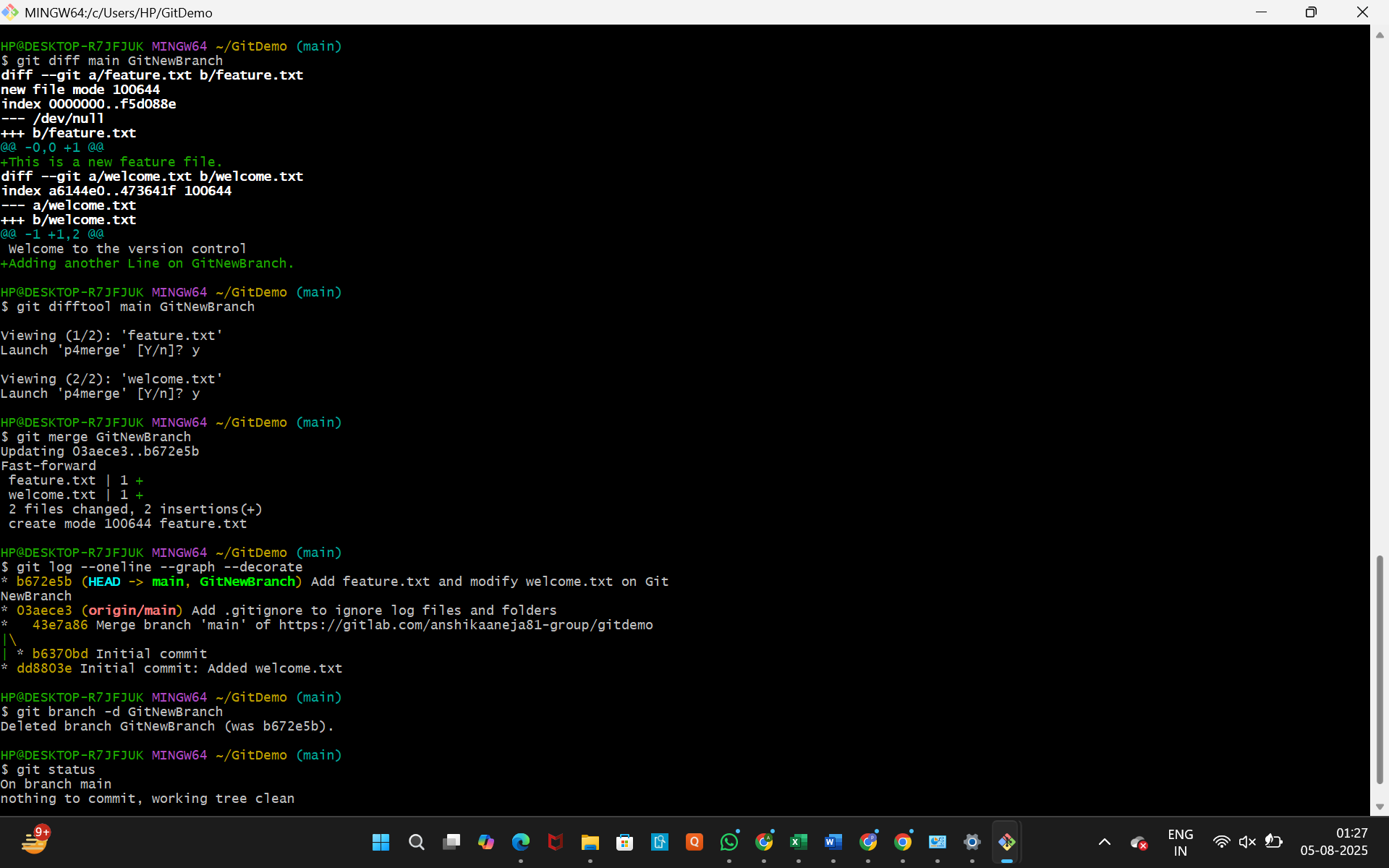
1. **List out all the visual differences between main and GitNewBranch using P4Merge tool.**
   * **Purpose:** To view the differences in a graphical diff tool (P4Merge), which is often easier to interpret.
   * **Command:**



* + **What to expect:** P4Merge (or your configured diff tool) should launch, displaying the differences between the two branches side-by-side. Close P4Merge when you are done inspecting the differences.

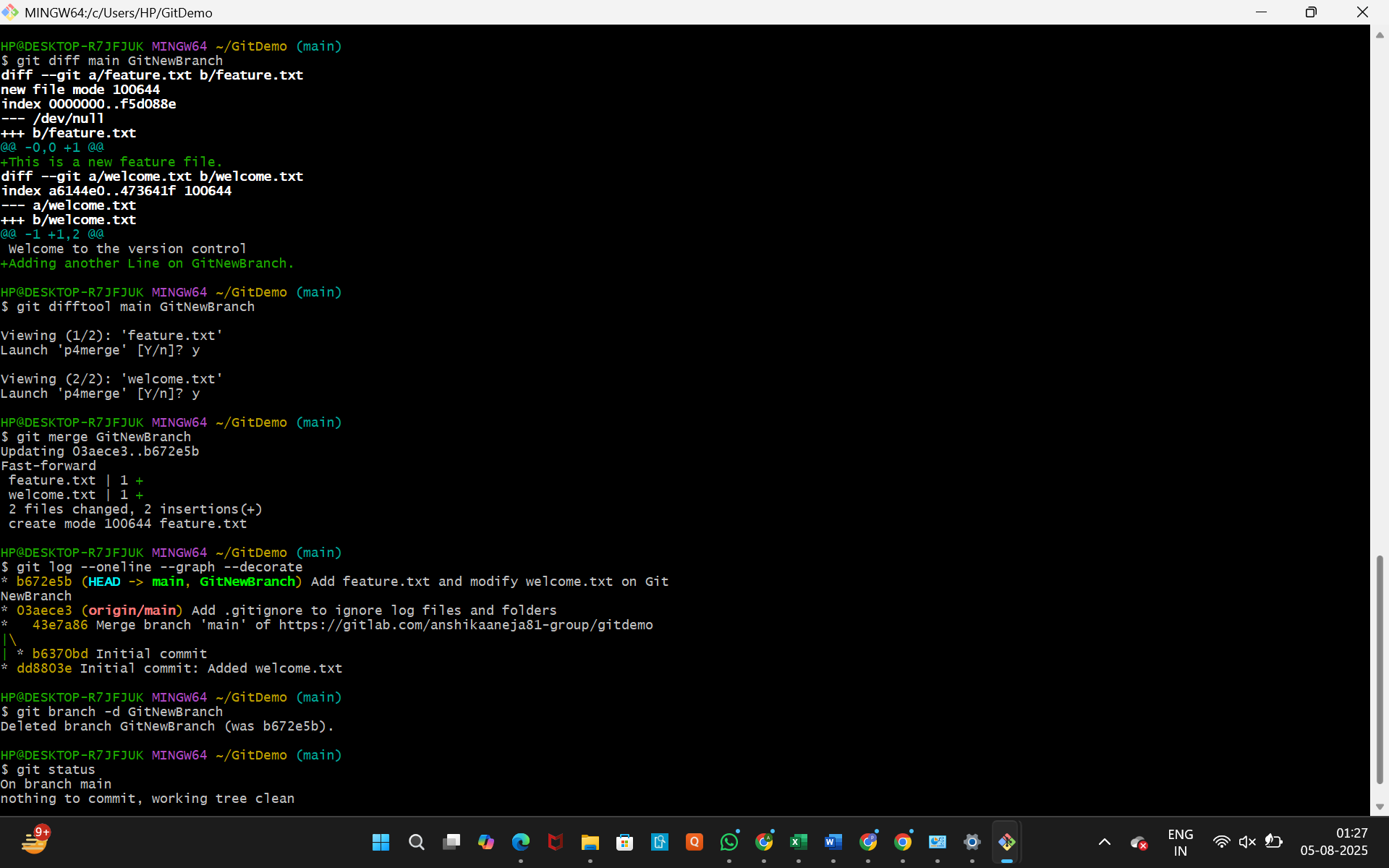


1. **Merge the source branch (GitNewBranch) to the main (or trunk) branch.**
   * **Purpose:** To integrate all the changes from GitNewBranch into your main branch.
   * **Command:** Ensure you are on the main branch (from step 1 of Merging), then execute:



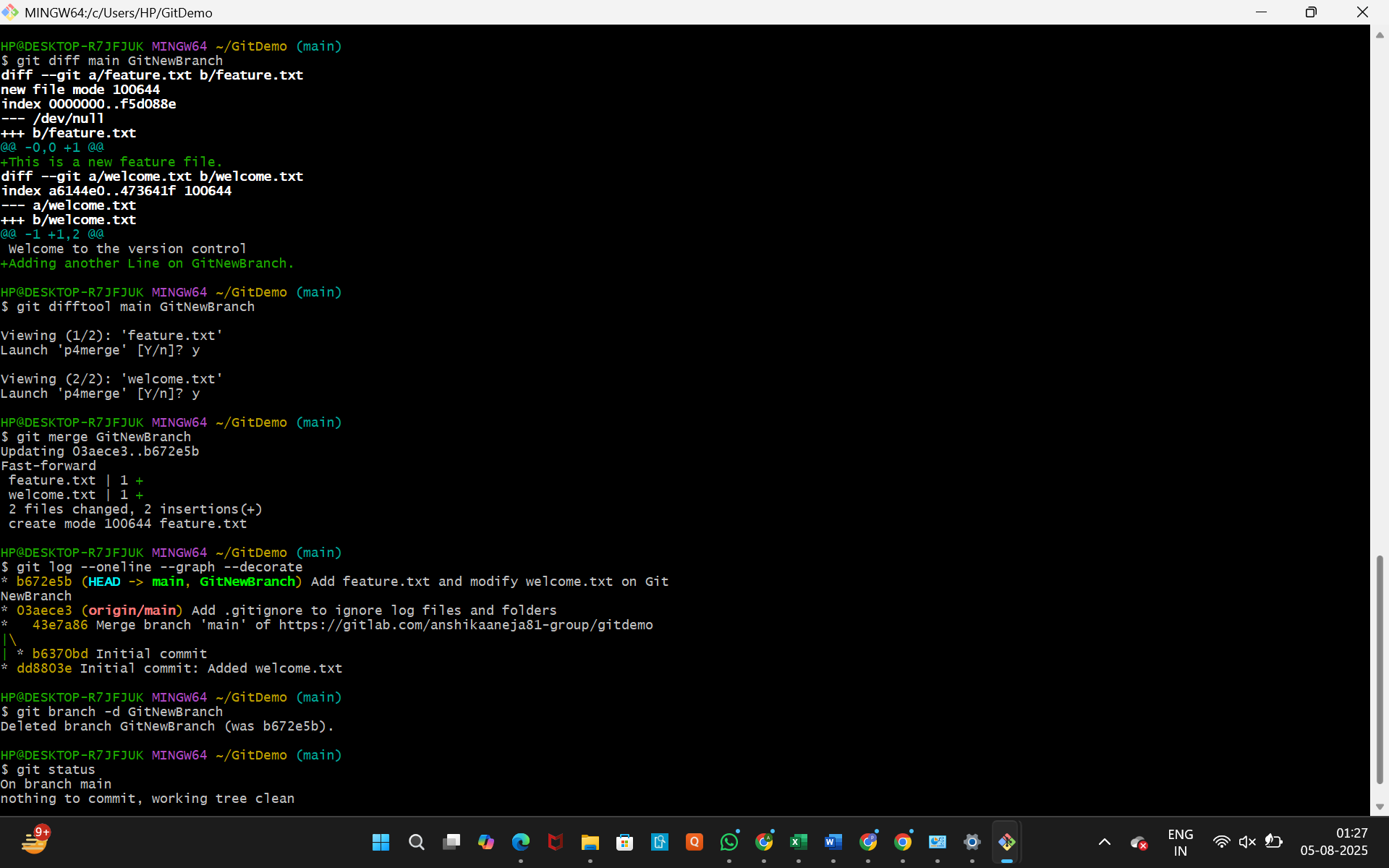
* + **What to expect:**
    - If there are no conflicts, Git will perform a "fast-forward" merge (if main hasn't diverged) or a regular merge (creating a merge commit if main has new commits).
    - You might see a default commit message appear in Notepad++ for a merge commit. Save and close it.
    - Your Git Bash will show output indicating the merge happened (e.g., "Updating...", "Fast-forward", or "Merge branch 'GitNewBranch'").
    - If you now type ls, you should see feature.txt in your main branch.

1. **Observe the logging after merging using "git log --oneline --graph --decorate".**
   * **Purpose:** To visualize the commit history, including how the branches merged.
   * **Command:**

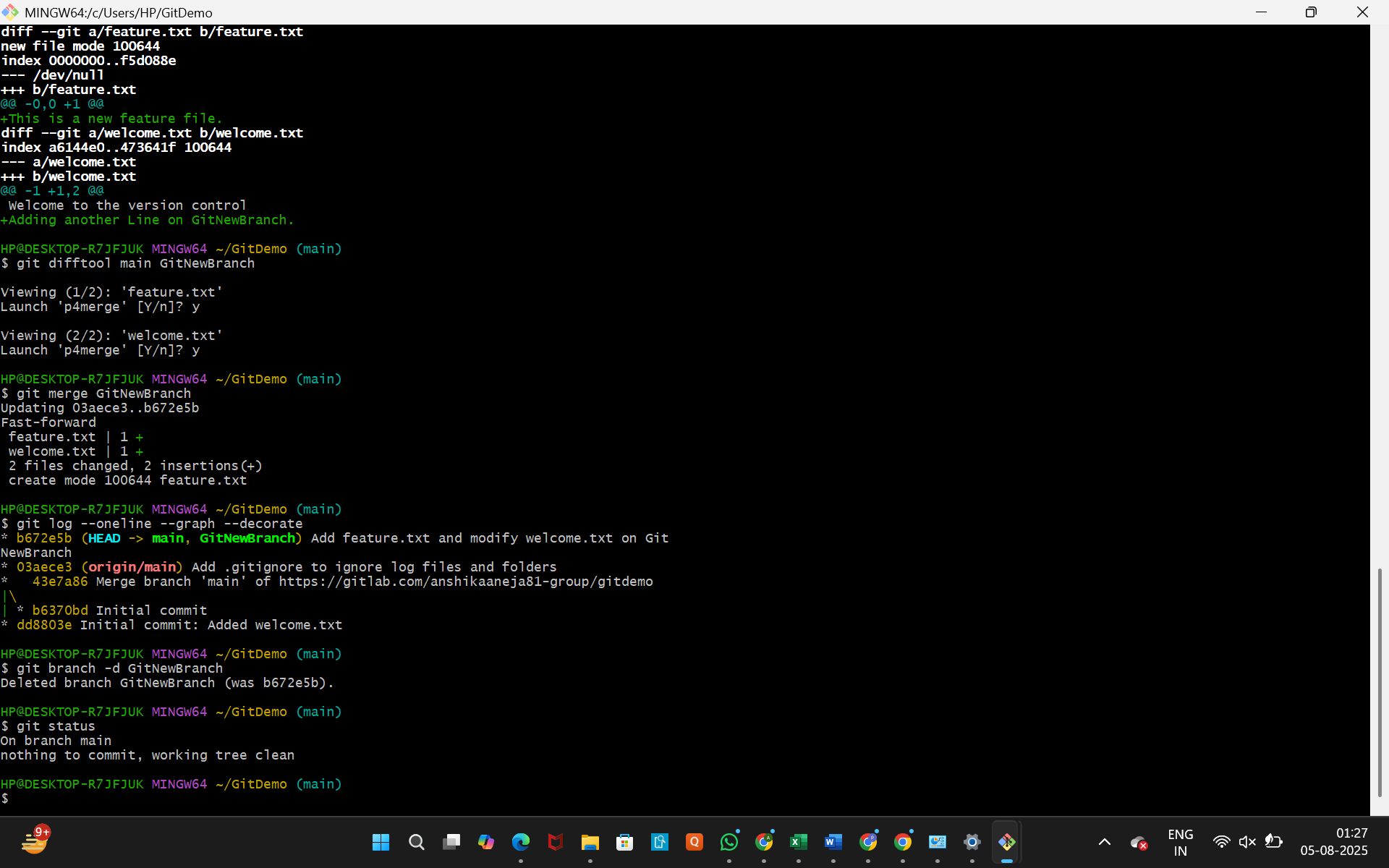


* + **What to expect:** You will see a graphical representation of your commit history. You should clearly see where GitNewBranch diverged and then merged back into main. The (HEAD -> main, origin/main) annotation should now be on your latest commit, which is the merge commit.

1. **Delete the branch after merging with the trunk and observe the git status.**
   * **Purpose:** Once changes are merged into main, the feature branch is often no longer needed.
   * **Command (Delete branch):**



* + - **Note:** git branch -d will only delete a branch if its changes have been fully merged. If you wanted to force deletion of an unmerged branch (not recommended here), you would use git branch -D.
  + **Command (Observe status):**



* + **What to expect:**
    - The git branch -d command should execute without errors, confirming GitNewBranch is deleted.
    - git status should show On branch main and nothing to commit, working tree clean. If you run git branch -a again, GitNewBranch should no longer be listed as a local branch.