**Git-HOL-1**

**Step 1: Git Installation and Configuration**

1. Check Git version:

git –version

Checks if Git is installed.

2. Configure username and email:

git config --global user.name "Your Name"

git config --global user.email [your.email@example.com](mailto:your.email@example.com)

Sets your Git identity globally.

3. Verify configuration:

git config --list

Shows configured details*.*

**Step 2: Set Notepad as Default Git Editor**

1. Set Notepad as editor:

git config --global core.editor "notepad"

Makes Notepad the default editor for Git commits*.*

1. Verify editor:

git config --global -e

Opens the global Git configuration file in Notepad*.*

**Step 3: Create Local Repository and Add File**

1. Create folder and initialize Git:

mkdir GitDemo

cd GitDemo

git init

Creates a new Git repository.

1. Create a file:

echo "Welcome to Git Demo" > welcome.txt

Adds text to a new file named welcome.txt.

1. Check status:

git status

Shows file status in the repo.

1. Add file to staging:

git add welcome.txt

Stages the file for commit.

1. Commit file:

git commit

Opens Notepad to write a commit message, then saves the commit.

**Step 4: Push to Remote Repository**

1. Add remote repository:

git remote add origin https://github.com/your-username/GitDemo.git

Links local repo to GitHub.

1. Push to remote branch:

git push --set-upstream origin main

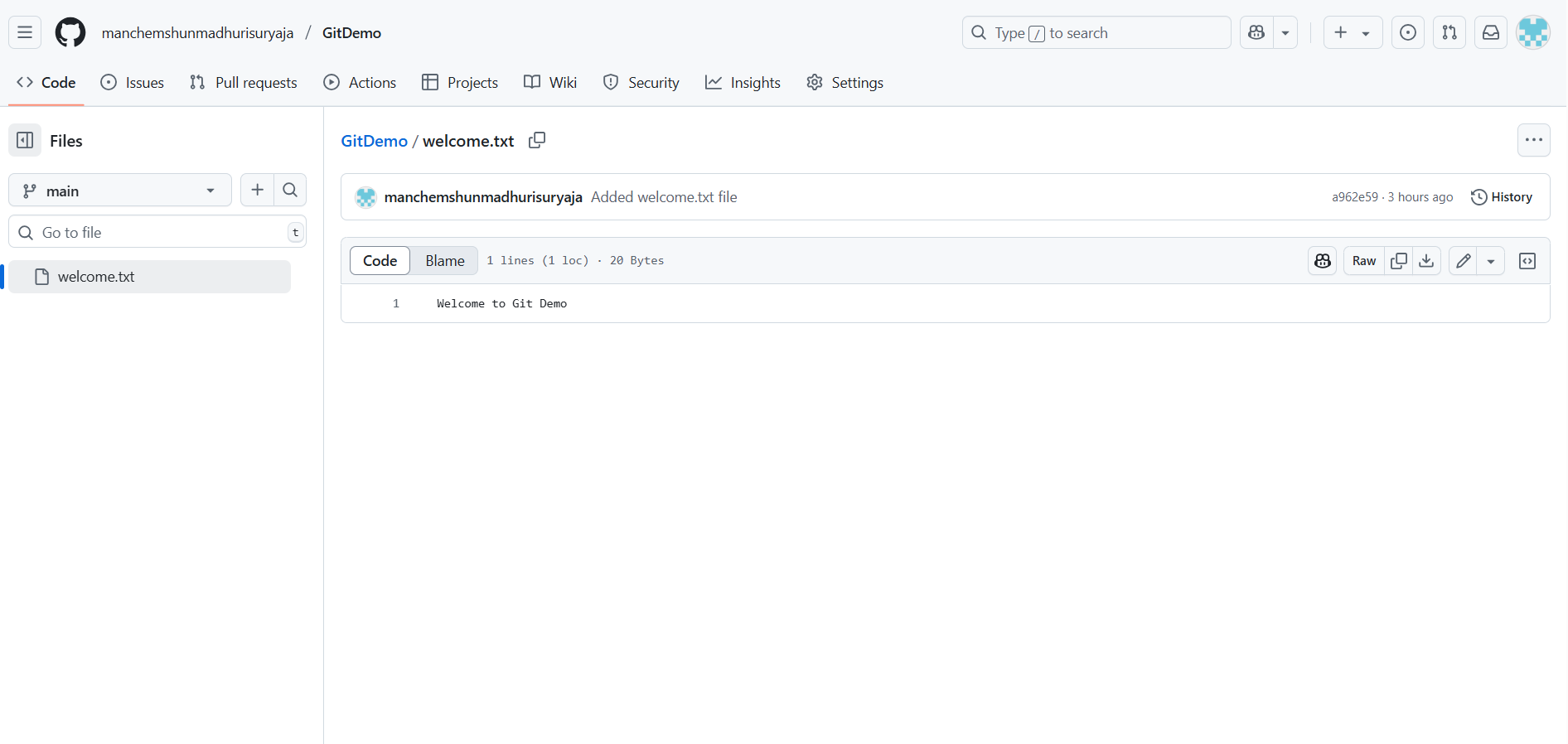
Pushes the local repository to the main branch on GitHub.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.



**Git-HOL-2**

**1. Explain git ignore**

The .gitignore file in Git is used to tell Git which files or directories to ignore and not track in the repository. This helps prevent unnecessary or sensitive files (such as temporary files, log files, system files, or build artifacts) from being added to version control.

**2. Explain how to ignore unwanted files using git ignore**

To ignore unwanted files in a Git repository:

1. Create a file named .gitignore in the root of your repository.
2. Add patterns of files or folders you want Git to ignore.

\*.log

/node\_modules/

temp/

3. Save the .gitignore file and commit it:

git add .gitignore

git commit -m "Added .gitignore file"

4. Git will now ignore these files in future commits.

**Git Hands-On Lab: Implement .gitignore**

**Step 1: Create Files to Ignore**

Create a .log file and a log folder inside the local Git working directory:

echo "This is a log file" > error.log

mkdir log

echo "Log folder file" > log/info.txt

**Step 2: Create .gitignore File**

Create a .gitignore file and add the ignore patterns:

notepad .gitignore

Add the following lines inside .gitignore:

\*.log

log/

Save and close the file.

**Step 3: Stage and Commit .gitignore**

Check status:

git status

Stage .gitignore:

git add .gitignore

Commit:

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

**Step 4: Verify Ignored Files**

Run:

git status

The .log file and log folder should not appear in the output because they are ignored.

**Step 5: Push Changes to Remote Repository**

If not linked, add the remote:

git remote add origin <remote-repo-URL>

Push the changes:

git push --set-upstream origin main

Successfully implemented .gitignore to ignore .log files and the log folder in the Git repository.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Git-HOL-3**

**1.Explain Branching and Merging**

* Branching in Git allows you to create separate lines of development within a repository. Each branch can have its own changes without affecting the main codebase.  
  Example:
* git branch feature1
* git checkout feature1

This creates and switches to a new branch named feature1.

* Merging combines changes from one branch into another. After finishing work in a feature branch, you merge it into the main branch:
* git checkout main
* git merge feature1

This integrates changes from feature1 into main.

**2.Explain Creating a Branch in GitLab**

1. Go to your GitLab project repository.
2. Click on Repository → Branches.
3. Click New Branch.
4. Enter a branch name (e.g., feature-login) and select the source branch (usually main).
5. Click Create Branch.  
   *The branch will now be visible in the repository.*

**3. Explain Creating a Merge Request in GitLab**

1. In GitLab, go to your project.
2. Click on Merge Requests → New Merge Request.
3. Select the source branch (e.g., feature-login) and target branch (e.g., main).
4. Click Compare Branches and Continue.
5. Add a title and description for the merge request.
6. Click Submit Merge Request.
7. After review, click Merge to combine changes into the target branch.

**Git Hands-On Lab: Branching and Merging:**

**Step 1: Create and Work with a Branch**

1. Create a new branch

git branch GitNewBranch

Creates a new branch named GitNewBranch.

1. List all branches

git branch

Displays all local branches. The \* indicates the current branch.

1. Switch to the new branch

git checkout GitNewBranch

Switches to GitNewBranch.

1. Add a new file in the branch

echo "This is a new branch file" > branchfile.txt

1. Stage and commit the file

git add branchfile.txt

git commit -m "Added branchfile.txt in GitNewBranch"

1. Check the status

git status

**Step 2: Merge Branch into Main**

1. Switch to the main branch

git checkout main

1. Show differences between main and branch

git diff GitNewBranch

Displays differences between branches.

1. (Optional) View differences in P4Merge  
   Use P4Merge tool for visual comparison (if configured).
2. Merge the branch into main

git merge GitNewBranch

1. View merge history

git log --oneline --graph --decorate

1. Delete the branch after merging

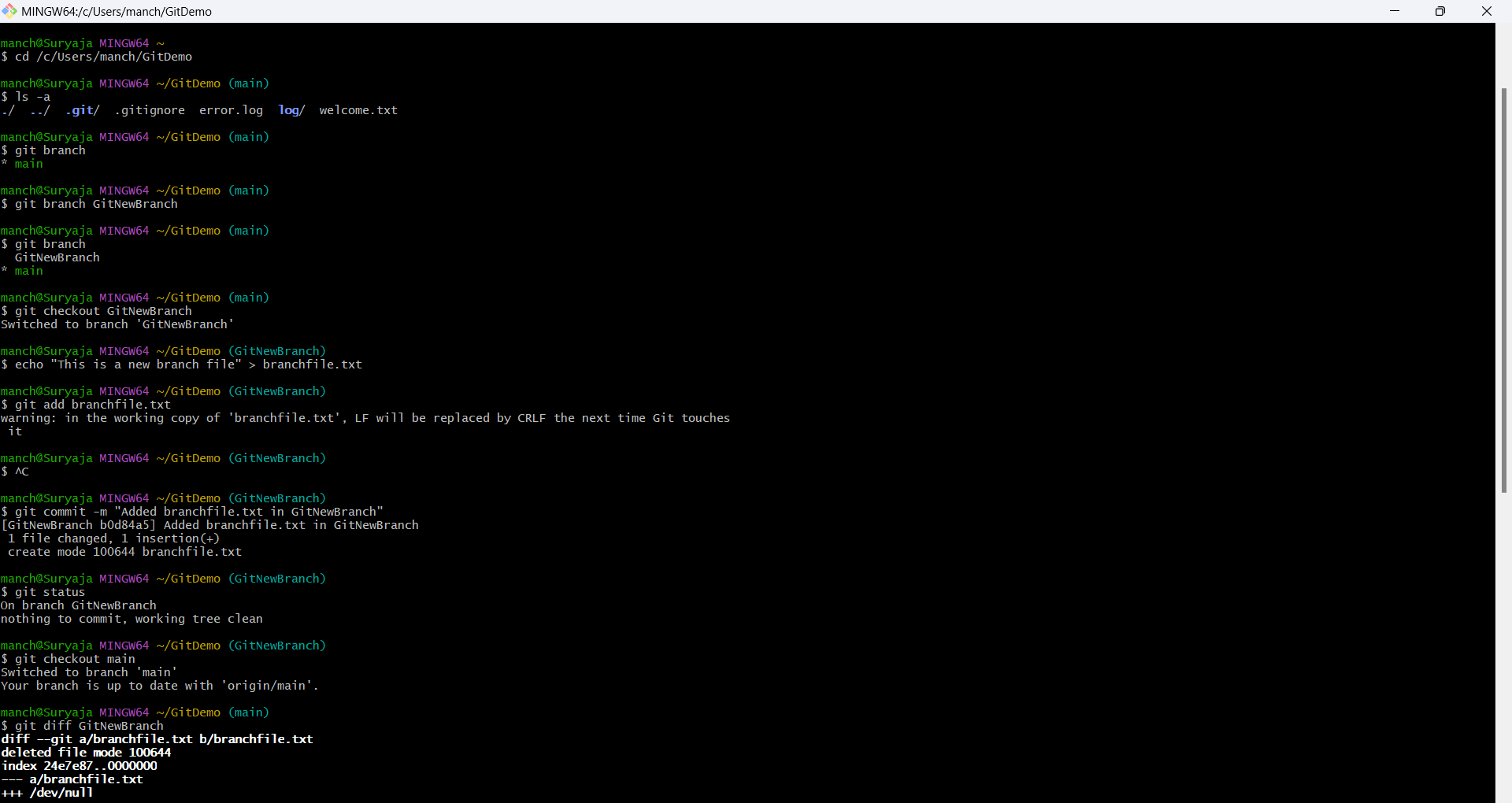
git branch -d GitNewBranch

1. Check status

git status

**Step 3: Push Changes to Remote**

git push



A black screen with white text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Git-HOL-4**

**Git Hands-On Lab: Resolving Merge Conflicts**

**Step 1: Verify Clean State**

Check if the main branch is clean:

git checkout main

git status

**Step 2: Create and Work on Branch**

1. Create branch:

git branch GitWork

1. Switch to branch:

git checkout GitWork

1. Create hello.xml:

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

1. Commit:

git add hello.xml

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

**Step 3: Switch to Main and Create Conflict**

1. Switch to main:

git checkout main

1. Create different hello.xml:

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

1. Commit:

git add hello.xml

git commit -m "Added hello.xml in main"

**Step 4: Observe Differences**

View commit graph:

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

Check differences:

git diff GitWork

**Step 5: Merge and Create Conflict**

git merge GitWork

A merge conflict will appear in hello.xml.

**Step 6: Resolve Conflict**

1. Open hello.xml and edit:

<message>Hello from both branches</message>

1. Stage the resolved file:

git add hello.xml

1. Commit the merge:

git commit

**Step 7: Handle Backup Files**

Add backup files to .gitignore:

echo "\*.orig" >> .gitignore

git add .gitignore

git commit -m "Added .gitignore for backup files"

**Step 8: Clean Up**

1. List branches:

git branch

1. Delete merged branch:

git branch -d GitWork

1. View final log:

git log --oneline --graph --decorate

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A computer screen shot of text

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Git-HOL-5**

**Explain how to clean up and push changes back to the remote repository.**

**Step 1: Verify Main Branch is Clean**

git checkout main

git status

Checks that there are no uncommitted changes in the main branch.

**Step 2: List All Branches**

git branch -a

Lists all local and remote branches.

**Step 3: Pull Latest Changes from Remote**

git pull origin main

Brings the latest changes from the remote repository into your local main branch.

**Step 4: Push Pending Changes to Remote**

git push

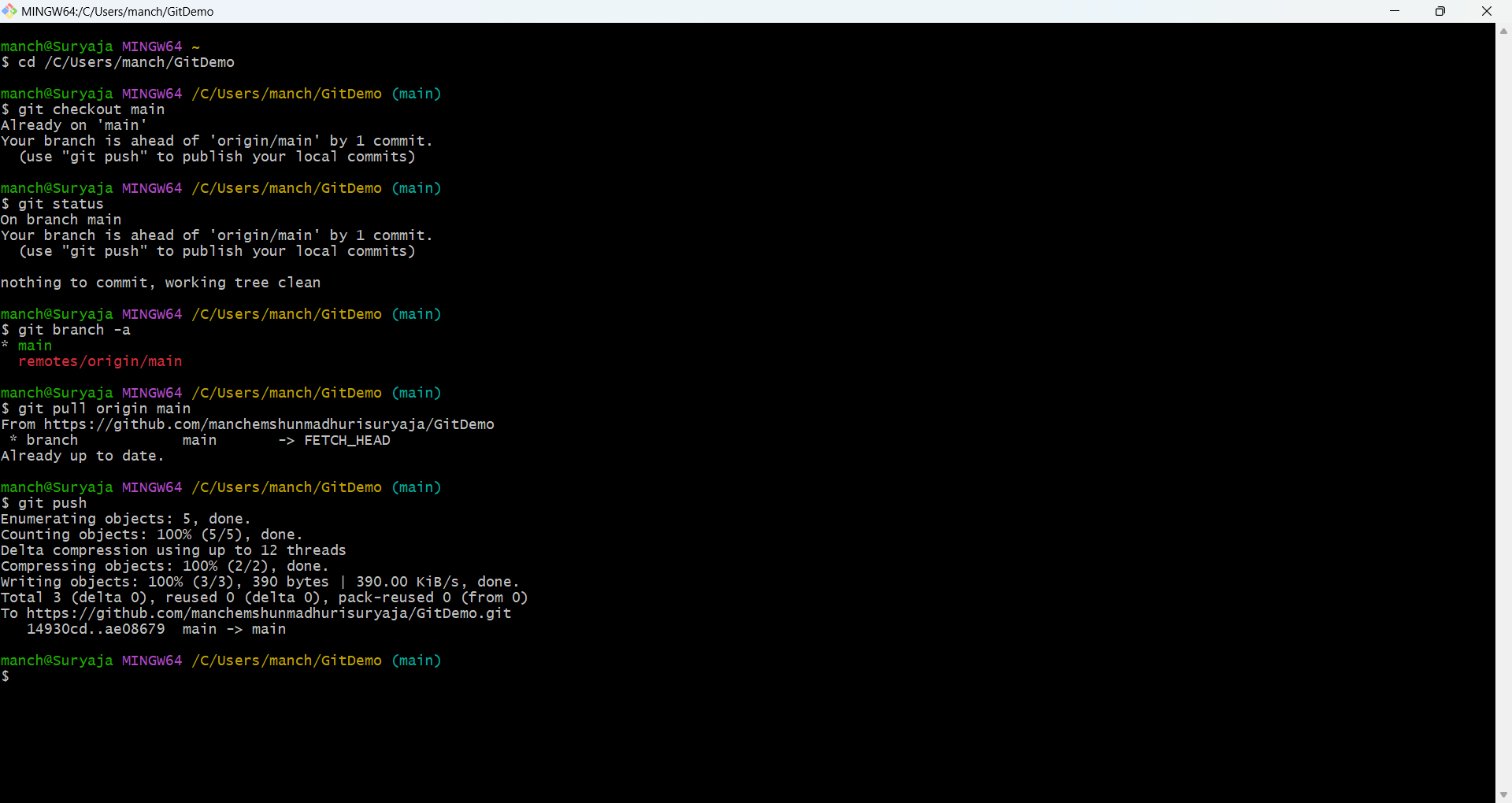
Pushes all local commits to the remote repository.

**Step 5: Verify in Remote Repository**

* Open GitHub (or GitLab) in a browser.
* Check if the latest changes and commits appear in the repository.

**Conclusion**

Successfully cleaned up the main branch and pushed all local changes to the remote repository.

****

A screenshot of a computer

AI-generated content may be incorrect.