**WEEK 8**

**GIT HOL**

**1.Git-HOL**

**Step 1: Check Git Installation**

**Open Git Bash and run:**

*git –version*

**Step 2: Configure Git User Details**

**Set your name:**

*git config --global user.name "Sidharth K"*

**Set your email:**

*git config --global user.email "iamsidharthkarthikeyan@gmail.com"*

**Verify configuration:**

*git config --global --list*

**Step 3: Add Notepad++ to PATH**

1. Find your Notepad++ install path C:\Program Files\Notepad++\notepad++.exe
2. Add to Environment Variables → Path (User Variables) → Add the above folder path.
3. Verify: notepad++

**Step 4: Make Notepad++ Default Git Editor**

git config --global core.editor "notepad++ -multiInst -nosession"

**Verify:**

git config --global -e

**Step 5: Create Local Repository**

mkdir GitDemo

cd GitDemo

git init

**Step 6: Create a File**

dir

type welcome.txt

**Step 7: Check Git Status**

git status

**Step 8: Add File to Staging**

git add welcome.txt

git status

**Step 9: Commit Changes**

git commit -m "Initial commit - added welcome.txt"

**Step 10: Link Remote Repo**

git remote add origin <https://github.com/sid110305/GitDemo.git>

**Step 11: Push to Remote**

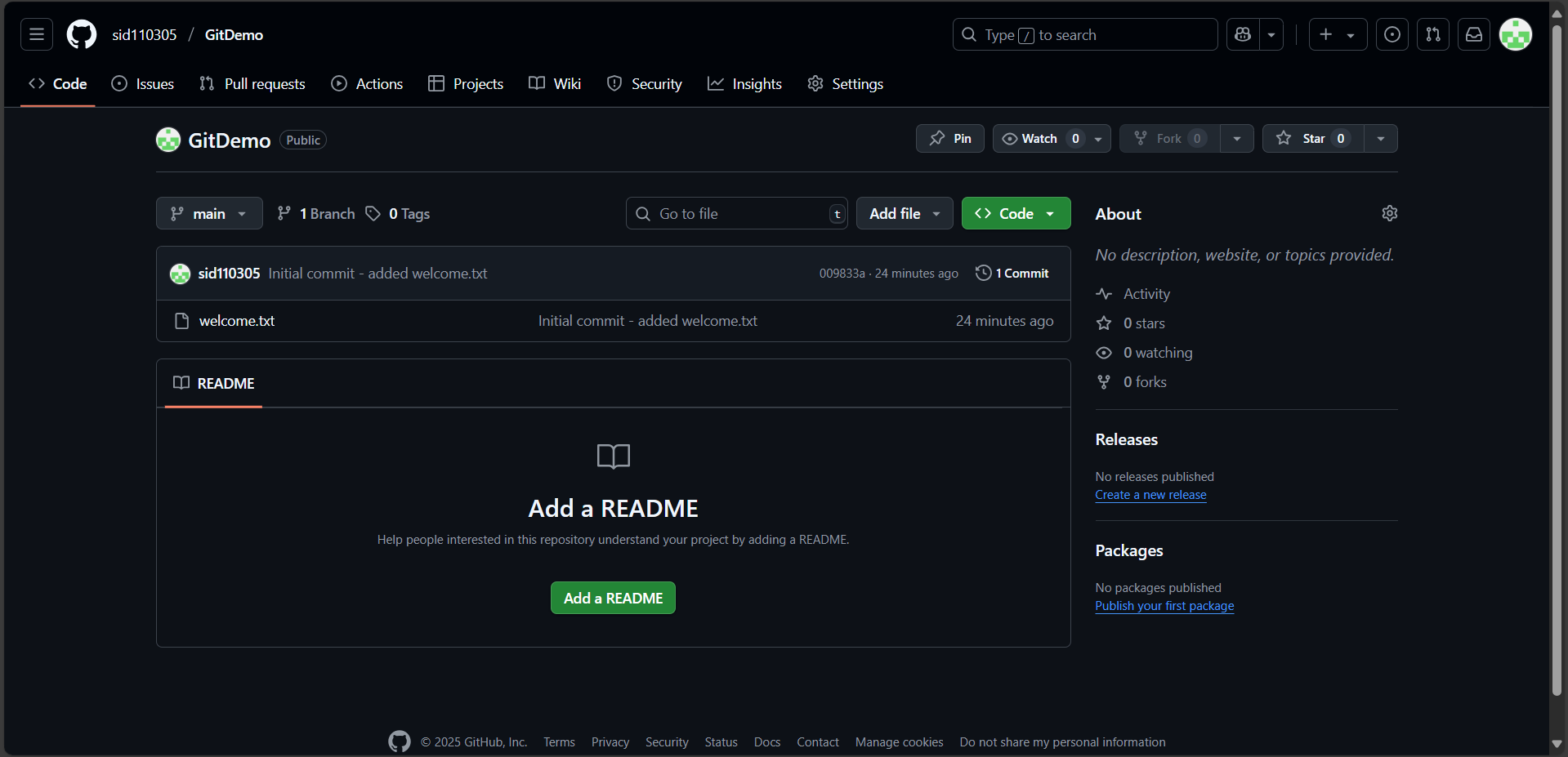
git branch -M main

git push -u origin main

**Step 12: Pull from Remote (Optional Test)**

git pull origin main

**OUTPUT:**



**2.Git-HOL-Ignoring Files in Git (Git Bash)**

**Step 1: Ensure Git is initialized**

git init

**Step 2: Create unwanted files and folder**

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

mkdir log

echo "This is inside the log folder" > log/info.txt

**Step 3: Create .gitignore file**

echo "\*.log" > .gitignore

echo "log/" >> .gitignore

**Step 4: Stage .gitignore (ignored files won’t be staged)**

git add .gitignore

**Step 5: Commit changes**

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

**Step 6: Add Remote**

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

**Step 7: Push to GitHub**

git pull origin master --allow-unrelated-histories

git push -u origin master

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**3.Git-HOL**

**Branching (Git Bash)**

1. **Create a New Branch**

git branch GitNewBranch

1. **List All Branches (Local & Remote)**

git branch -a

1. **Switch to the New Branch**

git checkout GitNewBranch

1. **Add a New File & Add Content**

echo "This is content for the new branch" > branchfile.txt

1. **Stage and Commit Changes**

git add branchfile.txt

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

1. **Check Status**

git status

**Merging (Git Bash)**

1. **Switch Back to main Branch**

git checkout main

1. **View Differences Between main & Branch (CLI)**

git diff main GitNewBranch

1. **Merge 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. **Final Status Check**

git status

**Output:**

A computer screen with text

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

**4.GIT HOL**

**Steps 1. Verify if master/main is clean**

git checkout main

git status

**Step 2. Create a new branch GitWork and add a file**

git branch GitWork

git checkout GitWork

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

**Step 3. Stage and commit changes in branch**

git add hello.xml

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

**Step 4. Switch back to master/main**

git checkout main

**Step 5. Add a different hello.xml in main**

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

git add hello.xml

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

**Step 6. View commit history (all branches)**

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

**Step 7. Compare branches (CLI diff)**

git diff main GitWork

**Step 8. Compare branches (P4Merge – optional)**

git difftool main GitWork

**Step 9. Merge GitWork into main (expect conflict)**

git merge GitWork

**Step 10. Resolve conflict (3-way merge)**

Open hello.xml → You'll see conflict markers:

<<<<<<< HEAD

<message>Hello from main branch</message> =======

<message>Hello from GitWork branch</message>

>>>>>>> GitWork

Manually edit to desired final content, e.g.:

<message>Hello from merged version</message>

**Then:**

git add hello.xml

git commit -m "Resolved merge conflict in hello.xml"

**Step 11. Add .gitignore for backup files**

echo "\*.bak" >> .gitignore

git add .gitignore

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

**Step 12. Delete the merged branch**

git branch -d GitWork

**Step 13. View final commit history**

git log --oneline --graph –decorate

**Output:**

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

**5.Git-HOL**

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

git status

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

git branch -a

**Step 3. Pull the remote Git repository to the master branch**

git checkout master

git pull origin master

**Step 4. Push the pending changes from “Git-T03-HOL\_002” to remote**

git checkout Git-T03-HOL\_002

git add .

git commit -m

git push origin Git-T03-HOL\_002

**Step 5. Merge Git-T03-HOL\_002 changes into master and push**

git checkout master

git merge Git-T03-HOL\_002

git push origin master

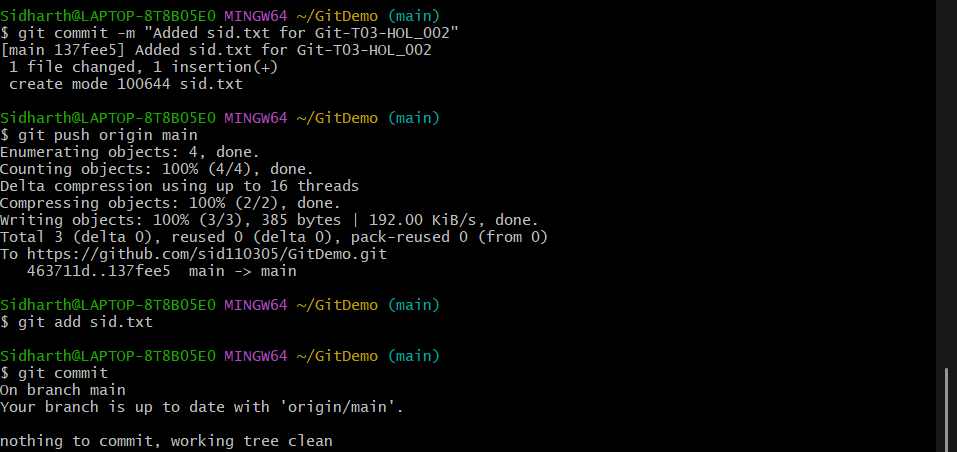
**then:**

git add sid.txt

git commit

git push origin master

**Output:**

 A screenshot of a computer

AI-generated content may be incorrect.