**DN 4.0 Dotnet FSE**

**Name: Sparsh Guha**

**Superset ID:6361106**

**Week 8**

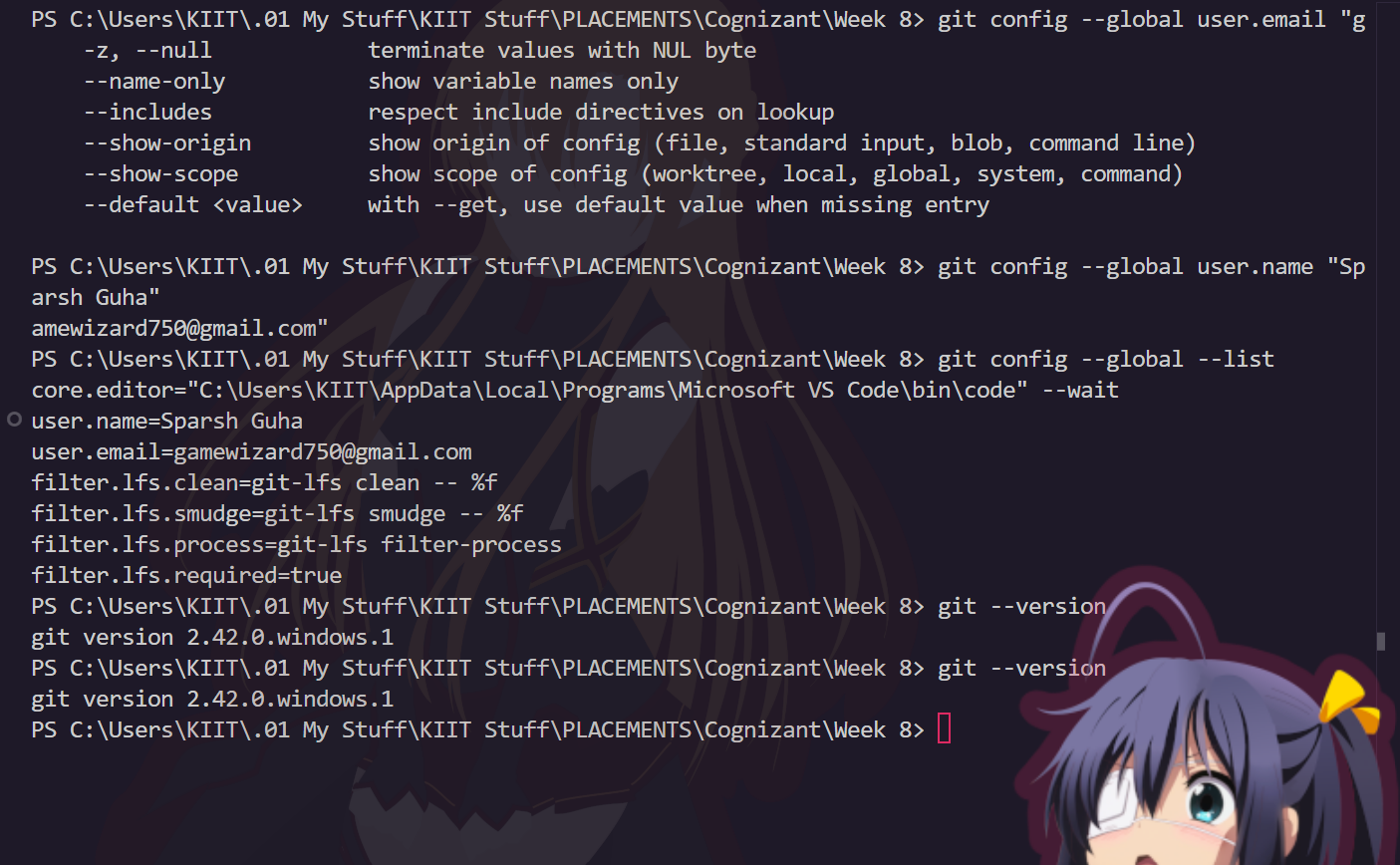
**QUESTION-1**

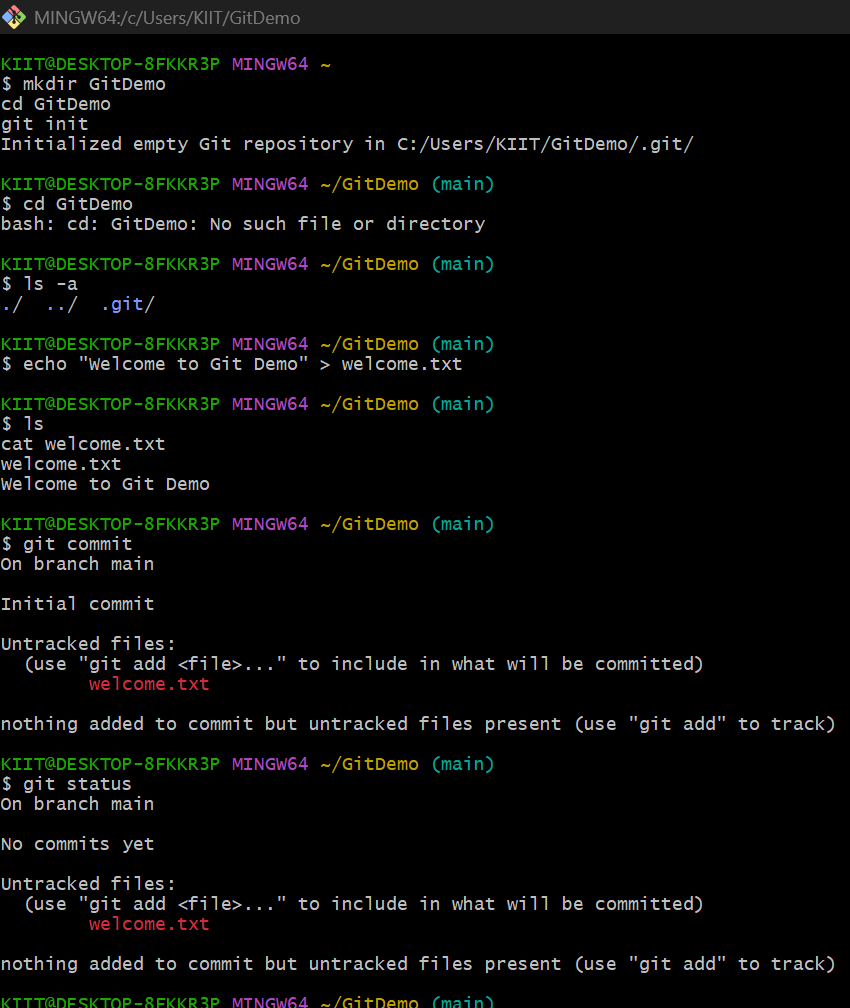
**Familiar with Git commands like git init, git status, git add, git commit, git push, and git pull.**

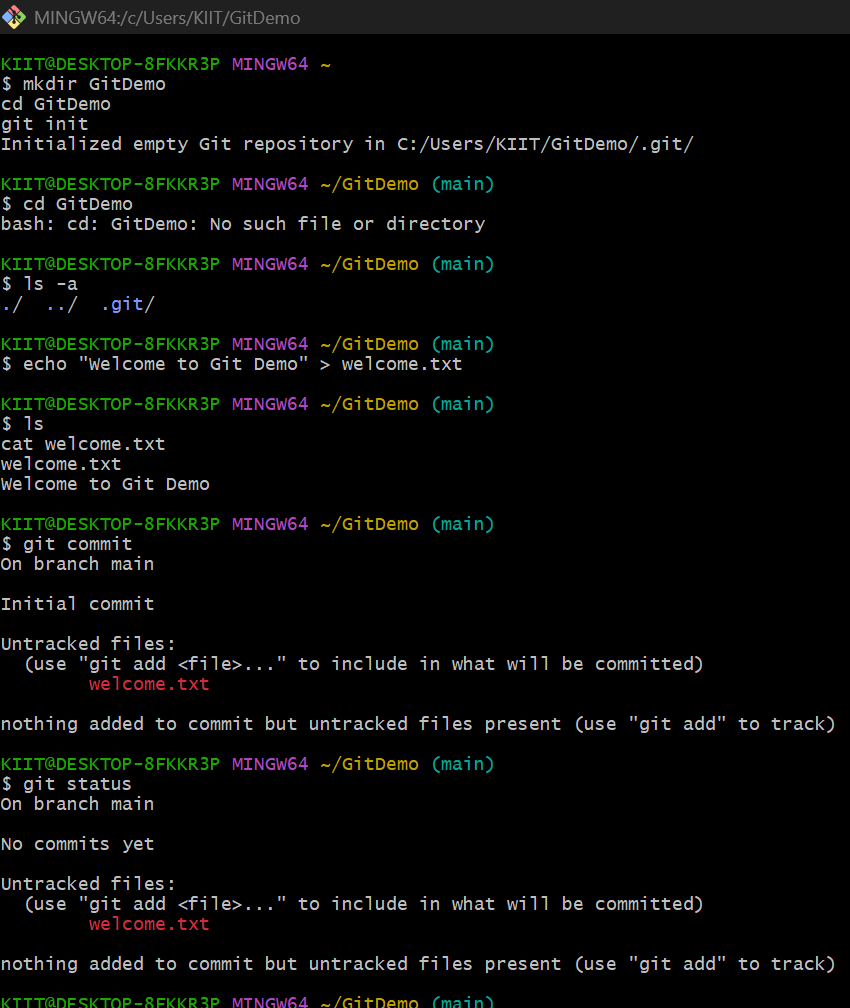
A. Check Git Installation  
   
git --version  
B. Configure User Name and Email  
git config --global user.name "Sparsh"  
git config --global user.email ["gamwizard750@gmail.com"](mailto:\"gamwizard750@gmail.com\")

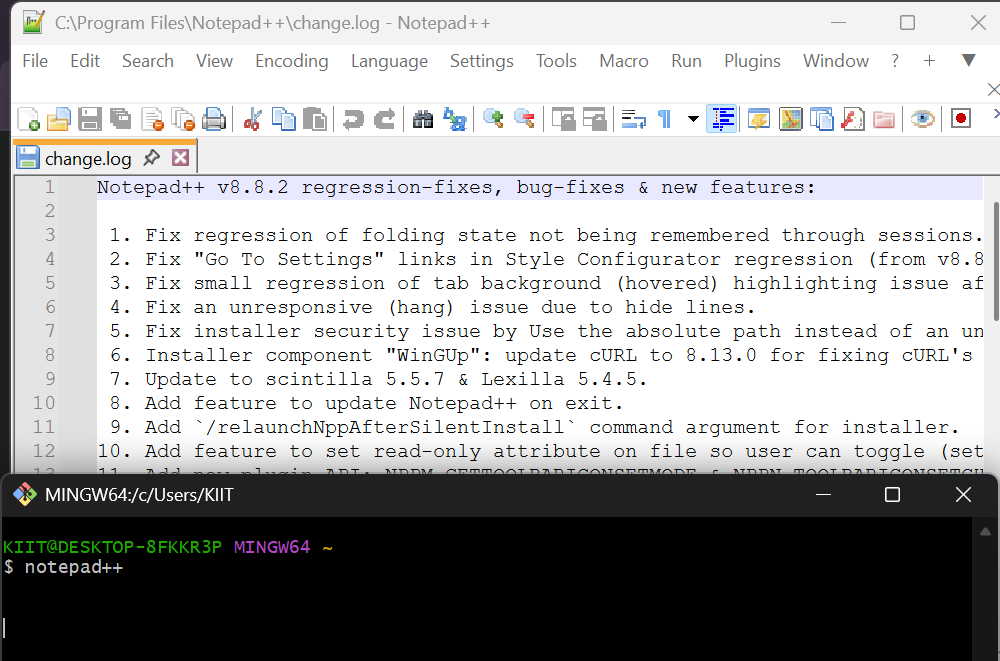
(Email registered with Github)  
  
C. Verify Configuration  
   
git config --global --list  
  
Output should show your user name and email.  
A. Test Notepad++ from Git Bash  
   
notepad++  
  
Control Panel → System → Advanced system settings → Environment Variables → Edit PATH  
B. Restart Git Bash and Test Again  
notepad++  
  
C. Verify Editor Setting  
   
git config --global -e  
  
Step 3: Add a File to a Source Code Repository  
A. Initialize a New Repository  
   
mkdir GitDemo  
cd GitDemo  
git init  
  
B. Check for Hidden Git Files  
   
ls -a  
  
C. Create a File and Add Content  
   
echo "Welcome to Git Demo" > welcome.txt  
  
D. Verify File Exists and Content  
   
ls  
cat welcome.txt  
  
E. Check Git Status  
   
git status  
  
welcome.txt should be listed as an untracked file.  
F. Stage the File  
   
git add welcome.txt  
  
G. Commit with Multi-line Comment (opens Notepad++)  
   
git commit  
  
git status  
  
Working directory should be clean.  
Step 4: Connect to Remote Repository (GitLab Example)  
A. Create a New Project on GitLab Named GitDemo  
B. Add Remote Origin  
   
git remote add origin https://gitlab.com/yourusername/GitDemo.git  
  
C. Pull Any Remote Changes (First Time)  
   
git pull origin master  
  
(If your GitLab uses main instead of master, use main.)  
  
D. Push Local Commits to Remote  
   
git push origin master

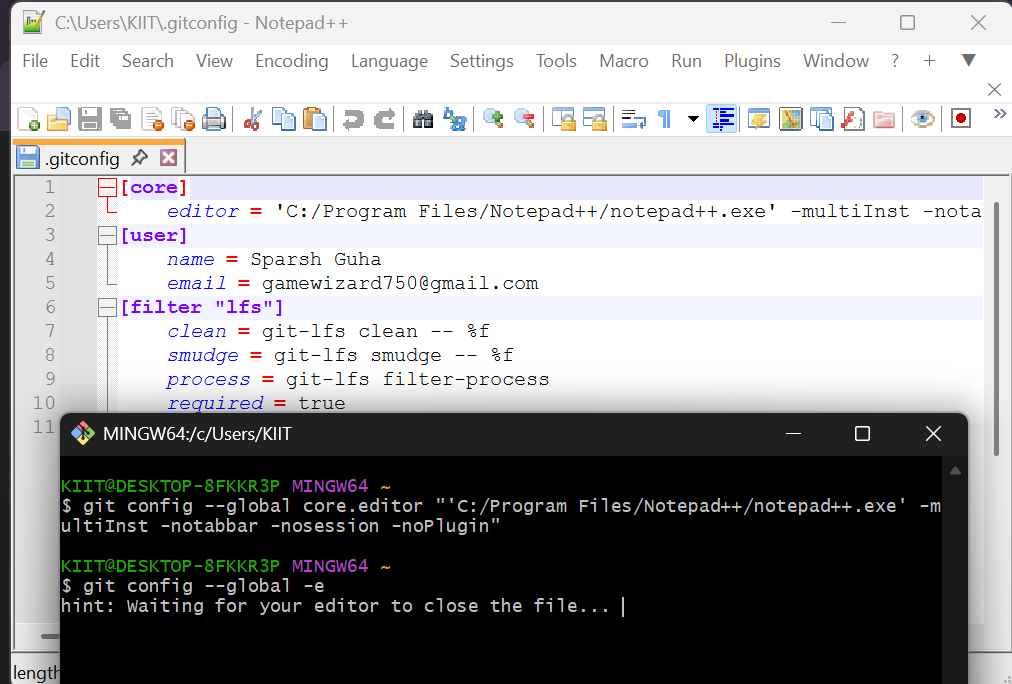
**OUTPUT:**

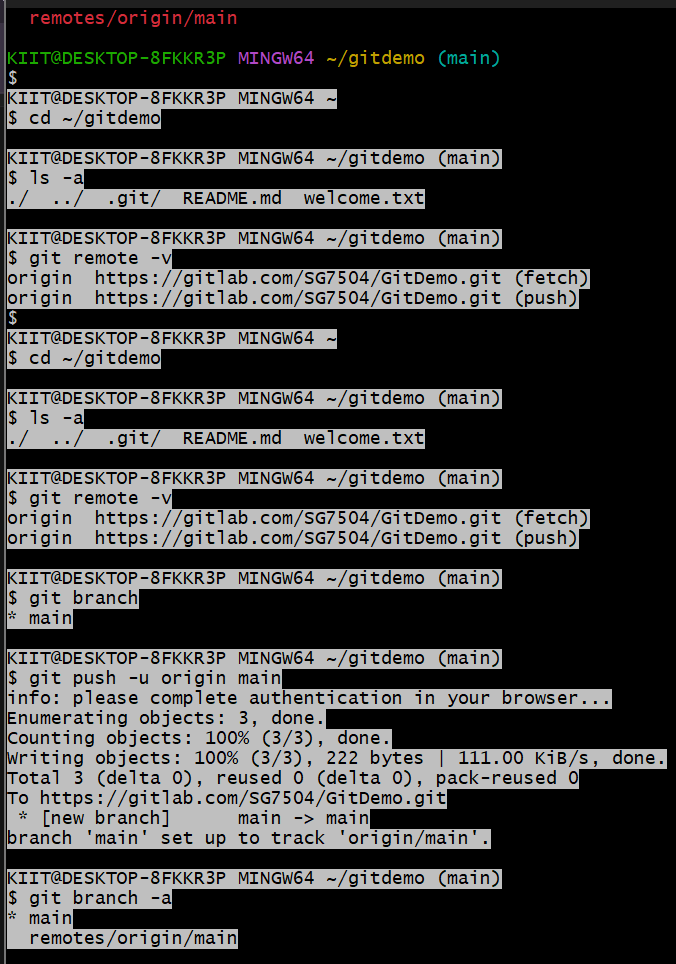
****

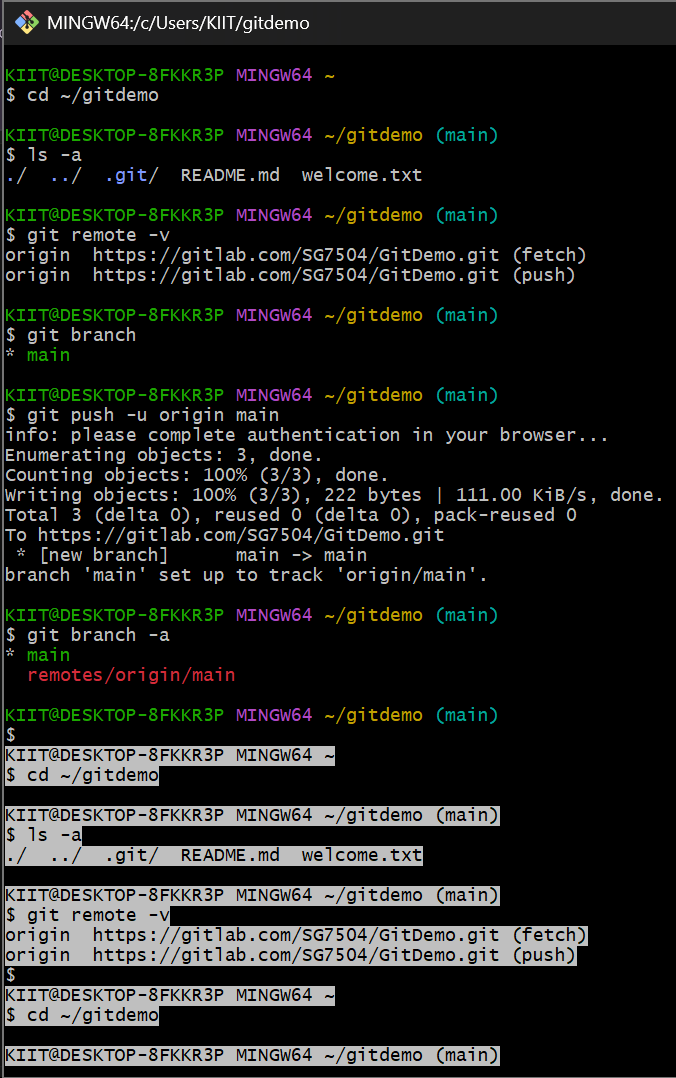










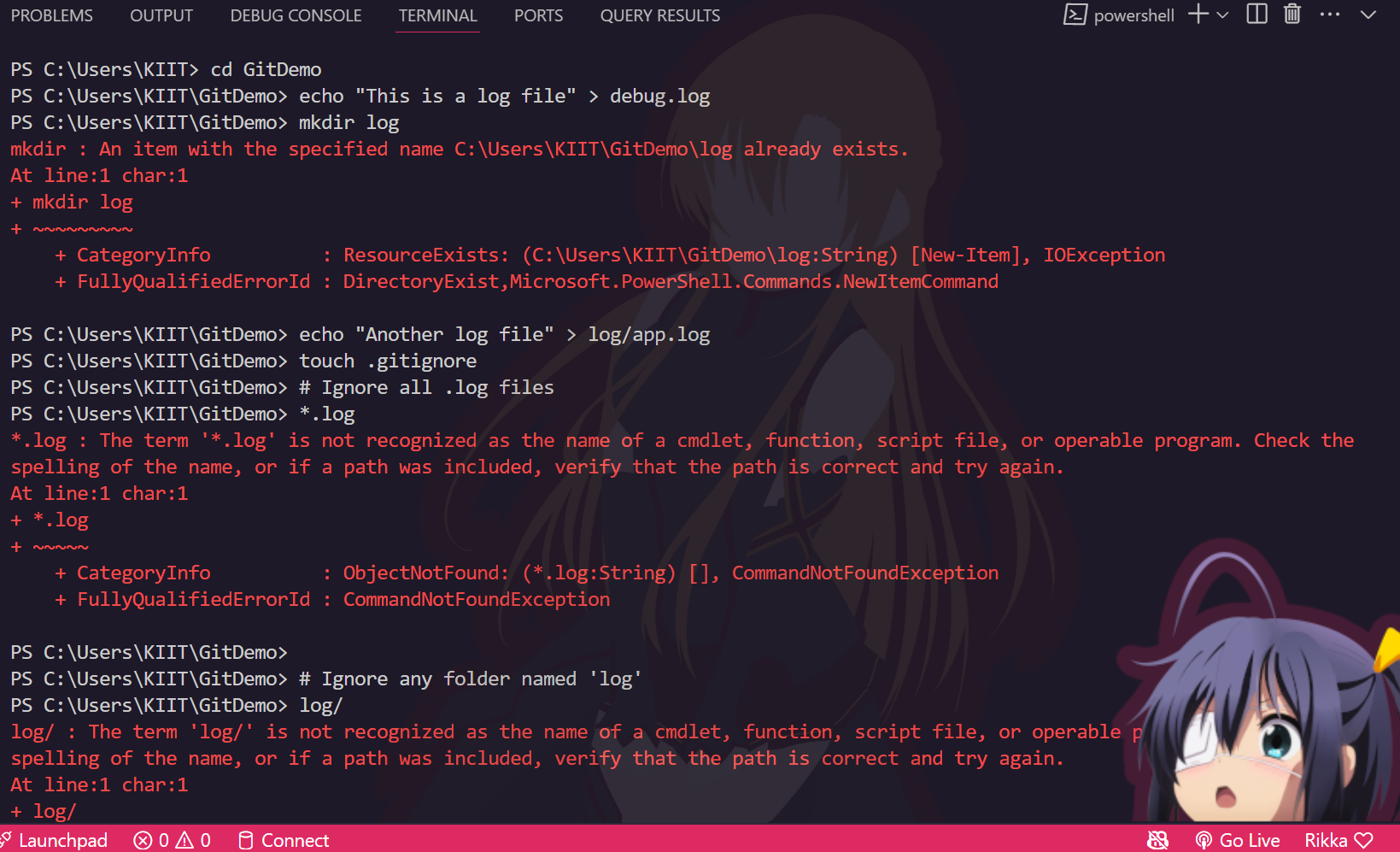


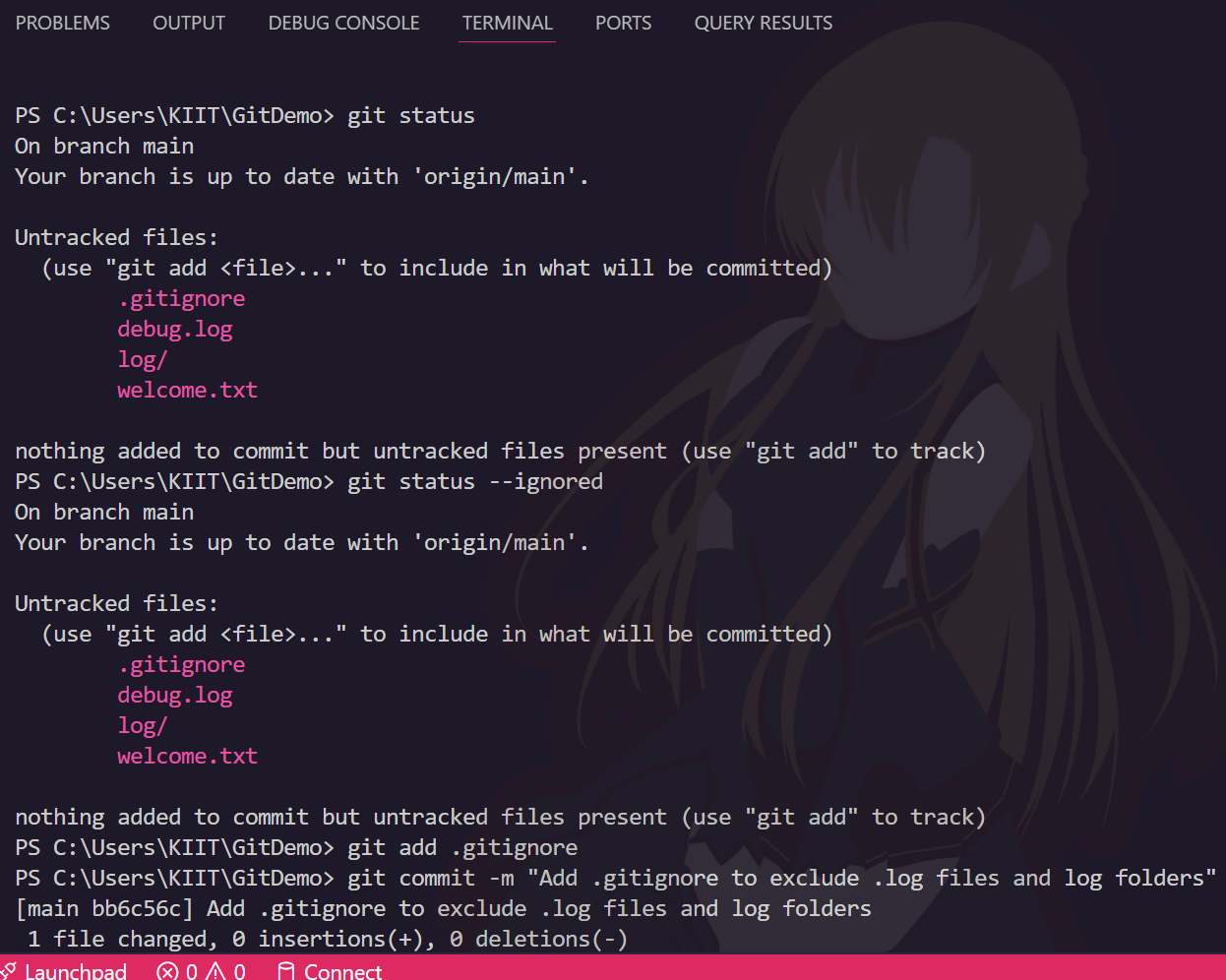
**QUESTION-2**

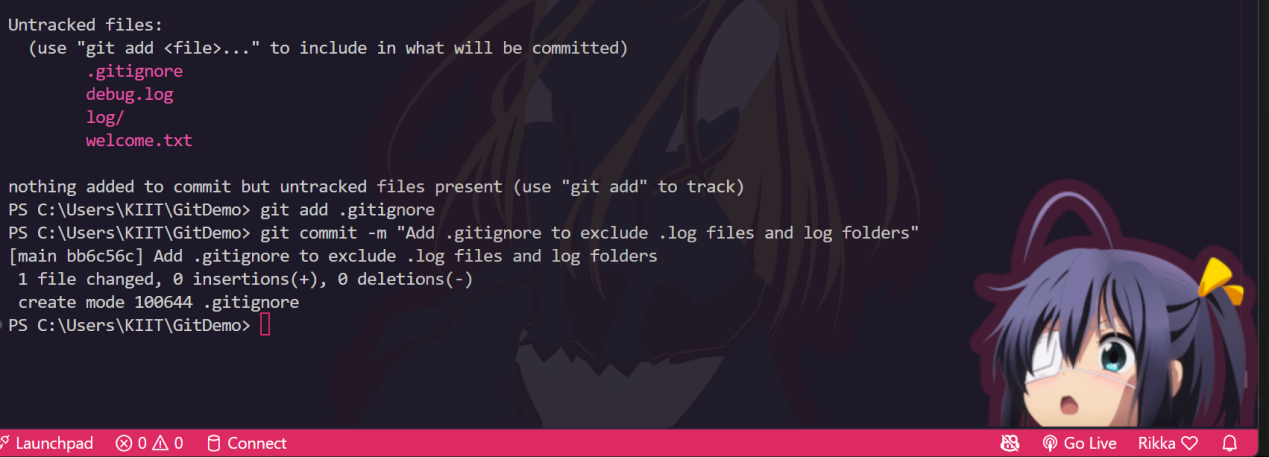
**Implement git ignore command to ignore unwanted files and folders**

1. What is .gitignore?  
   The .gitignore file tells Git which files and folders to ignore (not track or commit)  
   Commonly used to keep log files, temporary files, build artifacts, or personal files out of your repository.  
   The .gitignore file itself is tracked by Git, so everyone using the repository ignores the same files1.  
   2. Steps to Ignore .log Files and log Folders  
   A. Navigate to Your Git Repository  
   cd GitDemo  
     
   B. Create a .log File and a log Folder  
      
   echo "This is a log file" > debug.log  
   mkdir log  
   echo "Another log file" > log/app.log  
     
   C. Create or Edit .gitignore in the Root Directory  
      
   touch .gitignore  
     
   Open .gitignore in your editor and add:  
     
   # Ignore all .log files  
   \*.log  
     
   # Ignore any folder named 'log'  
   log/  
     
   \*.log ignores all files ending with .log anywhere in the repo12.  
   log/ ignores any folder named log in the root (add /log/ for only root, or use \*\*/log/ for any location).  
   D. Save .gitignore and Check Status  
   git status  
     
   You should not see debug.log or log/app.log listed as untracked files.  
   If you see them, they may have already been tracked; see the troubleshooting section below.  
   E. Verify Ignored Files  
   To see which files are ignored:  
   git status --ignored  
     
   F. Commit Your .gitignore  
      
   git add .gitignore  
   git commit -m "Add .gitignore to exclude .log files and log folders"  
     
   Troubleshooting  
   If a file is already tracked by Git, adding it to .gitignore won’t stop tracking it1:  
     
      
   git rm --cached debug.log  
   git rm --cached log/app.log  
     
   Then commit again.  
   Make sure there are no typos in your .gitignore (it is case-sensitive).

**OUTPUT:**

****

****

****

**QUESTION- 3**

**Construct a branch, do some changes in the branch, and merge it with master (or trunk)**

**INPUT AND CODE:**

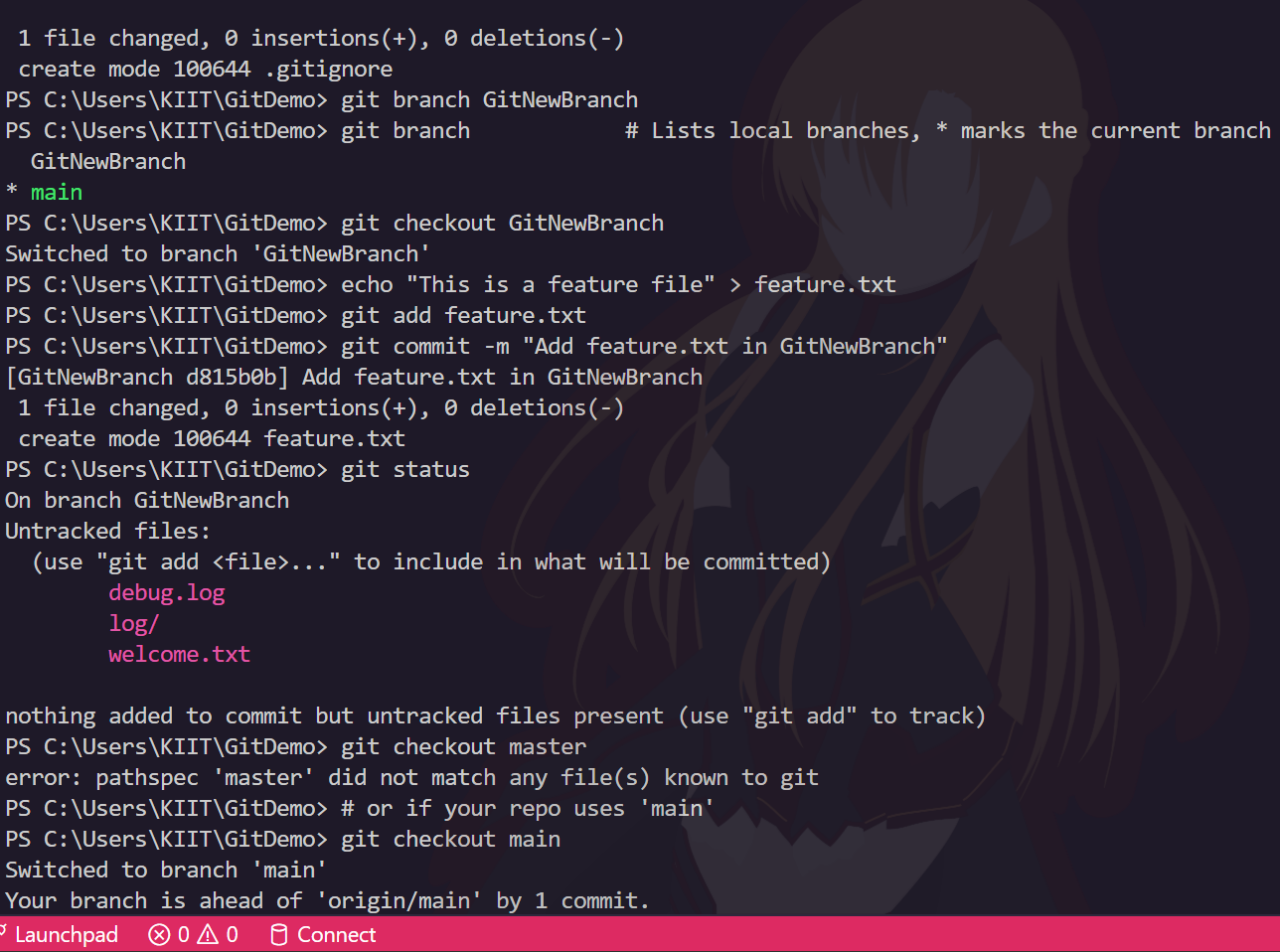
1. Branching  
A. Create a New Branch  
   
git branch GitNewBranch  
  
or (to create and switch in one step):  
  
   
git checkout -b GitNewBranch  
  
B. List All Local and Remote Branches  
   
git branch # Lists local branches, \* marks the current branch  
git branch -a # Lists local and remote branches  
  
C. Switch to the New Branch  
git checkout GitNewBranch  
  
D. Add Files and Make Changes  
  
echo "This is a feature file" > feature.txt  
git add feature.txt  
git commit -m "Add feature.txt in GitNewBranch"  
  
E. Check Status  
   
git status  
  
2. Merging  
A. Switch to Master/Main  
   
git checkout master  
# or if your repo uses 'main'  
git checkout main  
  
B. List Differences Between Branches (Command Line)  
   
git diff master..GitNewBranch  
git diff GitNewBranch  
  
C. Visualize Differences with P4Merge  
First, set up P4Merge as your mergetool (one-time setup):  
  
   
git config --global merge.tool p4merge  
git config --global mergetool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"  
  
(Adjust the path if P4Merge is installed elsewhere.)  
  
To view differences:  
  
   
git difftool master GitNewBranch  
  
D. Merge the Branch into Master/Main  
   
git checkout master   
git merge GitNewBranch  
  
E. Resolve Conflicts with P4Merge (if any)  
   
git mergetool  
  
P4Merge will open to help you resolve conflicts visually.  
After resolving, save and close P4Merge, then continue the merge process in Git.

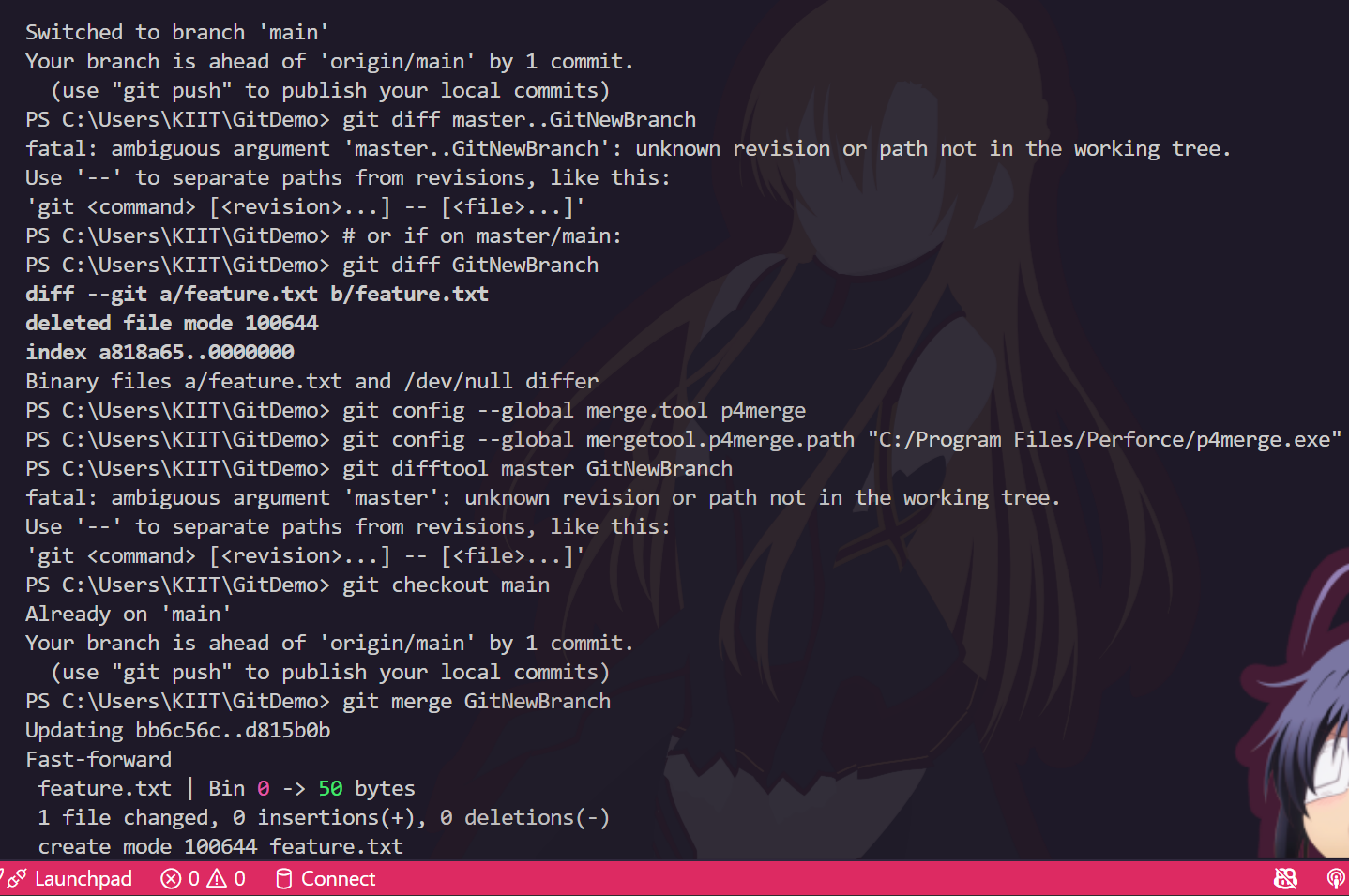
F. View Merge Log  
   
git log --oneline --graph --decorate  
  
This shows a visual history of branches and merges.

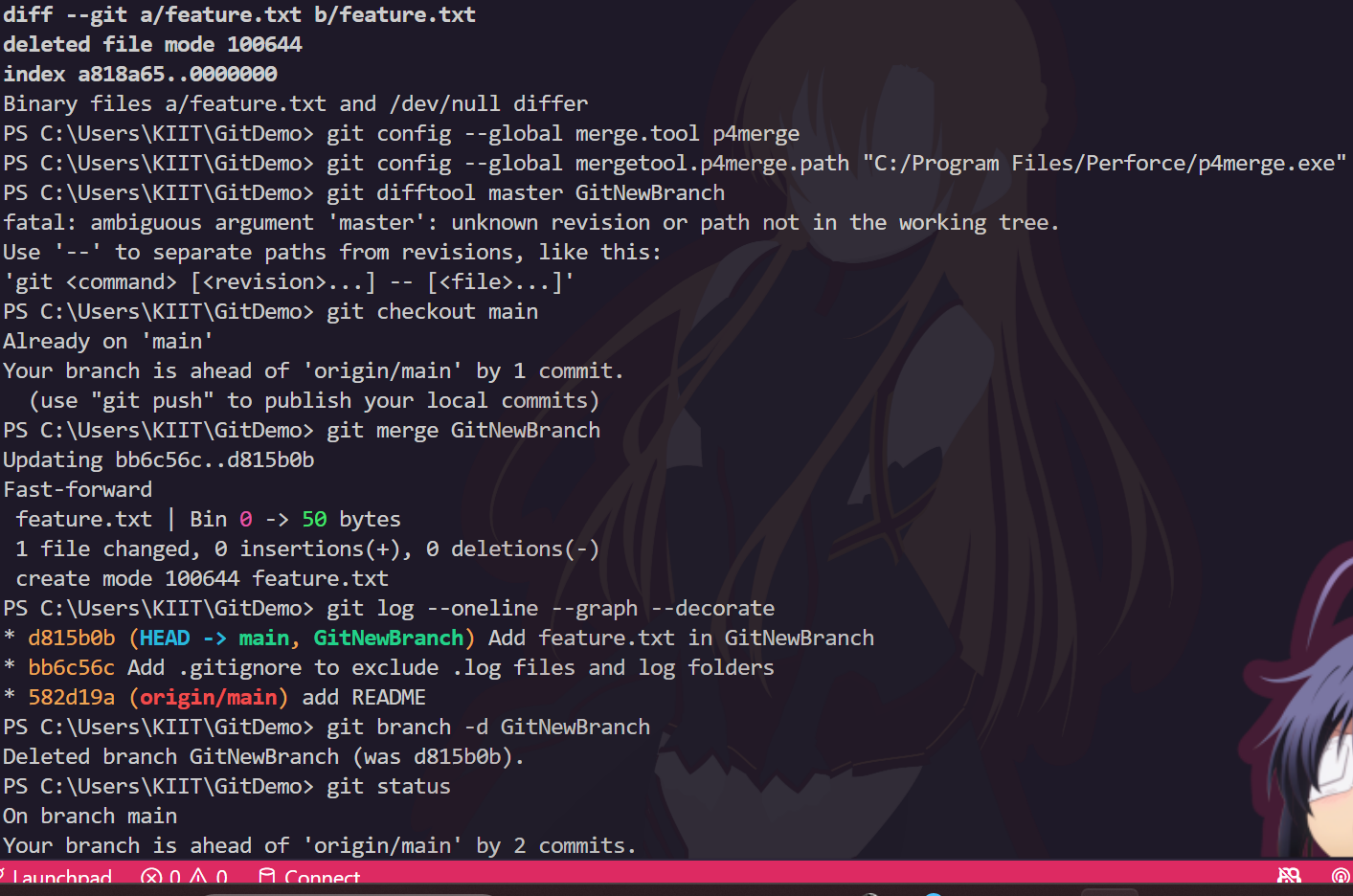
G. Delete the Branch After Merging  
   
git branch -d GitNewBranch

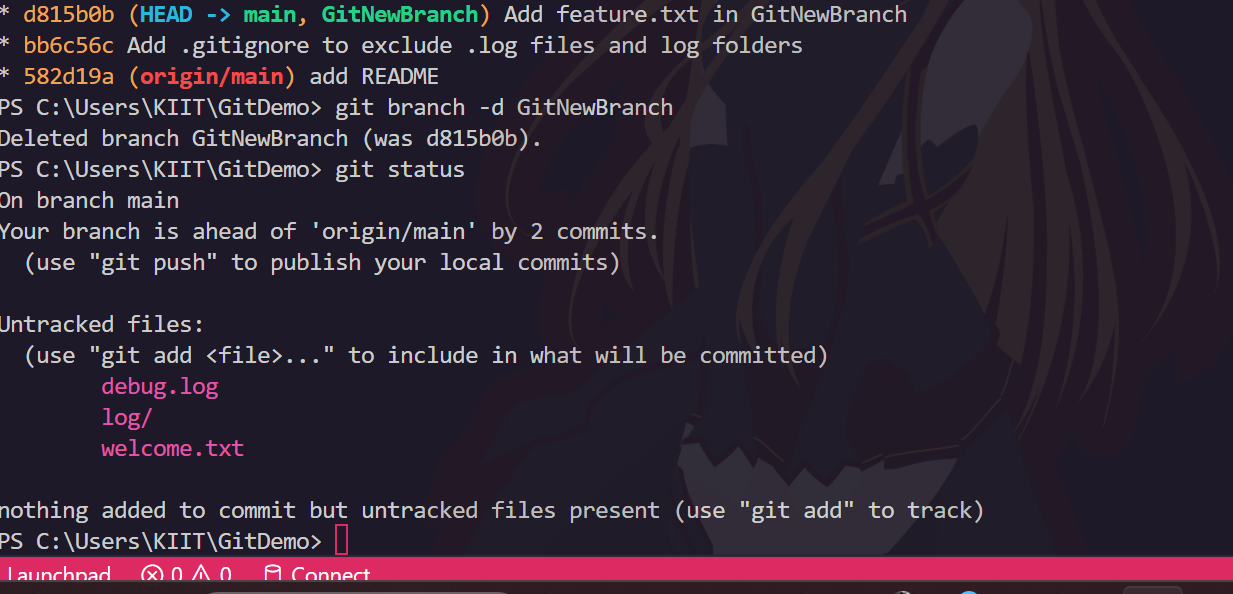
H. Check Status  
git status

**OUTPUT:**

****

****

****

****

**QUESTION-4**

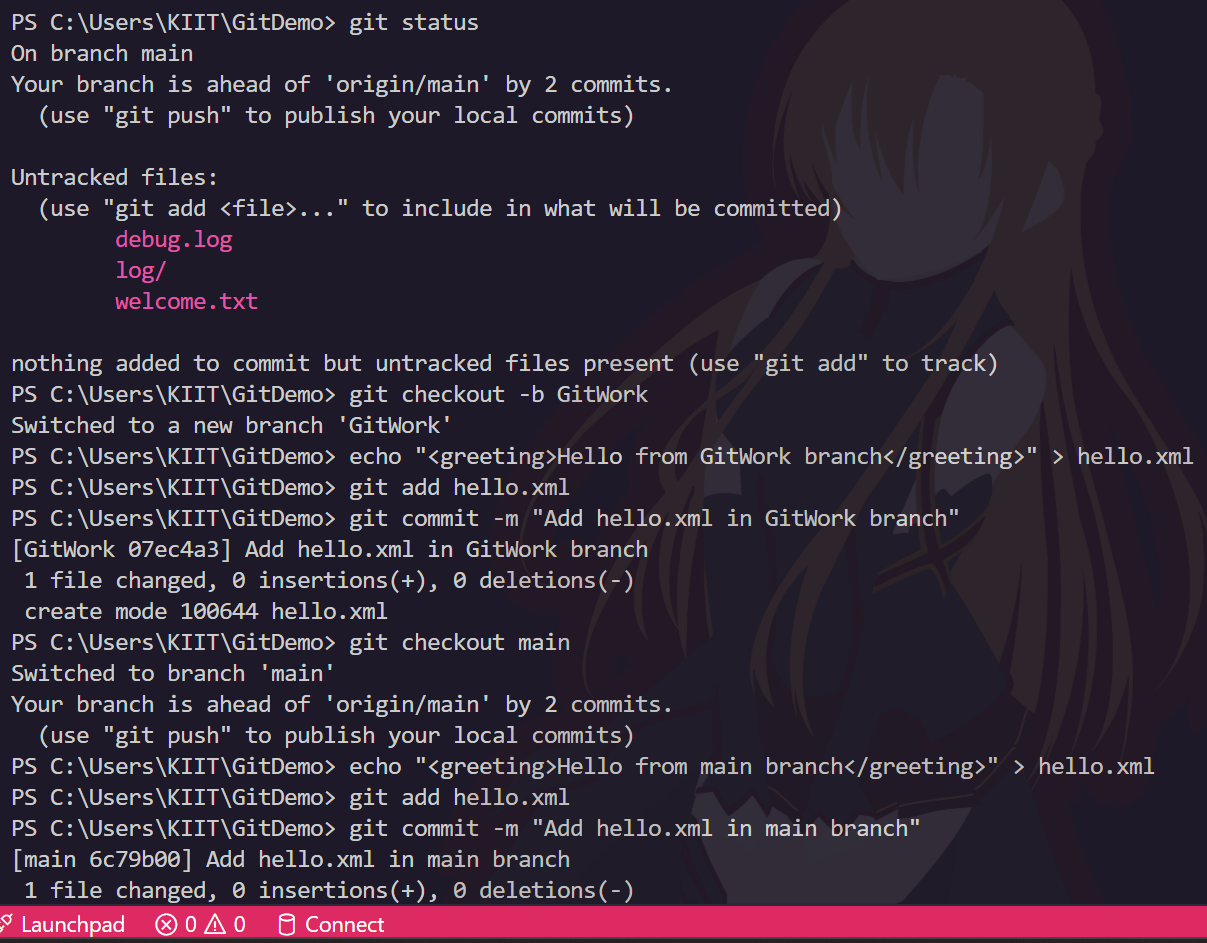
**Implement conflict resolution when multiple users are updating the trunk (or master) in such a way that it results into a conflict with the branch’s modification**.

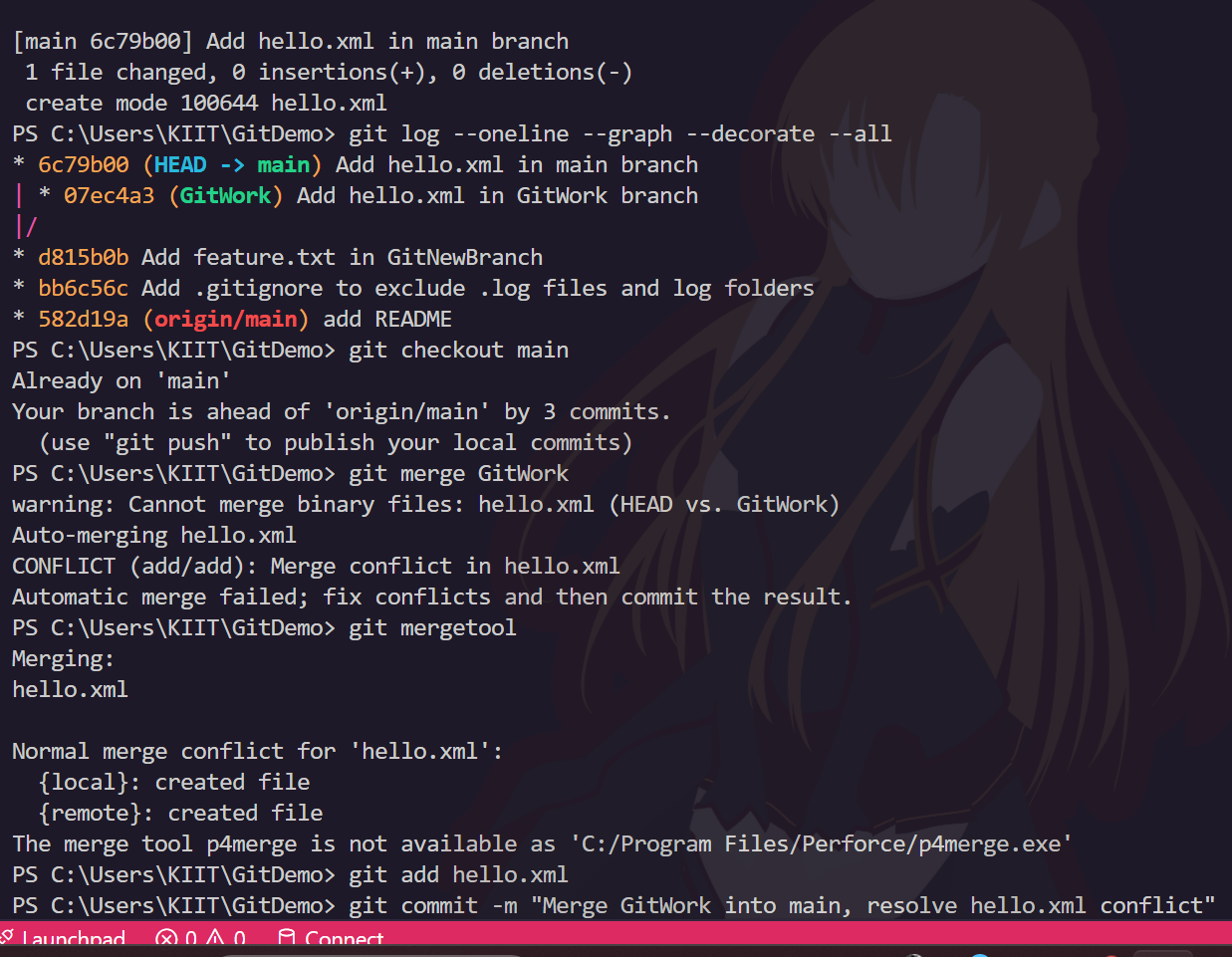
**INPUT AND CODE:**

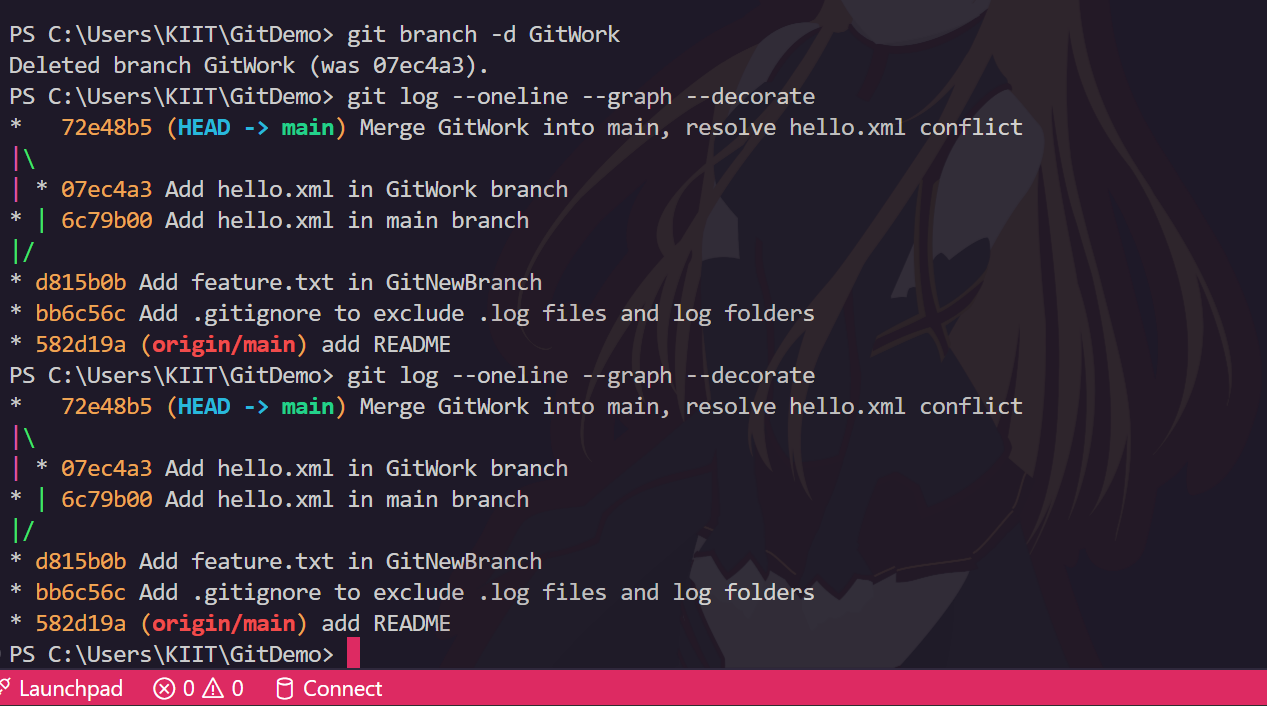
1. Ensure main is Clean  
   
git checkout main  
git status  
  
Confirm: “nothing to commit, working tree clean”.  
2. Create and Switch to a New Branch  
   
git checkout -b GitWork  
  
3. Add and Commit a File on the Branch  
   
echo "<greeting>Hello from GitWork branch</greeting>" > hello.xml  
git add hello.xml  
git commit -m "Add hello.xml in GitWork branch"  
  
4. Switch Back to main and Add a Different Version  
   
git checkout main  
echo "<greeting>Hello from main branch</greeting>" > hello.xml  
git add hello.xml  
git commit -m "Add hello.xml in main branch"  
  
5. View Log and Differences  
   
git log --oneline --graph --decorate --all  
git diff main..GitWork  
  
  
6. Merge the Branch into main (This Will Cause a Conflict)  
   
git checkout main  
git merge GitWork  
  
a conflict in hello.xml.

7.. Resolve the Conflict  
Use P4Merge  
git mergetool  
  
10. Mark as Resolved and Commit  
   
git add hello.xml  
git commit -m "Merge GitWork into main, resolve hello.xml conflict"  
  
11. Clean Up: Delete the Merged Branch  
   
git branch -d GitWork  
  
12. View Final Log  
   
git log --oneline --graph --decorate

**OUTPUT:**

****

****

****

**QUESTION - 5**

**Execute steps involving clean up and push back to remote Git.**

1. Verify if main is in a Clean State  
Check for uncommitted changes:  
  
git status

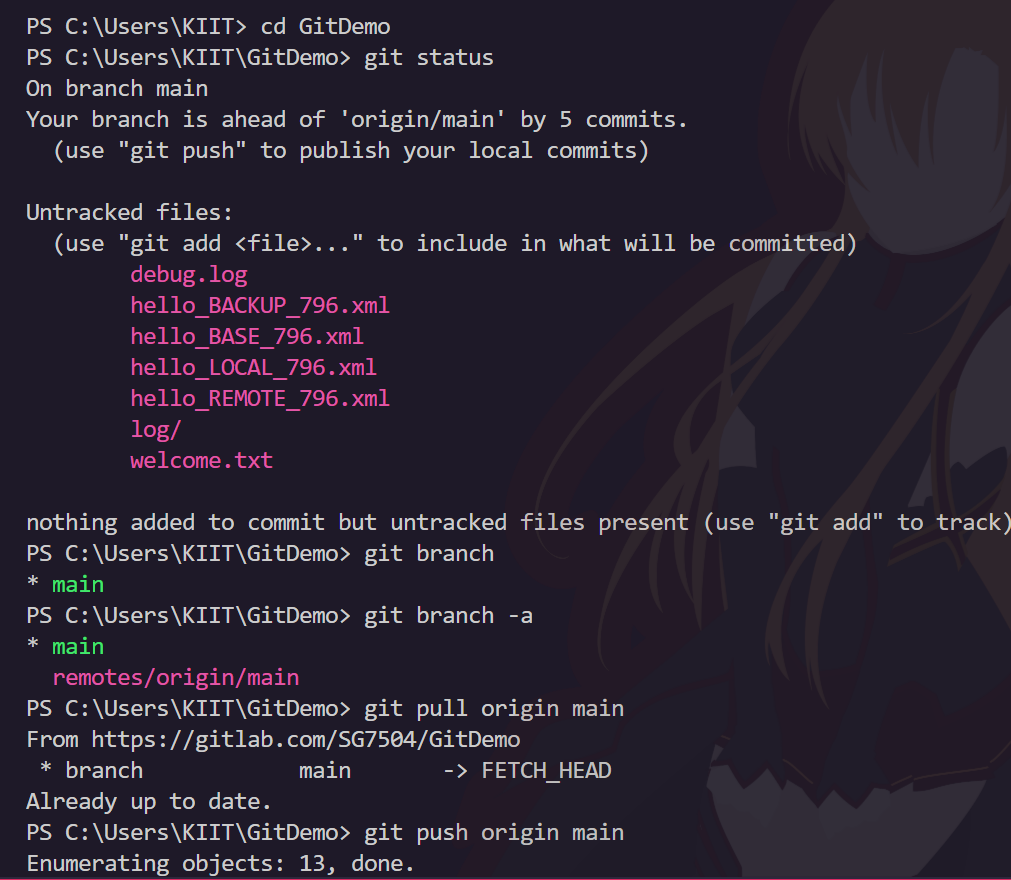
Or for a machine-readable check:  
git status --porcelain  
  
No output means your repo is clean2.  
2. List All Available Branches  
Show local branches:  
  
git branch  
  
Show all local and remote branches:  
  
git branch -a  
  
Remote branches are shown as remotes/origin/branchname

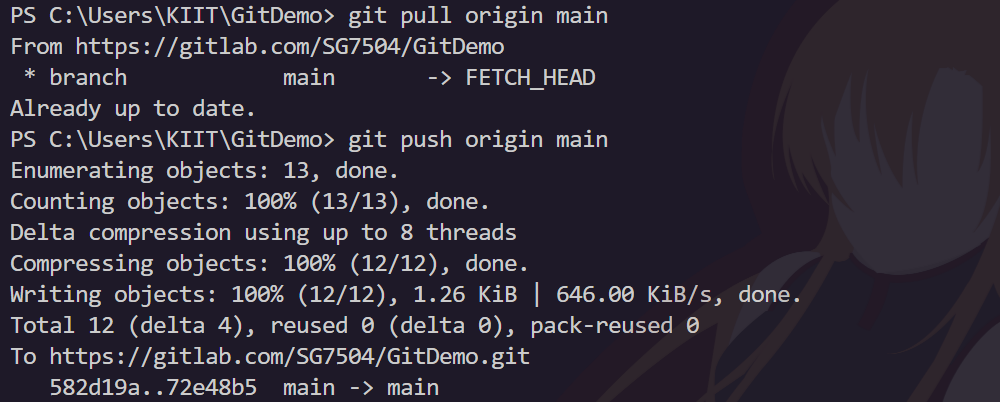
3. Pull the Remote Repository to main  
Update your local branch with remote changes:  
  
git pull origin main  
  
This downloads and merges any changes from the remote main branch into your local main

4. Push Pending Changes to Remote

Push your local commits to the remote repository:  
  
git push origin main  
  
This uploads local commits to the remote main branch

**OUTPUT:**

****

****