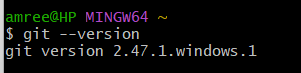
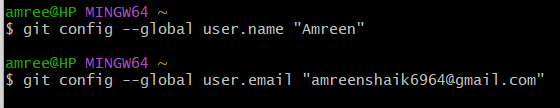
**Week-8 Handson**

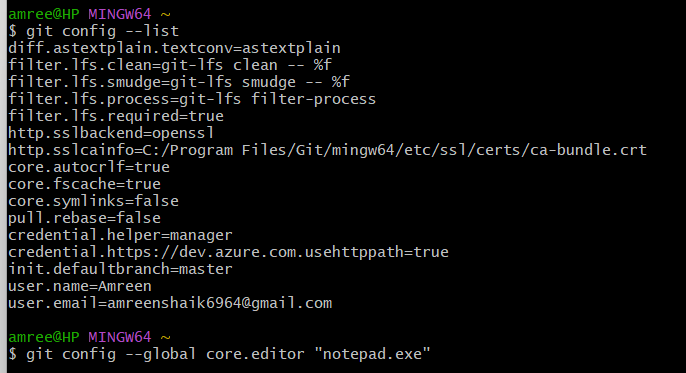
**1.Git-HOL**

Setup your machine with Git Configuration

**git –version**

**Configure your identity**

****

**Set Notepad as Git’s default editor**

Create a local repository

mkdir GitDemo

cd GitDemo

git init

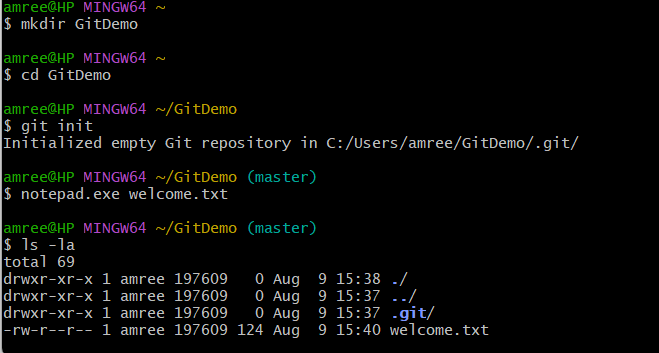
Create a file using Notepad

notepad.exe welcome.txt

Verify file:

ls -la

cat welcome.txt

l

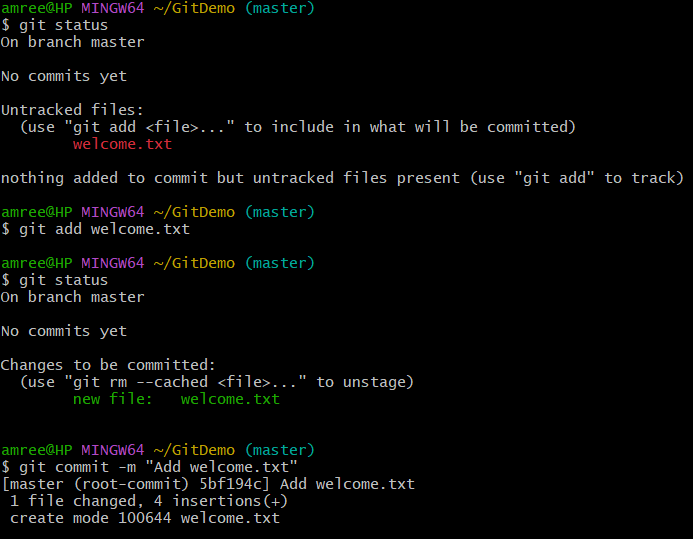
Stage and commit the file:

git status

git add welcome.txt

git status

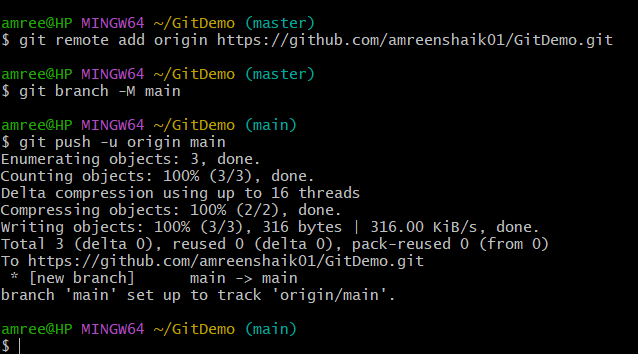
git commit -m "Add welcome.txt"

l

Create a remote repository (GitHub):

Created a new repository named GitDemo

Link local repo to remote and push:



**2.Git-HOL:**

We can do this using already created previous GitDemo repository.

In that, you can do this:

**Create sample .log files and a log folder:**

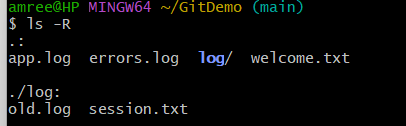
echo "error: failed to start" > errors.log

echo "app running" > app.log

mkdir log

echo "old log entry" > log/old.log

echo "session data" > log/session.txt

l

Check status (you should see these as untracked):

git status

Create the .gitignore file:

notepad.exe .gitignore

Notepad will be opened here:

In that

# Ignore all .log files anywhere in the repo

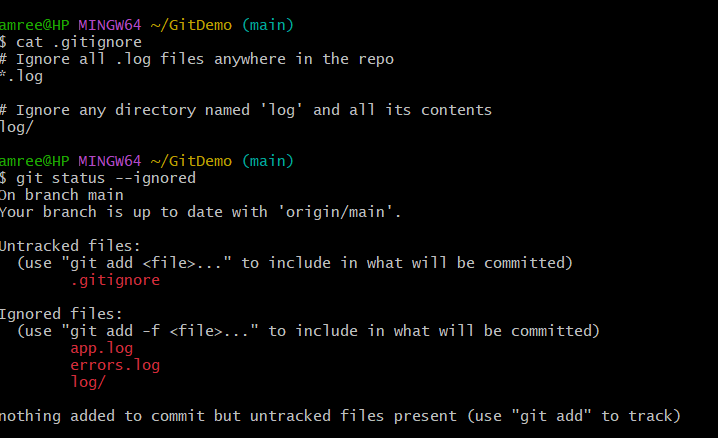
\*.log

# Ignore any directory named 'log' and all its contents

log/

Save it

**Verify .gitignore content:**

**cat .gitignore**

git status

ignored files will not appear here now.

Commit the .gitignore:

git add .gitignore

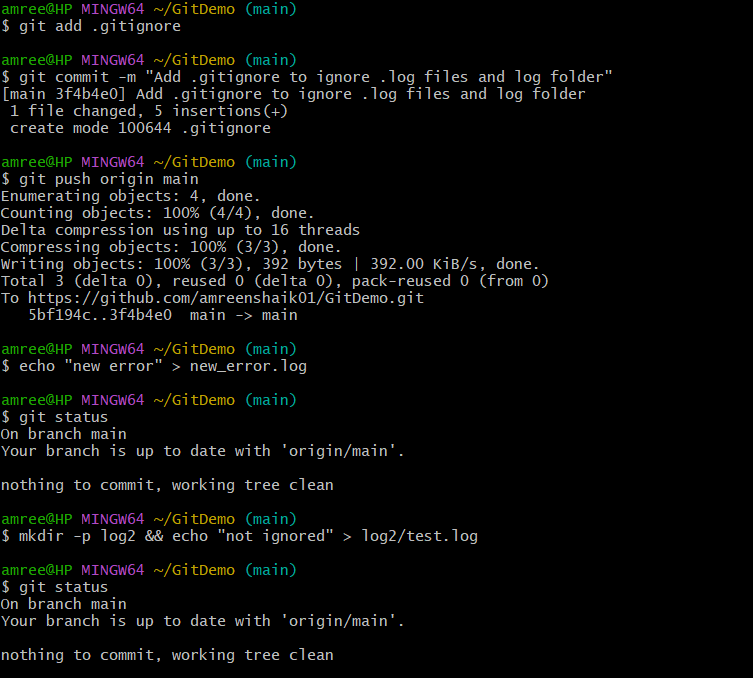
git commit -m "Add .gitignore to ignore .log files and log folder"

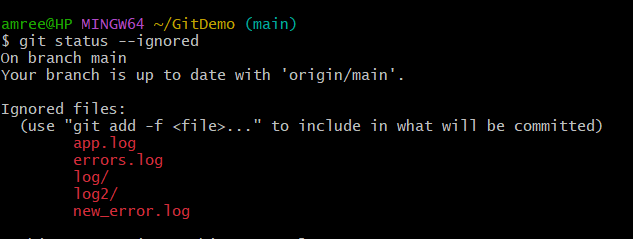
**Quick tests to confirm ignoring works**

Create a new log file

Create a file inside a log folder:

Files in log/ are ignored; files in log2/ are not (unless you add a pattern to ignore log2/).





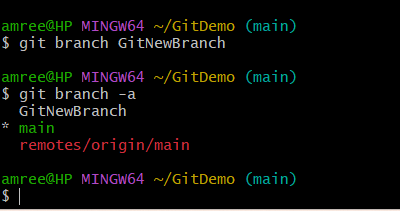
**3.Git-HOL**

In GitBash after going to the respective repository,

We will see how to create a branch

**Create a new branch**

git branch GitNewBranch

l

**Switch to the new branch**

git checkout GitNewBranch

Add some files to the branch:

Create a file in this branch:

echo "Hello from new branch" > branchfile.txt

Stage it:

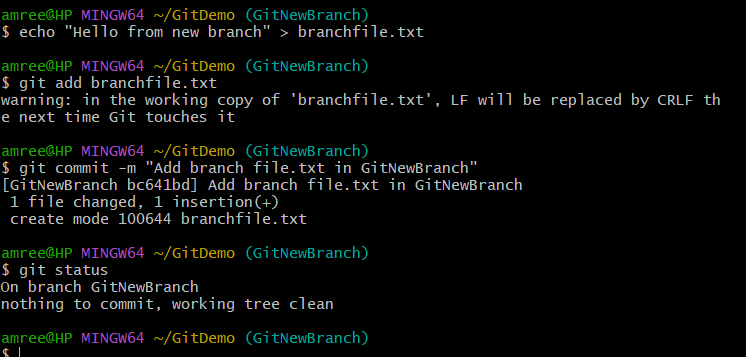
git add branchfile.txt

Commit:

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

Check branch status

git status

l

**Merging:**

**Switch to main**

git checkout main

See differences between master and branch (CLI)

git diff master..GitNewBranch

Merge the branch into main:

git merge GitNewBranch

View merge history:

git log --oneline --graph –decorate

It shows branches, merges, and commits visually.

Delete the merged branch:

git branch -d GitNewBranch

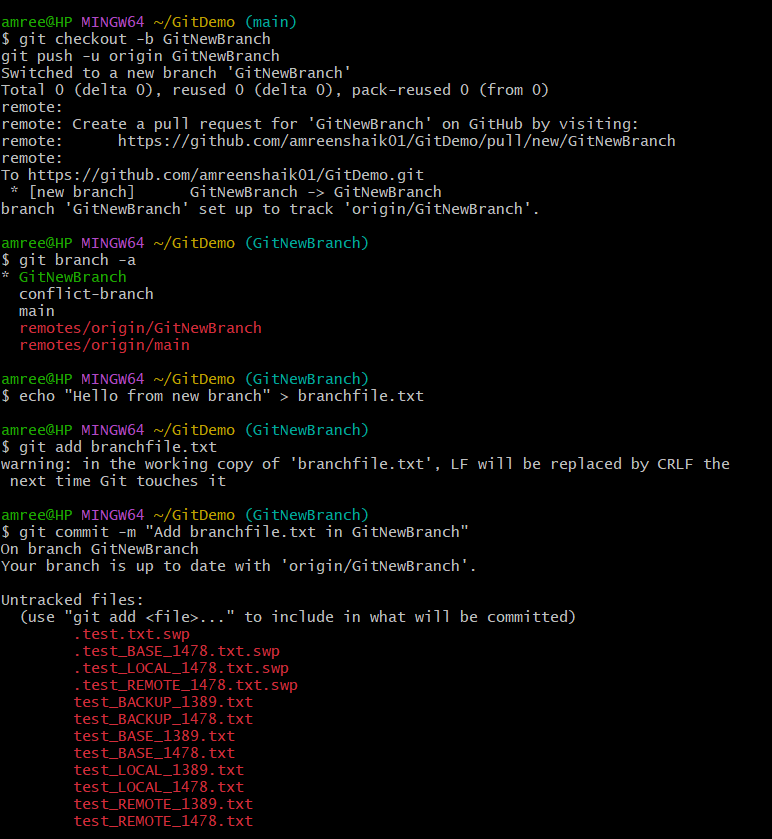
Push changes to remote:

git push origin master

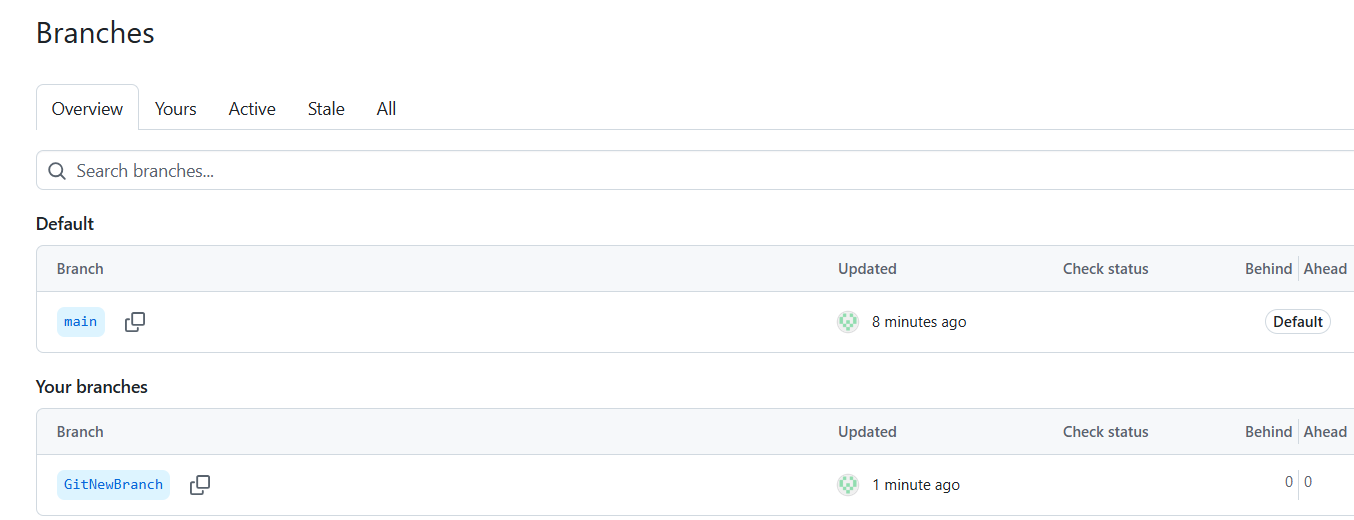
git push origin --delete GitNewBranch

Creating a Merge Request in Github

git push origin GitNewBranch



A new branch is created here:



**4.Git-HOL:**

**Resolve merge conflict lab**

Creating a repo and initializing it

Verify main is in clean state or not

git status

Create branch GitWork and add hello.xml:

git checkout -b GitWork

Content:

cat > hello.xml <<'EOF'

<hello>Branch version: Hello from GitWork</hello>

EOF

git add hello.xml

Update content of hello.xml (on the branch) and observe status:

cat > hello.xml <<'EOF'

<hello>Branch version: UPDATED content from GitWork</hello>

EOF

git status

Here, you should see hello.xml as modified (or staged if you add it)

Commit the changes to the branch:

git add hello.xml

git commit -m "Add/Update hello.xml on GitWork"

Switch back to main:

git checkout main

Add a different hello.xml on main (create conflict later):

cat > hello.xml <<'EOF'

<hello>Master version: Hello from master</hello>

EOF

git add hello.xml

Commit the changes to main

git commit -m "Add hello.xml on master with different content"

See the history graph (all branches):

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

Check differences with git diff:

# show difference between master and GitWork.

git diff main..GitWork

# to just list file names:

git diff --name-only main..GitWork

Now,run:

git difftool main..GitWork

Merge the branch into main (this will create a conflict):

git checkout main

git merge GitWork

**Observe the git markup (conflict markers)**

**git status**

It shows both modified hello.xml

cat hello.xml

Use a 3-way merge tool to resolve the conflict:

git mergetool

Commit the merge (once conflicts resolved):

git commit -m "Merge branch 'GitWork' into master — resolved hello.xml conflict"

Observe git status and add backup files to .gitignore:

cat >> .gitignore <<'EOF'

\*BACKUP\*

\*.swp

\*.swx

\*.orig

EOF

git add .gitignore

Commit the .gitignore:

git commit -m "Ignore merge backup and swap files"

**List all branches**

git branch -a

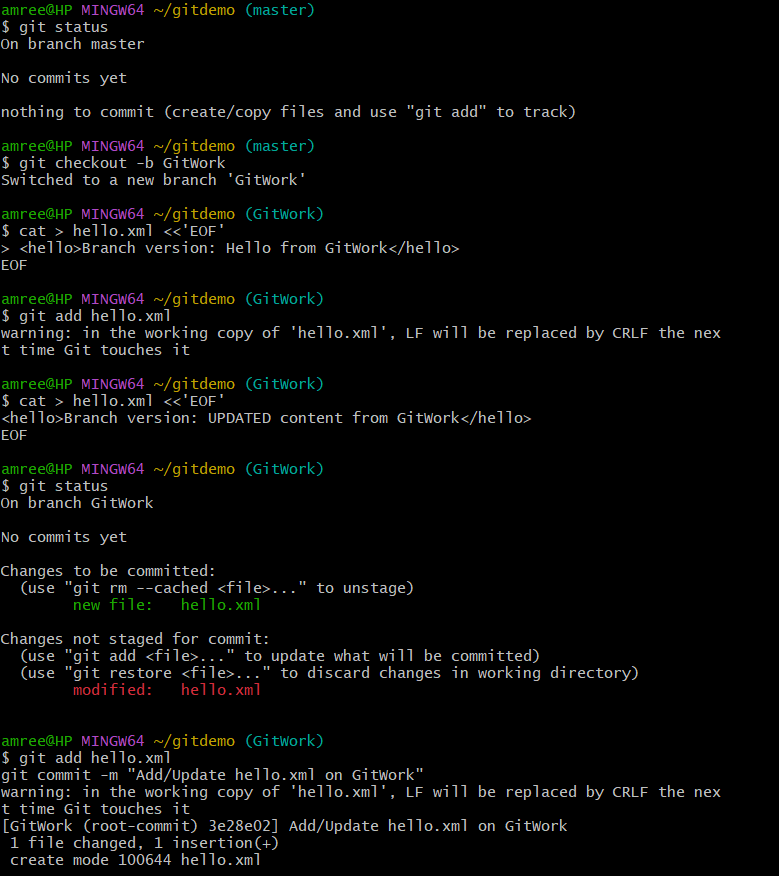
Delete the branch you merged into master:

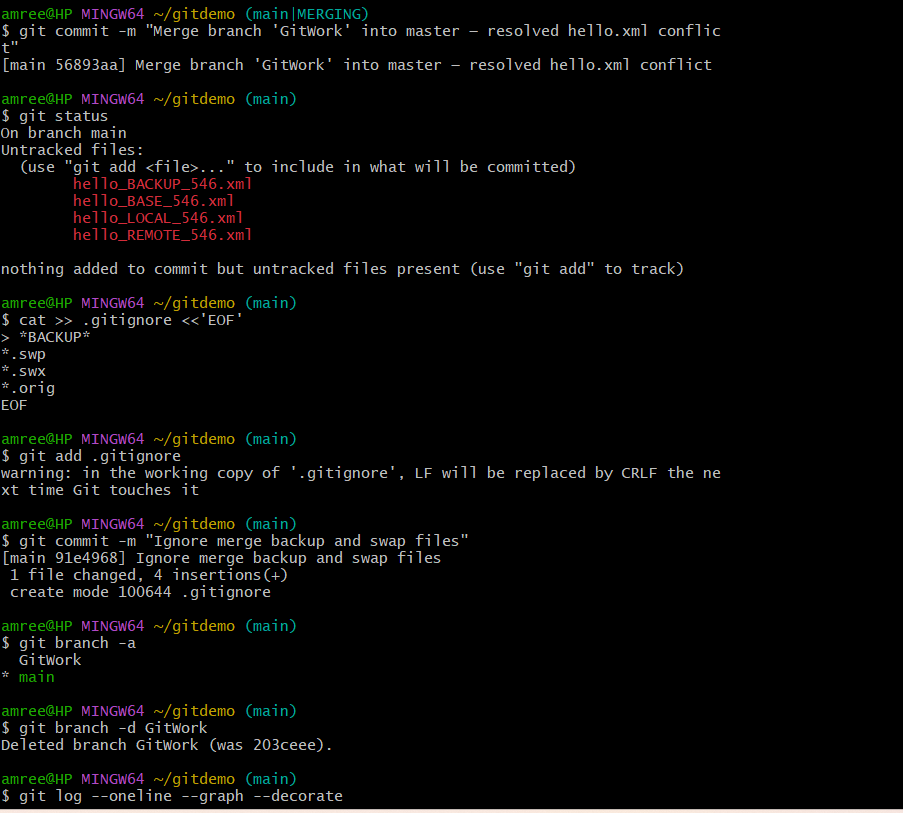
git branch -d GitWork

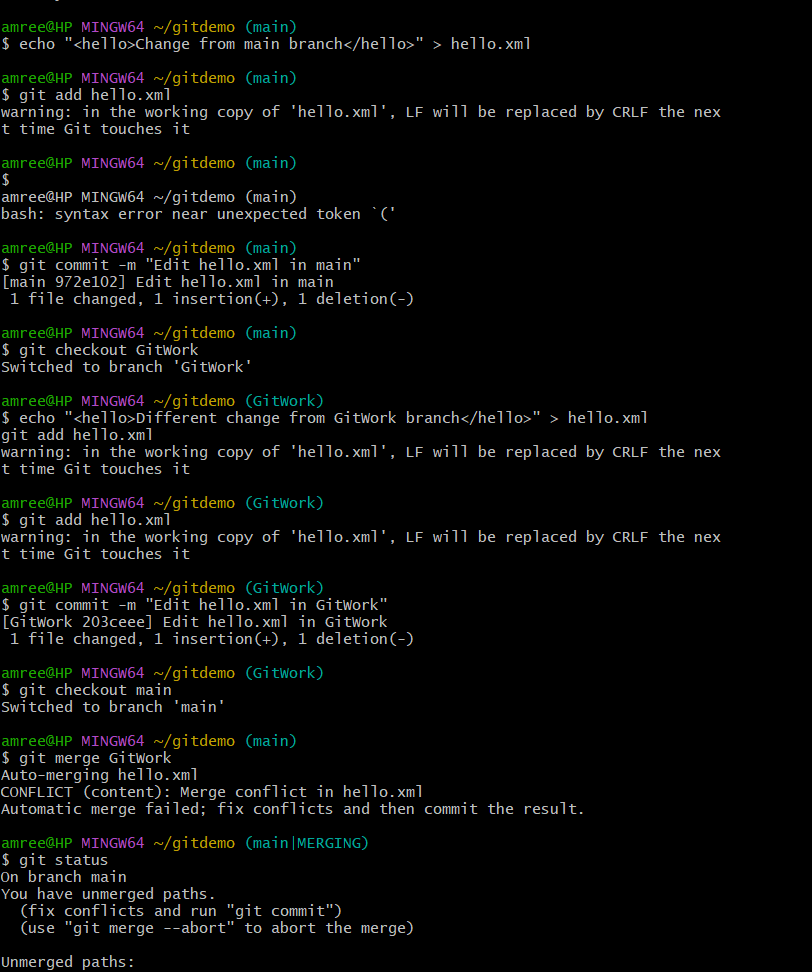
Final log: confirm merge is recorded:

git log --oneline --graph –decorate

You should see a merge commit in the graph.







**5.Git-HOL:**

Clean and push back to remote git

Verify the branch (master/main) is in a clean state:

git status

List all available branches (local + remote)

git branch -a

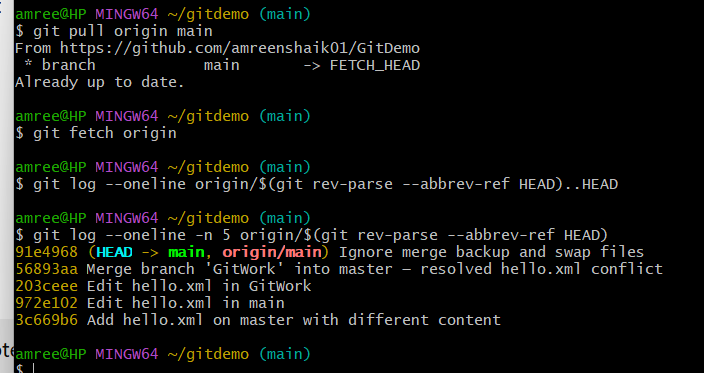
Pull the remote repository into the (default) branch:

git branch -a | grep origin | grep -E 'origin/(main)' -m1

Now,

git checkout main

git pull origin main



Verify the push and check on remote:

# update refs from remote

git fetch origin

# show commits that exist locally but not on remote (current branch)

git log --oneline origin/$(git rev-parse --abbrev-ref HEAD)..HEAD

# show the remote branch tip

git log --oneline -n 5 origin/$(git rev-parse --abbrev-ref HEAD)

Useful cleanup commands if needed