**WEEK 8 MANDATORY HANDS ON BY (6377625)**

GIT- HOL

## **Fileno 1:**

## **Objectives**

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

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

* Setup your machine with Git Configuration
* Integrate notepad++.exe to Git and make it a default editor
* Add a file to source code repository

**Step 1 – Git Configuration**

**1. Verify Git installation**

git --version

**Output:**

git version 2.xx.x.windows.x

**2. Configure your Git username and email**

git config --global user.name "Uthayabi"

git config --global user.email "uthayabi77@gmail.com"

**3. Verify configuration**

git config –list

**Output:**

user.name=Uthayabi

user.email= uthayabi77@gmail.com

**Step 2 – Integrate Notepad++ as Default Editor**

**1. Check if Notepad++ opens**

notepad++

**2. Create alias for Notepad++**

alias np='notepad++'

**3. Configure Git to use Notepad++**

git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -nosession"

**4. Verify default editor**

git config --global -e

**Step 3 – Create GitDemo Project & Add File**

**1. Create a new directory and initialize Git**

mkdir GitDemo

cd GitDemo

git init

**Expected Output:**

Initialized empty Git repository in C:/path/to/GitDemo/.git/

**2. Verify hidden .git folder exists**

ls -a

**Expected Output:**

. .. .git

**3. Create a file and add content**

echo "Welcome to Git Demo" > welcome.txt

**4. Verify file creation**

ls

**Expected Output:**

welcome.txt

**5. View file content**

cat welcome.txt

**Expected Output:**

Welcome to Git Demo

**6. Check status**

git status

**Output:**

Untracked files:

(use "git add <file>..." to include in what will be committed)

welcome.txt

**7. Stage the file**

git add welcome.txt

**8. Commit the file (multi-line message in Notepad++)**

git commit

**9. Verify status**

git status

**Output:**

nothing to commit, working tree clean

**Step 4 – Push to Remote Repository**

**1. Link local repo to remote GitLab**

git remote add origin https://gitlab.com/<uthayabi77>/GitDemo.git

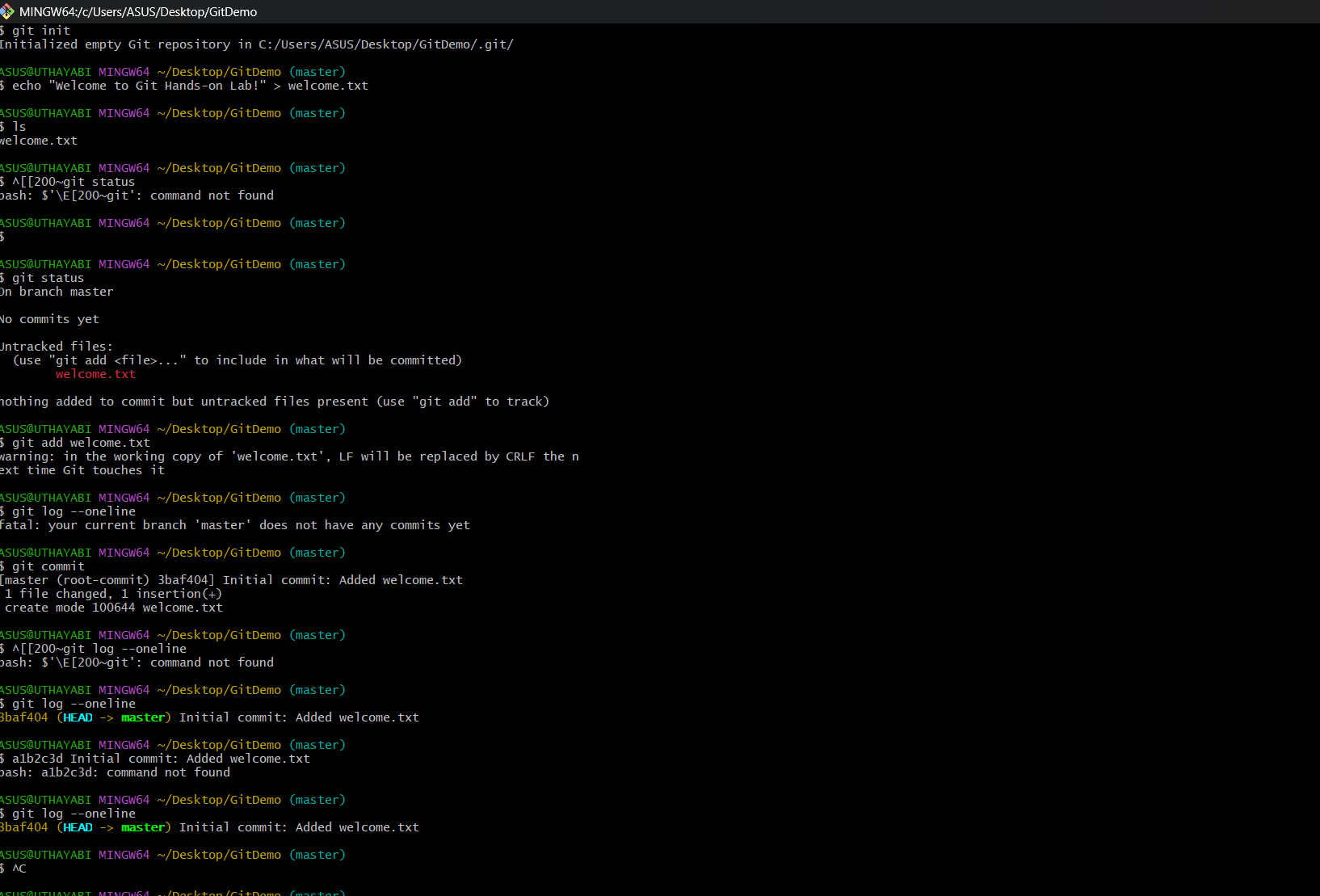
**2. Pull remote (if any changes)**

git pull origin master --allow-unrelated-histories

**3. Push local repo to remote**

git push -u origin master

**Output:**



**File no : 2**

## **Objectives**

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

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

* Setup your machine with Git Configuration
* Integrate notepad++.exe to Git and make it a default editor
* Add a file to source code repository

**1.Open Git Bash and navigate to your working directory**

cd ~/Desktop/GitDemo

**2. Initialize Git (if not already initialized)**

git init

**3. Create a .log file and a log folder**

touch test.log

mkdir log

touch log/file1.txt

**4. Create or edit the .gitignore file**

nano .gitignore

Add the following lines inside .gitignore:

\*.log

log/

(Press CTRL + O → Enter → CTRL + X to save and exit if using nano)

**5. Check Git status**

git status

You should see that .log files and the log folder are **not tracked**.

**6. Add and commit changes (only tracked files will be committed)**

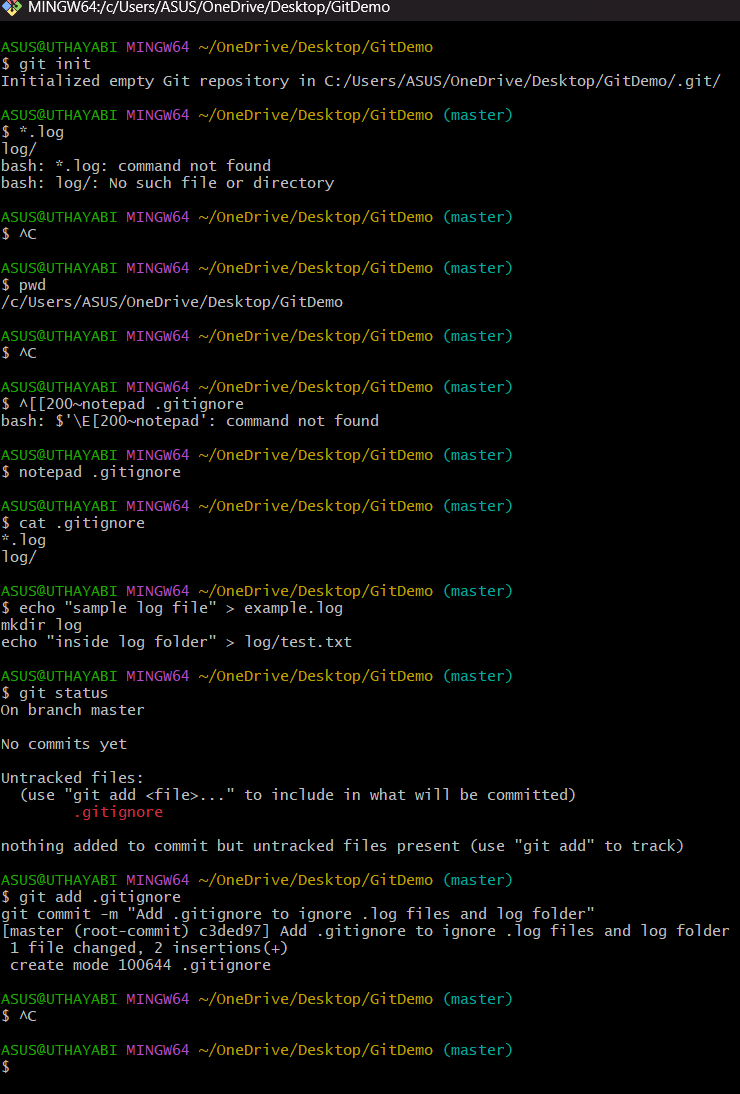
git add .

git commit -m "Added gitignore to ignore log files and log folder"

**7. Verify ignored files again**

git status

**Output:**

****

**File no : 3**

**Objectives**

Explain git ignore

Explain how to ignore unwanted files using git ignore

* Setting up Git environment
* Integrate Notepad++ as a default editor
* A Git repository in the local system and a remote repository in GitLab

**Steps :**

1. Open Git Bash in your working directory

cd ~/Desktop/GitDemo

2. Create a .log file and a log folder

touch sample.log

mkdir log

touch log/test.txt

3. Create a .gitignore file

touch .gitignore

4. Open .gitignore file in Notepad++

notepad++ .gitignore

5. Add the following entries to .gitignore

\*.log

log/

Explanation:

* \*.log → Ignores all .log files in the repository
* log/ → Ignores the entire log folder

6. Add .gitignore file to staging

git add .gitignore

7. Commit changes

git commit -m "Added .gitignore to ignore .log files and log folder"

8. Verify status

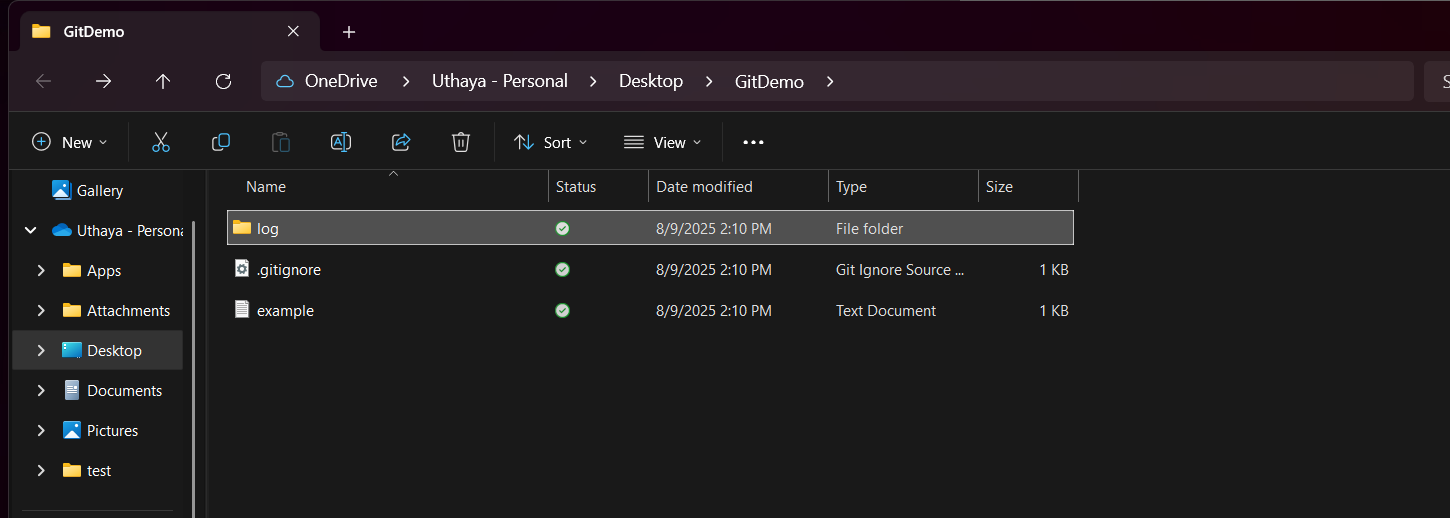
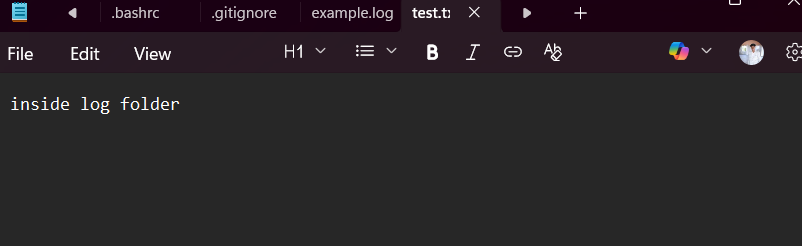
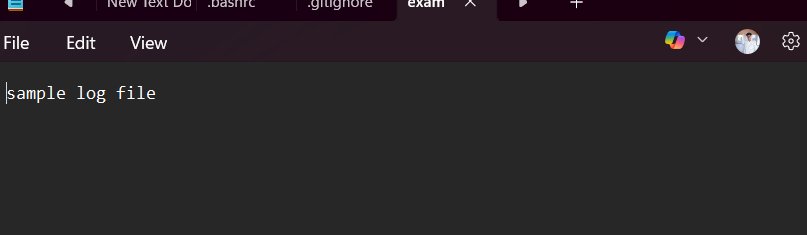
git status

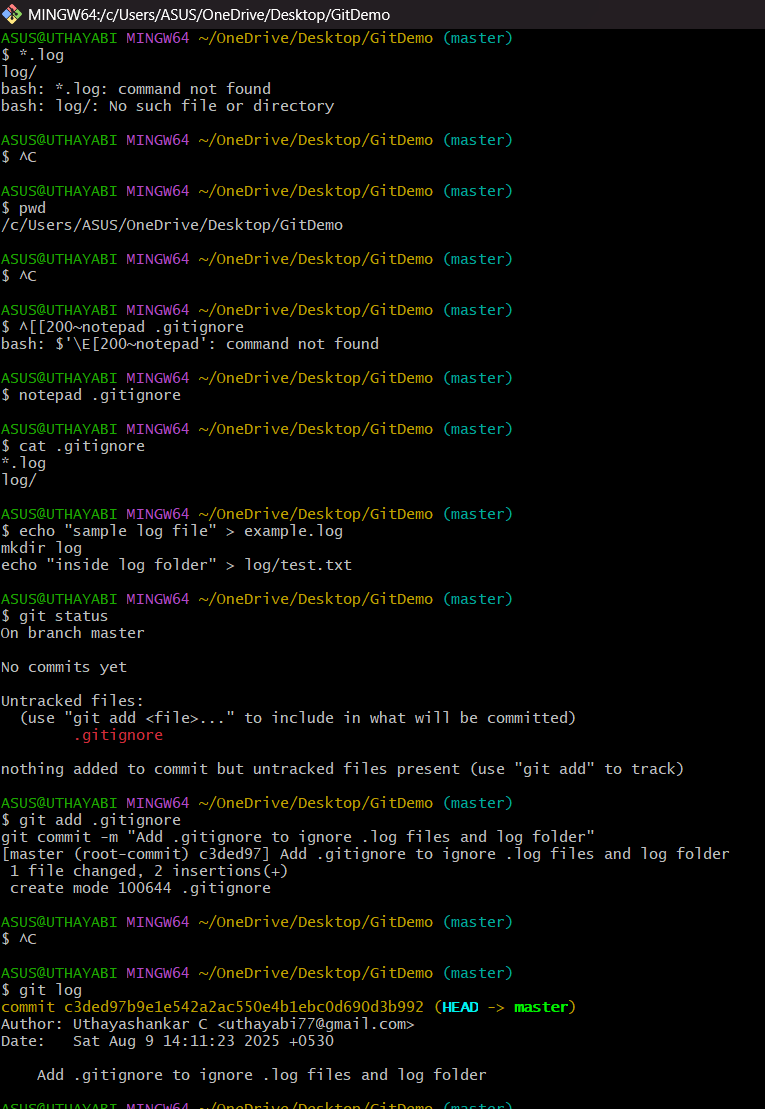
You should see .log files and log folder not being tracked.

9. Push changes to remote repository

git push origin master

**OUTPUT:**





**File no : 4**

**Objectives**

* Explain how to resolve the conflict during merge.
* Hands-on ID: “Git-T03-HOL\_001

Steps:

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

git status

**2. Create a branch “GitWork” and switch to it**

git checkout -b GitWork

**3. Add a file “hello.xml”**

echo "<message>Hello from GitWork branch</message>" > hello.xml

**4. Update the content of “hello.xml”**

echo "<message>Updated content from GitWork branch</message>" > hello.xml

git status

**5. Commit the changes in the branch**

git add hello.xml

git commit -m "Added hello.xml in GitWork branch"

**6. Switch back to master**

git checkout master

**7. Add a different version of “hello.xml” in master**

echo "<message>Hello from master branch</message>" > hello.xml

**8. Commit the changes in master**

git add hello.xml

git commit -m "Added hello.xml in master branch"

**9. Observe the commit history**

git log --oneline --graph --decorate --all

**10. Check differences between master and branch**

git diff GitWork

**11. (Optional) Use P4Merge tool for better visualization**

p4merge

**12. Merge the branch into master**

git merge GitWork

At this step, a **merge conflict** will occur because both master and GitWork modified hello.xml differently.

**13. Observe Git conflict markup in hello.xml**  
Open the file and you will see something like:

php-template

Copy code

<<<<<<< HEAD

<message>Hello from master branch</message>

=======

<message>Updated content from GitWork branch</message>

>>>>>>> GitWork

**14. Resolve the conflict using a 3-way merge tool or manually**

* Keep the desired content and remove the conflict markers (<<<<<<<, =======, >>>>>>>).

**15. Mark the conflict as resolved and commit**

git add hello.xml

git commit -m "Resolved merge conflict between master and GitWork"

**16. Check Git status**

git status

**17. Add backup file to .gitignore**

echo "\*.orig" >> .gitignore

git add .gitignore

git commit -m "Added .gitignore to ignore backup files"

**18. List out all available branches**

git branch

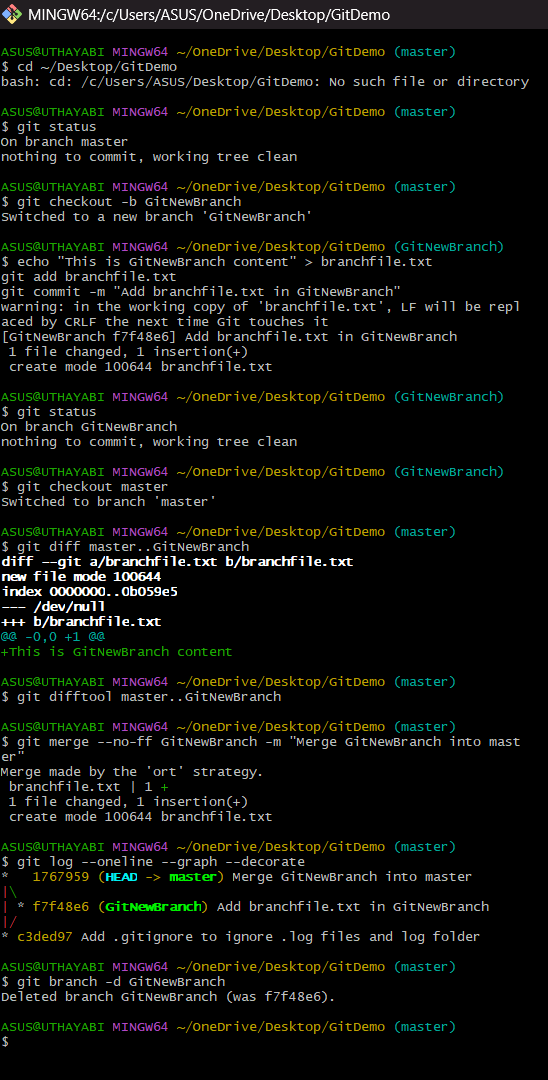
**19. Delete the merged branch**

git branch -d GitWork

**20. Observe final commit history**

git log --oneline --graph –decorate

**Output:**



**File no : 4**

**Objectives**

**Hands-on ID:** Git-T03-HOL\_002  
**Username:** uthayasankar  
**Email:** uthayabi77@gmail.com

Step:

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

git status

* If it shows **"working tree clean"**, your branch is already clean.
* If there are untracked or modified files, either commit them or discard:

git add .

git commit -m "Final commit before cleanup"

Or to discard changes:

git reset --hard

git clean -fd

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

git branch -a

* -a lists both local and remote branches.

**3. Pull the remote git repository to the master**

git checkout master

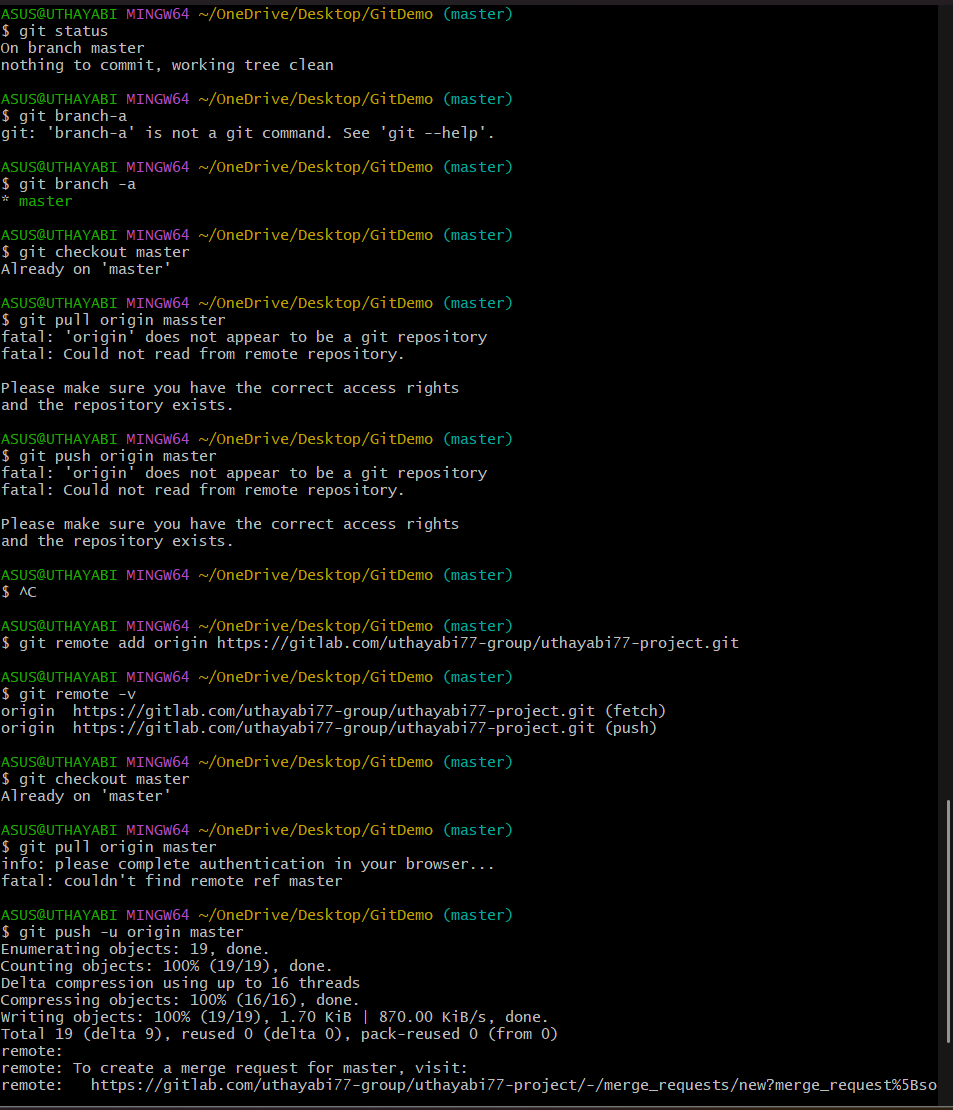
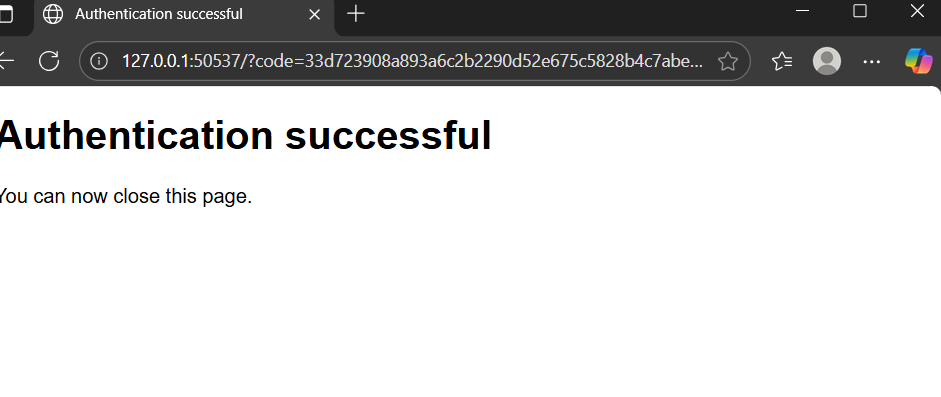
git pull origin master

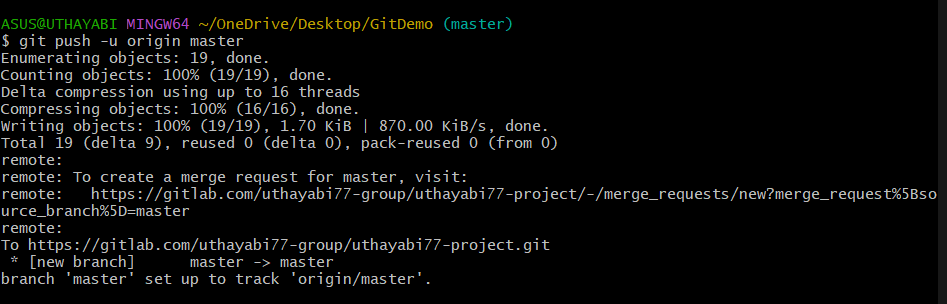
* This ensures your local master branch is updated with the latest remote changes.

**4. Push the changes from Git-T03-HOL\_002 to remote repository**

git push origin master

* This will push all local commits (from the previous lab “Git-T03-HOL\_002”) to the remote master branch.

**OUTPUT:**

****

Completed by :

6377625