1. **Git Hands-on Lab Solution**

# Step 1: Setup Git Configuration

Command to check Git version:

git --version

Output:  
git version 2.49.0.windows.1

Set Git username and email:

git config --global user.name "SANDHYAA"  
git config --global user.email "sandhyaa6804@gmail.com"

Verify configuration:

git config --global --list  
user.name=SANDHYAA  
user.email=sandhyaa6804@gmail.com

# Step 2: Initialize Git Repository

Create a new folder and move into it:

mkdir GitDemo  
cd GitDemo

Initialize repository:

git init

Output:  
Initialized empty Git repository in C:/Users/Asus/GitDemo/.git/

# Step 3: Create and Add File to Repository

Create a welcome.txt file:

echo "Welcome to Git Demo" > welcome.txt

Check if file exists:

dir  
Output: welcome.txt listed

View file content:

type welcome.txt  
Output: Welcome to Git Demo

Check status:

git status  
Output: welcome.txt shown as untracked

Stage the file:

git add welcome.txt

Commit the file:

git commit -m "First commit - Added welcome.txt"

# Step 4: Connect to GitHub and Push

Create a remote repository named GitDemo on GitHub (without README).

Add remote origin:

git remote add origin https://github.com/Sandhyaa68/GitDemo.git

Push to GitHub:

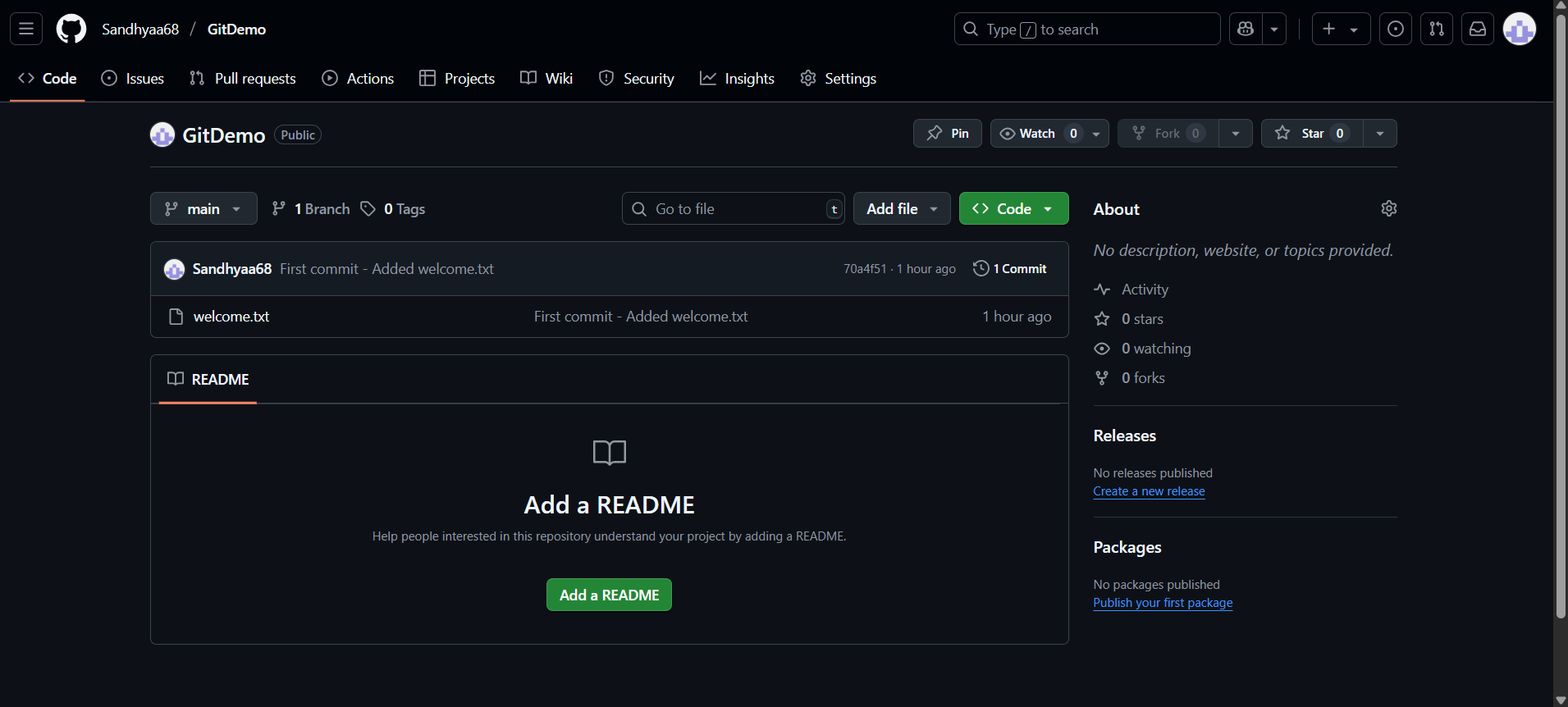
git branch -M main  
git push -u origin main

After authentication with Git Credential Manager, the file was successfully pushed to GitHub.

# Step 5: Verification

Visited the repository URL: https://github.com/Sandhyaa68/GitDemo  
Confirmed that welcome.txt was present with the correct commit message and content.

# OUTPUT:



**2. Git Hands-on Lab Solution**

This document contains the complete step-by-step solution for the Git Hands-on Lab (Git-HOL). It includes all commands executed, outputs obtained, and the final verification on GitHub.

# Step 1: Setup Git Configuration

Command to check Git version:

git --version

Output:  
git version 2.49.0.windows.1

Set Git username and email:

git config --global user.name "SANDHYAA"  
git config --global user.email "sandhyaa6804@gmail.com"

Verify configuration:

git config --global --list  
user.name=SANDHYAA  
user.email=sandhyaa6804@gmail.com

# Step 2: Initialize Git Repository

Create a new folder and move into it:

mkdir GitDemo  
cd GitDemo

Initialize repository:

git init

Output:  
Initialized empty Git repository in C:/Users/Asus/GitDemo/.git/

# Step 3: Create and Add File to Repository

Create a welcome.txt file:

echo "Welcome to Git Demo" > welcome.txt

Check if file exists:

dir  
Output: welcome.txt listed

View file content:

type welcome.txt  
Output: Welcome to Git Demo

Check status:

git status  
Output: welcome.txt shown as untracked

Stage the file:

git add welcome.txt

Commit the file:

git commit -m "First commit - Added welcome.txt"

# Step 4: Connect to GitHub and Push

Create a remote repository named GitDemo on GitHub (without README).

Add remote origin:

git remote add origin https://github.com/Sandhyaa68/GitDemo.git

Push to GitHub:

git branch -M main  
git push -u origin main

After authentication with Git Credential Manager, the file was successfully pushed to GitHub.

# Step 5: Verification

Visited the repository URL: https://github.com/Sandhyaa68/GitDemo  
Confirmed that welcome.txt was present with the correct commit message and content.

# Step 6: Implementing .gitignore to Ignore Log Files

In this step, we configure Git to ignore all '.log' files and the 'log/' folder, so that temporary log files do not appear in git status or get committed to the repository.

Create or overwrite the .gitignore file with the following rules:

echo "\*.log" > .gitignore  
echo "log/" >> .gitignore

The rules mean:

- '\*.log': Ignore all files ending in .log anywhere in the repository.

- 'log/': Ignore the entire log folder and its contents.

Verify that .gitignore is working:

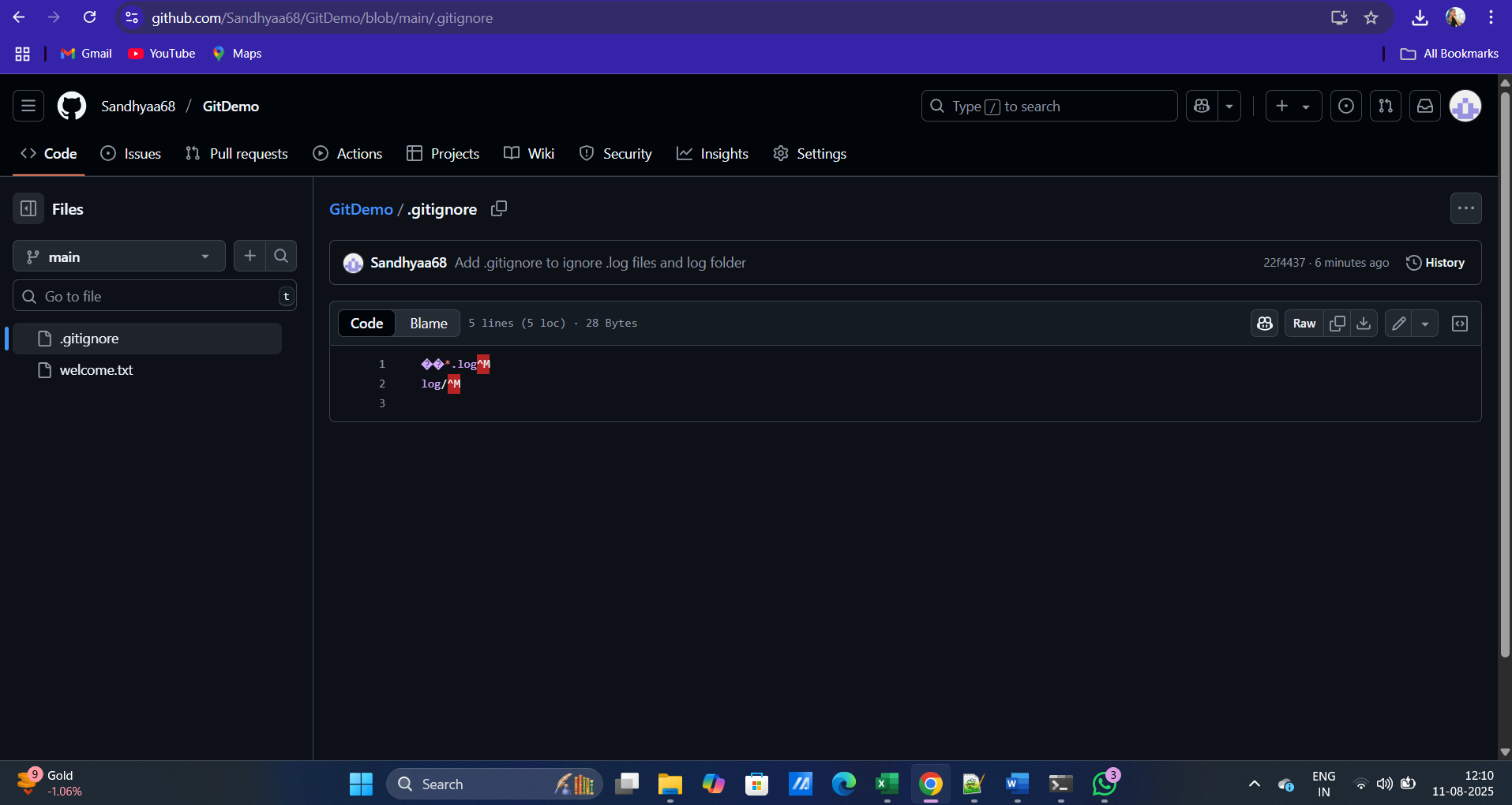
git status  
git status --ignored  
git check-ignore -v error.log  
git check-ignore -v log\app.log

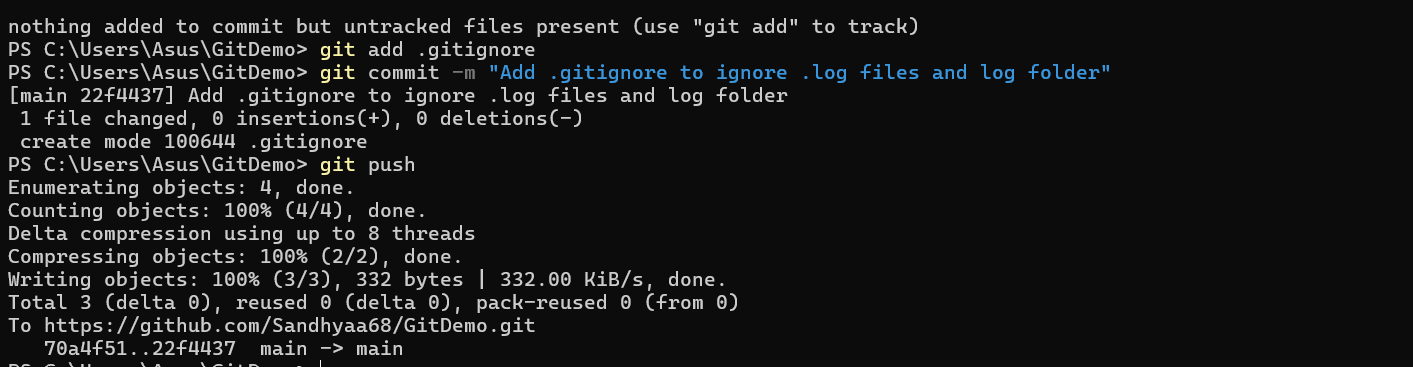
Expected output:  
 - git status: Does not show error.log or log/ under untracked files.  
 - git status --ignored: Shows them under 'Ignored files' section.  
 - git check-ignore -v: Shows the matching rule in .gitignore that causes the file to be ignored.

To save .gitignore in the repository:

git add .gitignore  
git commit -m "Add .gitignore to ignore .log files and log folder"  
git push

# OUTPUT:





**3. Git Hands-on Lab Solution**

# Step 1: Create or Clone a Repository

Command:  
git init  
  
Description:  
Initializes a new Git repository in the current folder.

# Step 2: Add Remote Repository URL

Command:  
git remote add origin https://github.com/Sandhyaa68/GitDemo.git  
  
Description:  
Links the local repository to the remote GitHub repository.

# Step 3: Create Files

Commands:  
echo "Welcome to Git" > welcome.txt  
echo "Error log" > error.log  
mkdir log  
echo "test" > log/app.log  
  
Description:  
Creates a text file, an error log, and a log folder with a sample file.

# Step 4: Create .gitignore

Command:  
echo "\*.log  
log/" > .gitignore  
  
Description:  
Specifies files and folders to be ignored by Git (e.g., .log files and log folder).

# Step 5: Stage and Commit Files

Commands:  
git add .gitignore welcome.txt  
git commit -m "Initial commit with welcome.txt and .gitignore"  
  
Description:  
Stages the selected files and commits them with a message.

# Step 6: Push Changes to Remote

Command:  
git push -u origin main  
  
Description:  
Pushes the committed changes from the local repository to the remote GitHub repository.

# Step 7: Create and Switch to a New Branch

Commands:  
git branch GitNewBranch  
git checkout GitNewBranch  
  
Description:  
Creates a new branch named GitNewBranch and switches to it.

# Step 8: Make Changes in New Branch

Command:  
echo "This is a change in GitNewBranch" > branchfile.txt  
git add branchfile.txt  
git commit -m "Add branchfile.txt in GitNewBranch"  
  
Description:  
Creates a new file and commits it to the GitNewBranch branch.

# Step 9: Compare Branches with P4Merge

Commands:  
git config --global merge.tool p4merge  
git config --global mergetool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"  
git config --global diff.tool p4merge  
git difftool main GitNewBranch  
  
Description:  
Opens P4Merge to visually compare differences between main and GitNewBranch.

# Step 10: Merge GitNewBranch into main

Commands:  
git checkout main  
git merge GitNewBranch  
  
Description:  
Merges changes from GitNewBranch into the main branch.

# Step 11: Push Final Changes to GitHub

Command:  
git push origin main  
  
Description:  
Uploads the merged content to the GitHub repository.

# OUTPUT:

