**Week 8**

**GIT**

**1.Git-HOL**

**Introduction:**

This exercise demonstrates the step-by-step process of creating a local Git repository, adding and committing a file, linking the repository to GitHub, and pushing the changes to a remote repository. It helps in understanding the basic Git workflow for version control.

**Code:**

git config --global user.name "Mahalakshmi P"

git config --global user.email "mahalakshmip05805@gmail.com"

mkdir GitDemo

cd GitDemo

git init

echo "Welcome to GitDemo" > welcome.txt

git add welcome.txt

git commit -m "Add welcome.txt"

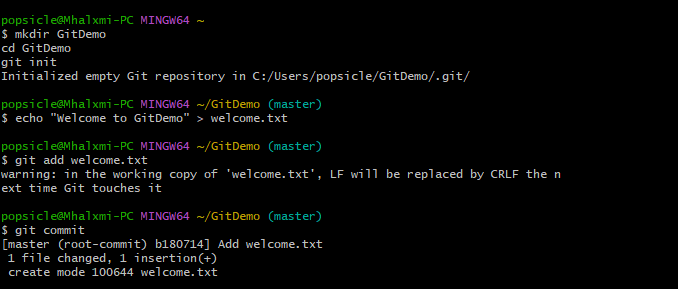
git remote add origin https://github.com/Mahalakshmi2505/GitDemo.git

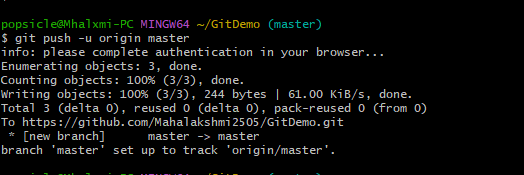
git remote -v

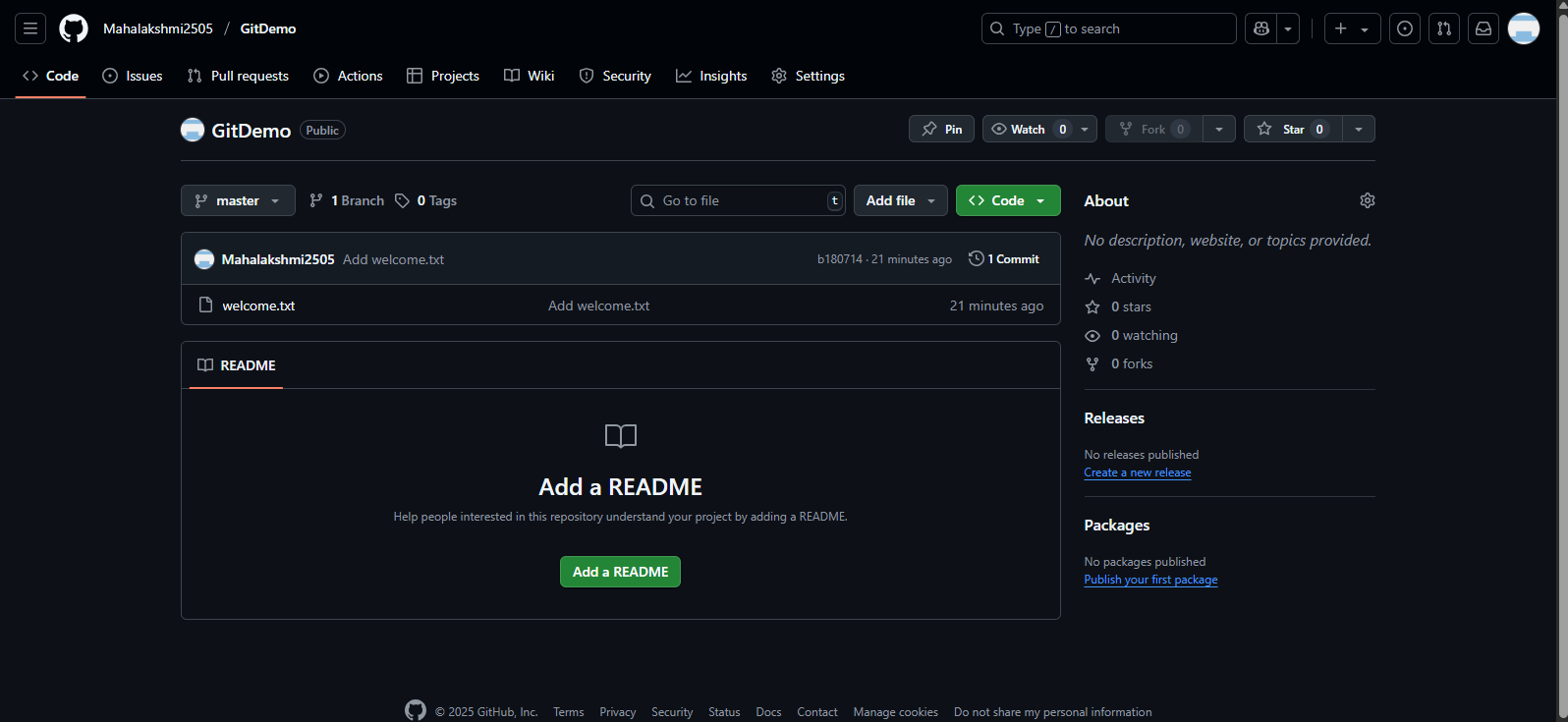
git branch

git push -u origin master

**Screenshots:**







**Conclusion:**

The repository was successfully created locally, linked to GitHub, and the committed file was pushed to the remote repository. This process establishes the foundation for managing code changes and collaborating effectively using Git.

2.Git-HOL

**Introduction:**

When working with Git repositories, some files and folders should not be tracked—such as log files, temporary build files, and personal configuration files. Git provides the .gitignore file to specify patterns for files and directories that should be ignored. This ensures cleaner repositories, avoids committing unnecessary files, and prevents sensitive data from being uploaded.

Code:

**popsicle@Mhalxmi-PC MINGW64 ~**

**$ mkdir GitIgnoreDemo**

**cd GitIgnoreDemo**

**git init**

**Initialized empty Git repository in C:/Users/popsicle/GitIgnoreDemo/.git/**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

**$ echo "this is app log" > app.log**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

**$ echo "hidden log" > .log**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

**$ echo "hidden log" > .log**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

**$ echo "entry" > log/debug.log**

**bash: log/debug.log: No such file or directory**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

**$ mkdir log**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

**$ echo "entry" > log/debug.log**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

**$ cat > .gitignore <<'EOF'**

**> \*.log**

**>**

**log/**

**EOF**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

**$ git status --ignored**

**On branch master**

**No commits yet**

**Untracked files:**

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

**.gitignore**

**Ignored files:**

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

**.log**

**app.log**

**log/**

**nothing added to commit but untracked files present (use "git add" to track)**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

**$ git add .gitignore**

**warning: in the working copy of '.gitignore', LF will be replaced by CRLF the next time Git touches it**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

**$ git commit -m "Add .gitignore to ignore .log files and log folder"**

**[master (root-commit) 65316ea] Add .gitignore to ignore .log files and log folder**

**1 file changed, 3 insertions(+)**

**create mode 100644 .gitignore**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

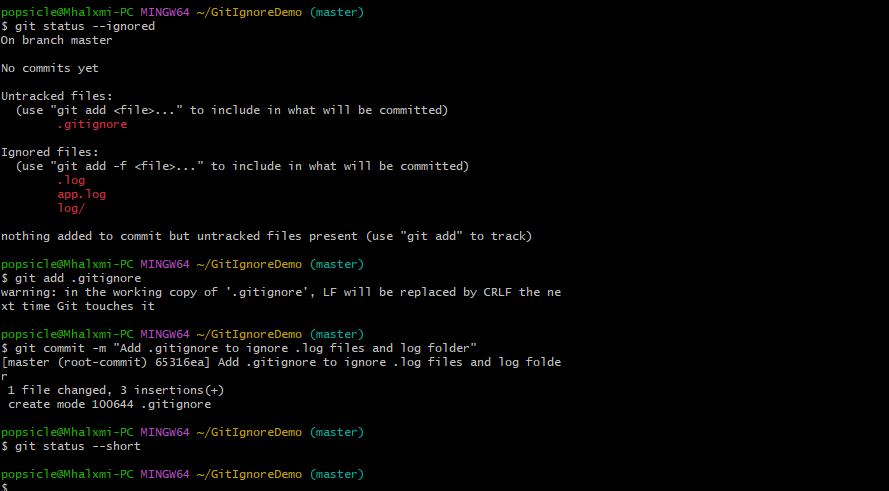
**$ git status --short**

**popsicle@Mhalxmi-PC MINGW64 ~/GitIgnoreDemo (master)**

**$**

**Screenshot:**

****

****

**Conclusion:**

In this exercise, we successfully created and tested a .gitignore file to ignore .log files and the log/ directory in a Git repository. This approach helps maintain a clean repository by preventing unnecessary or temporary files from being tracked, making collaboration easier and more efficient.

**3. Git-HOL**

**Introduction:**

Branching in Git allows developers to work on features or bug fixes in isolation without affecting the main codebase. Merging integrates those changes back into the main branch (often master or main). This hands-on lab demonstrates how to create a branch, make changes, and merge them back into the trunk, using both Git commands and the P4Merge visual tool.

**Code:**

popsicle@Mhalxmi-PC MINGW64 ~

$ mkdir GitBranchDemo

cd GitBranchDemo

git init

Initialized empty Git repository in C:/Users/popsicle/GitBranchDemo/.git/

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (master)

$ echo "This is the master branch file" > masterfile.txt

git add masterfile.txt

git commit -m "Initial commit in master branch"

warning: in the working copy of 'masterfile.txt', LF will be replaced by CRLF the next time Git touches it

[master (root-commit) bf6716f] Initial commit in master branch

1 file changed, 1 insertion(+)

create mode 100644 masterfile.txt

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (master)

$ git branch GitNewBranch

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (master)

$ git branch -a

GitNewBranch

\* master

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (master)

$ git checkout GitNewBranch

Switched to branch 'GitNewBranch'

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (GitNewBranch)

$ echo "This file is in GitNewBranch" > branchfile.txt

git add branchfile.txt

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

warning: in the working copy of 'branchfile.txt', LF will be replaced by CRLF the next time Git touches it

[GitNewBranch 3a6da89] Add branchfile.txt in GitNewBranch

1 file changed, 1 insertion(+)

create mode 100644 branchfile.txt

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (GitNewBranch)

$ git status

On branch GitNewBranch

nothing to commit, working tree clean

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (GitNewBranch)

$ git checkout master

Switched to branch 'master'

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (master)

$ git diff master GitNewBranch

diff --git a/branchfile.txt b/branchfile.txt

new file mode 100644

index 0000000..be68bfa

--- /dev/null

+++ b/branchfile.txt

@@ -0,0 +1 @@

+This file is in GitNewBranch

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (master)

$ git merge GitNewBranch

Updating bf6716f..3a6da89

Fast-forward

branchfile.txt | 1 +

1 file changed, 1 insertion(+)

create mode 100644 branchfile.txt

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (master)

$ git log --oneline --graph --decorate

\* 3a6da89 (HEAD -> master, GitNewBranch) Add branchfile.txt in GitNewBranch

\* bf6716f Initial commit in master branch

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (master)

$ git branch -d GitNewBranch

Deleted branch GitNewBranch (was 3a6da89).

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (master)

$ git status

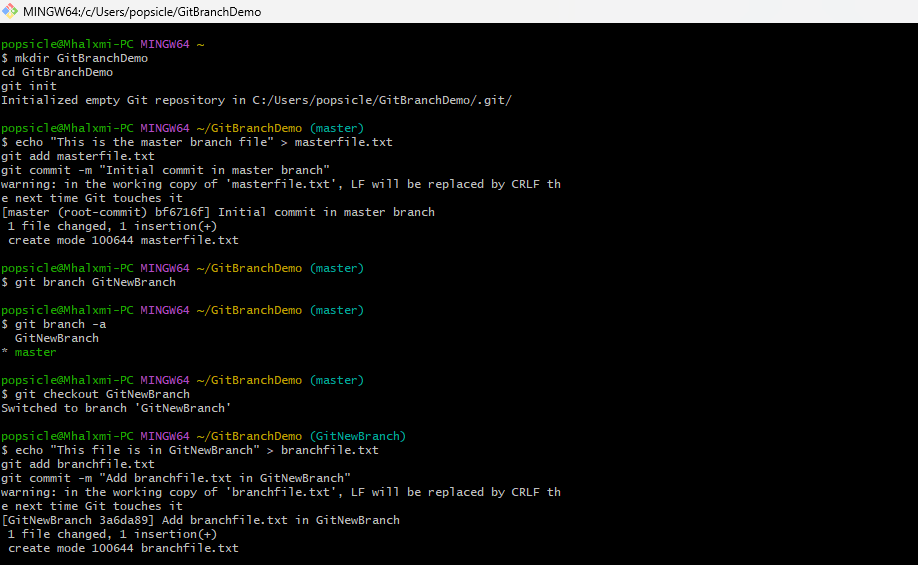
On branch master

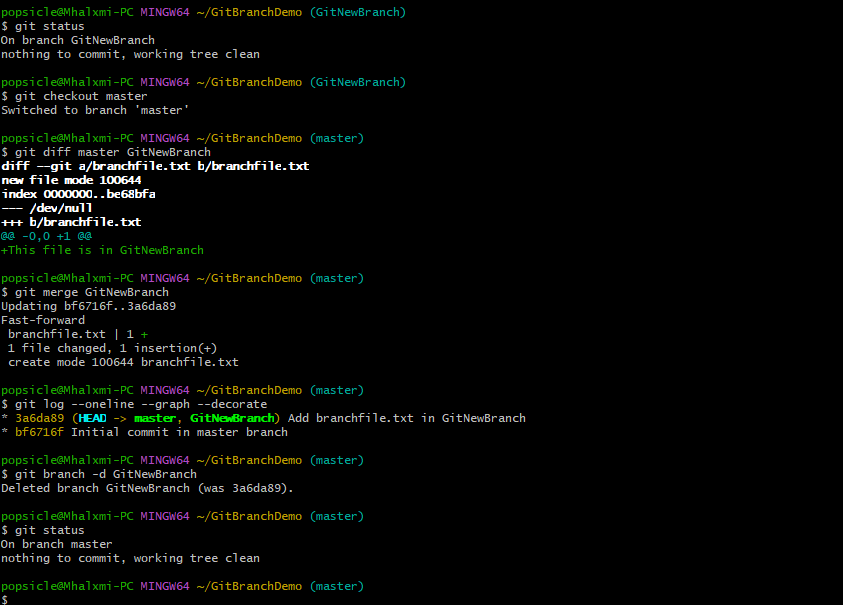
nothing to commit, working tree clean

popsicle@Mhalxmi-PC MINGW64 ~/GitBranchDemo (master)

$

**Screenshot:**

****

****

**Conclusion:**

In this lab, we created a branch, made changes, and merged it back into the master branch. We explored both command-line and visual difference tools (P4Merge) to compare changes. Finally, we deleted the merged branch to maintain a clean repository. This process demonstrates how branching and merging help manage changes in collaborative software development.

**4. Git-HOL**

**Introduction:**

In this lab exercise, we explored how to handle merge conflicts in Git using the visual merge tool P4Merge. Merge conflicts occur when changes from different branches overlap and Git cannot automatically reconcile them. By configuring P4Merge as the default merge tool, we resolved conflicts efficiently by visually comparing the conflicting versions (Local, Remote, and Base) and editing the merged result before committing the changes. This process helps maintain code integrity and ensures smooth collaboration in team projects.

**Code:**

popsicle@Mhalxmi-PC MINGW64 ~

$ mkdir GitConflictDemo

cd GitConflictDemo

git init

Initialized empty Git repository in C:/Users/popsicle/GitConflictDemo/.git/

popsicle@Mhalxmi-PC MINGW64 ~/GitConflictDemo (master)

$ cat > hello.xml <<'EOF'

<message>Hello — base version</message>

EOF

git add hello.xml

git commit -m "Initial commit: add hello.xml (base)"

warning: in the working copy of 'hello.xml', LF will be replaced by CRLF the next time Git touches it

[master (root-commit) 8c5125a] Initial commit: add hello.xml (base)

1 file changed, 1 insertion(+)

create mode 100644 hello.xml

popsicle@Mhalxmi-PC MINGW64 ~/GitConflictDemo (master)

$ git branch GitWork

git checkout GitWork

Switched to branch 'GitWork'

popsicle@Mhalxmi-PC MINGW64 ~/GitConflictDemo (GitWork)

$ cat > hello.xml <<'EOF'

<message>Hello from branch GitWork</message>

EOF

git add hello.xml

git commit -m "GitWork: update hello.xml"

warning: in the working copy of 'hello.xml', LF will be replaced by CRLF the next time Git touches it

[GitWork 52d0cbd] GitWork: update hello.xml

1 file changed, 1 insertion(+), 1 deletion(-)

popsicle@Mhalxmi-PC MINGW64 ~/GitConflictDemo (GitWork)

$ git checkout master

cat > hello.xml <<'EOF'

<message>Hello from master (different change)</message>

EOF

git add hello.xml

git commit -m "master: update hello.xml differently"

Switched to branch 'master'

warning: in the working copy of 'hello.xml', LF will be replaced by CRLF the next time Git touches it

[master 2d660a7] master: update hello.xml differently

1 file changed, 1 insertion(+), 1 deletion(-)

popsicle@Mhalxmi-PC MINGW64 ~/GitConflictDemo (master)

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

git branch -a

\* 2d660a7 (HEAD -> master) master: update hello.xml differently

| \* 52d0cbd (GitWork) GitWork: update hello.xml

|/

Merging:

hello.xml

Normal merge conflict for 'hello.xml':

{local}: modified file

{remote}: modified file

popsicle@Mhalxmi-PC MINGW64 ~/GitConflictDemo (master|MERGING)

$ git add hello.xml

popsicle@Mhalxmi-PC MINGW64 ~/GitConflictDemo (master|MERGING)

$ git commit -m "Resolved merge conflict using P4Merge"

[master 9369273] Resolved merge conflict using P4Merge

popsicle@Mhalxmi-PC MINGW64 ~/GitConflictDemo (master)

$ git add hello.xml

git commit -m "Resolved merge conflict using P4Merge"

On branch master

nothing to commit, working tree clean

popsicle@Mhalxmi-PC MINGW64 ~/GitConflictDemo (master)

$ git log --oneline --graph --decorate

git status

\* 9369273 (HEAD -> master) Resolved merge conflict using P4Merge

|\

| \* 52d0cbd (GitWork) GitWork: update hello.xml

\* | 2d660a7 master: update hello.xml differently

|/

\* 8c5125a Initial commit: add hello.xml (base)

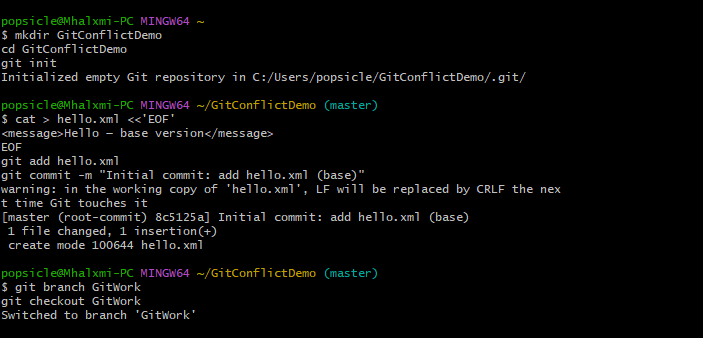
On branch master

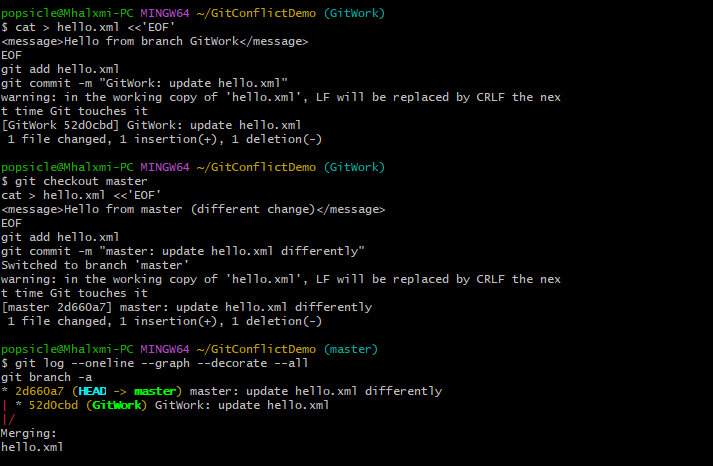
nothing to commit, working tree clean

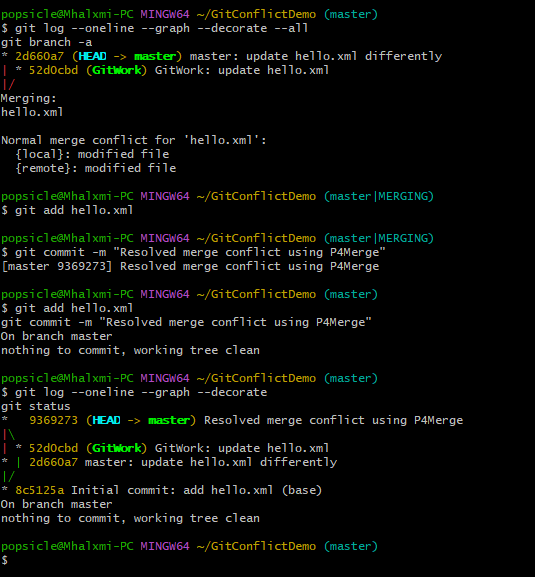
popsicle@Mhalxmi-PC MINGW64 ~/GitConflictDemo (master)

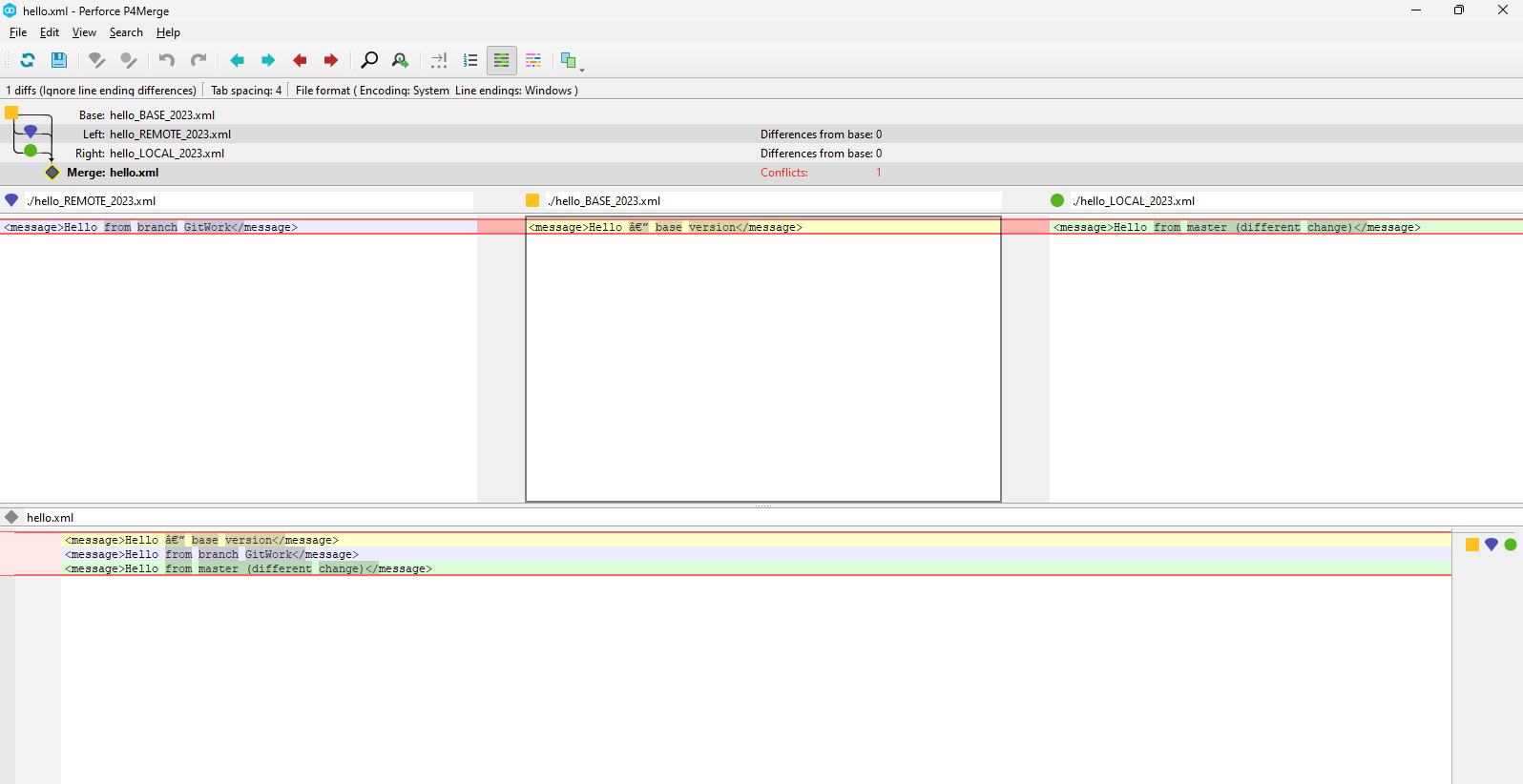
$

**Screenshots:**

****

****

****

****

**Conclusion:**

By completing this exercise, we gained practical experience in managing merge conflicts using P4Merge. We learned how to interpret the different panes representing conflicting changes, select or combine changes as needed, and finalize the merge. Proper conflict resolution is critical in software development workflows to integrate code from multiple contributors without losing important modifications. This hands-on practice reinforces the importance of tools like P4Merge in simplifying complex merges and maintaining a clean Git history.

**5. Git-HOL**

**Introduction:**

In this hands-on lab, we explored the fundamental Git operations of verifying a clean working state, synchronizing local and remote repositories, and pushing feature branch changes back to the remote GitHub repository. The lab reinforced best practices for branch management, ensuring that local branches are up to date before committing and pushing changes. This exercise helps build confidence in using Git for version control in collaborative software development projects.

**Code:**

popsicle@Mhalxmi-PC MINGW64 ~

$ cd GitDemo

popsicle@Mhalxmi-PC MINGW64 ~/GitDemo (master)

$ git status

git pull origin master

git branch -a

On branch master

Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

From https://github.com/Mahalakshmi2505/GitDemo

\* branch master -> FETCH\_HEAD

Already up to date.

\* master

remotes/origin/master

popsicle@Mhalxmi-PC MINGW64 ~/GitDemo (master)

$ git branch

\* master

popsicle@Mhalxmi-PC MINGW64 ~/GitDemo (master)

$ git checkout -b Git-T03-HOL\_002

Switched to a new branch 'Git-T03-HOL\_002'

popsicle@Mhalxmi-PC MINGW64 ~/GitDemo (Git-T03-HOL\_002)

$ git add .

popsicle@Mhalxmi-PC MINGW64 ~/GitDemo (Git-T03-HOL\_002)

$ git commit -m "Add changes for Git-T03-HOL\_002"

On branch Git-T03-HOL\_002

nothing to commit, working tree clean

popsicle@Mhalxmi-PC MINGW64 ~/GitDemo (Git-T03-HOL\_002)

$ git push origin Git-T03-HOL\_002

Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)

remote:

remote: Create a pull request for 'Git-T03-HOL\_002' on GitHub by visiting:

remote: https://github.com/Mahalakshmi2505/GitDemo/pull/new/Git-T03-HOL\_00

remote:

To https://github.com/Mahalakshmi2505/GitDemo.git

\* [new branch] Git-T03-HOL\_002 -> Git-T03-HOL\_002

popsicle@Mhalxmi-PC MINGW64 ~/GitDemo (Git-T03-HOL\_002)

$ git checkout master

Switched to branch 'master'

Your branch is up to date with 'origin/master'.

popsicle@Mhalxmi-PC MINGW64 ~/GitDemo (master)

$ git pull origin master

From https://github.com/Mahalakshmi2505/GitDemo

\* branch master -> FETCH\_HEAD

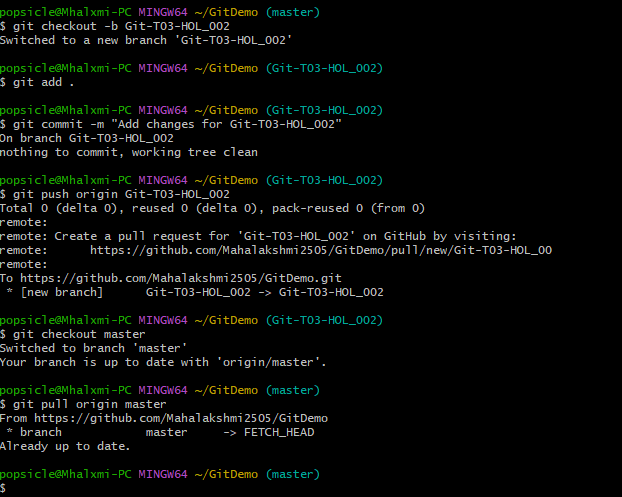
Already up to date.

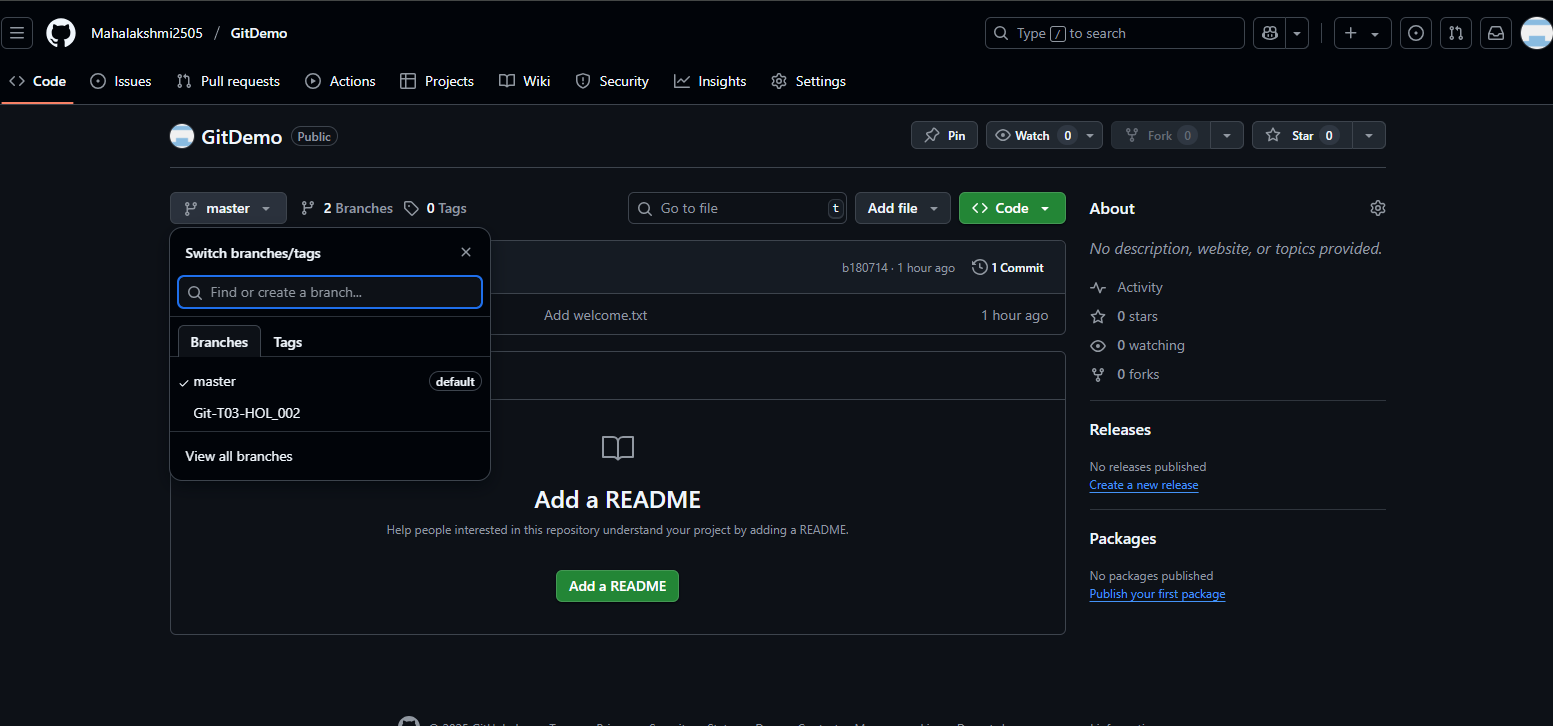
popsicle@Mhalxmi-PC MINGW64 ~/GitDemo (master)

$

**Screenshot:**







**Conclusion:**

This lab successfully demonstrated the process of managing branches in Git, including checking the branch status, pulling remote updates, creating and pushing a new feature branch to GitHub. By completing these steps, we ensured that local changes were safely integrated with the remote repository. Additionally, we learned how to create a pull request on GitHub, which facilitates code review and collaboration before merging changes into the main branch. Mastery of these Git workflows is essential for effective source code management in real-world development environments.