# HandsOn 1:

## Step 1: Setup your machine with Git Configuration

1. Install Git Bash from https://git-scm.com/ and complete the installation.  
2. Open Git Bash and verify installation:  
 git --version  
3. Configure your Git username and email:  
 git config --global user.name "Your Name"  
 git config --global user.email "your.email@example.com"  
4. Verify configuration:  
 git config --global --list

## Step 2: Integrate Notepad++ as Default Git Editor

1. Download and install Notepad++ from https://notepad-plus-plus.org/.  
2. Add Notepad++ installation path (e.g., C:/Program Files/Notepad++) to your system environment PATH.  
3. Configure Git to use Notepad++:  
 git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -nosession -wait"  
4. Verify editor configuration:  
 git config --global -e

## Step 3: Create a New Repository Locally

1. Navigate to your preferred drive (example: D drive):  
 cd /d  
2. Create and enter a new project folder:  
 mkdir GitDemo  
 cd GitDemo  
3. Initialize a new Git repository:  
 git init

## Step 4: Create and Commit a File

1. Create a new file with sample content:  
 echo "Welcome to Git Demo" > welcome.txt  
2. Check file status:  
 git status  
3. Stage the file:  
 git add welcome.txt  
4. Commit the file:  
 git commit -m "Added welcome.txt with welcome message"

## Step 5: Create Remote Repository on GitHub

1. Login to your GitHub account.  
2. Click 'New repository'.  
3. Name it 'GitDemo', keep it Public, and click 'Create repository'.  
4. Copy the repository URL (HTTPS).

## Step 6: Connect Local Repo to Remote Repo

1. In Git Bash, link the remote repository:  
 git remote add origin https://github.com/<your-username>/GitDemo.git  
2. Rename the branch to main:  
 git branch -M main

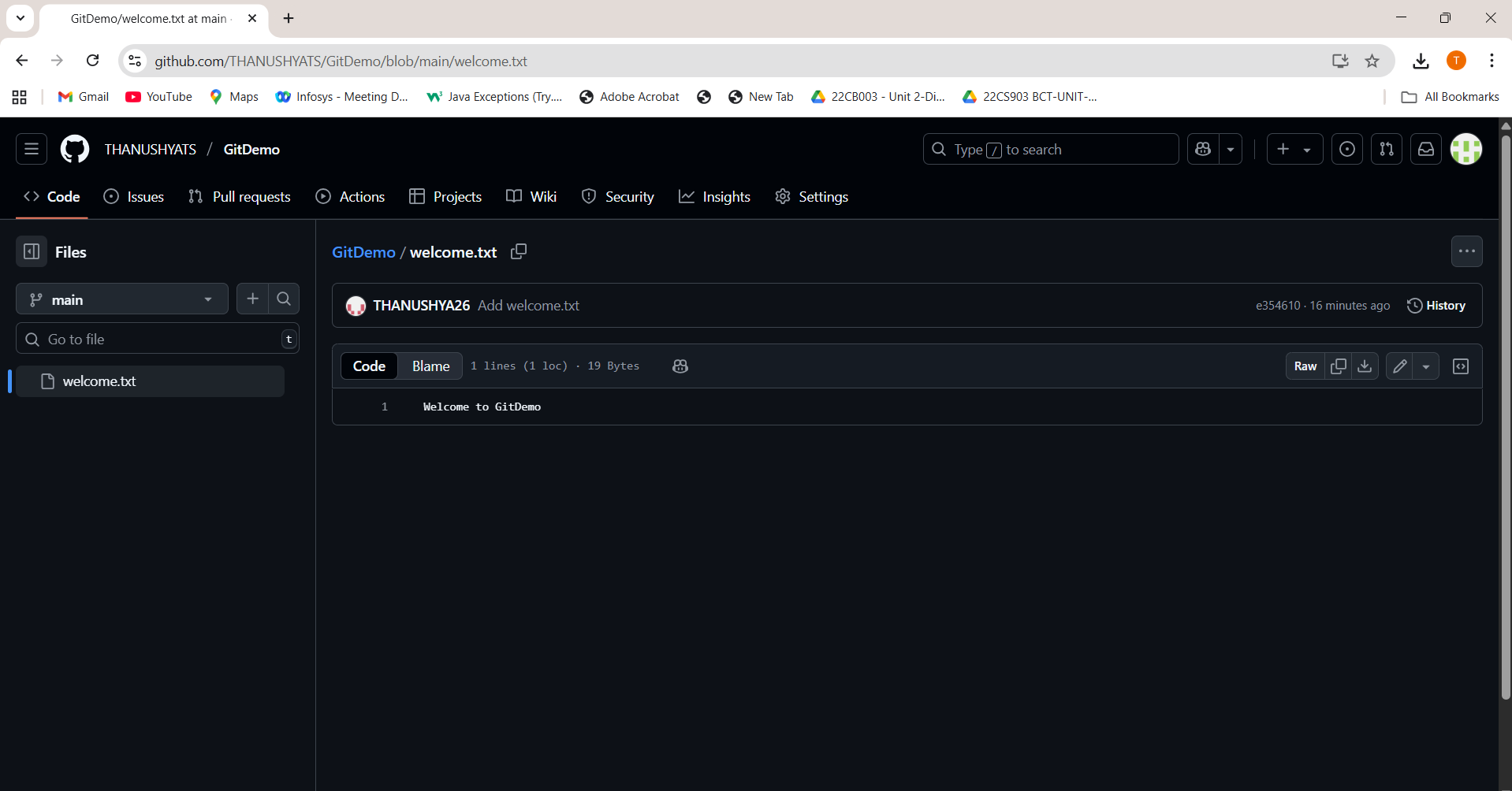
## Step 7: Setup GitHub Authentication via Personal Access Token

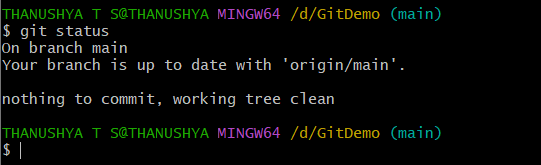
1. On GitHub, go to Settings > Developer settings > Personal Access Tokens > Tokens (classic).  
2. Click 'Generate new token (classic)'.  
3. Select 'repo' scope and generate token.  
4. Copy the token.  
5. Back in Git Bash, configure credential storage:  
 git config --global credential.helper store  
6. Push to GitHub:  
 git push -u origin main  
7. When prompted:  
 Username: your GitHub username  
 Password: paste the Personal Access Token

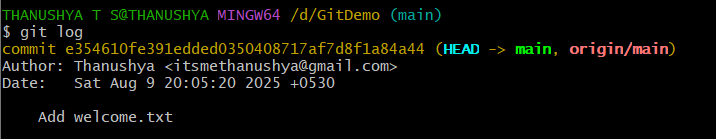
## Step 8: Verify Everything

1.Run git status — should show 'nothing to commit, working tree clean'.  
2. Run git log — should show your commit message.  
3. On GitHub, open the GitDemo repository and confirm that welcome.txt is present.

**OUTPUT:**



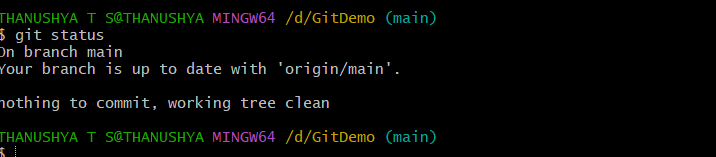




## HandsOn2:

1. 1. Open Git Bash and navigate to your Git repository:  
    cd /d/GitDemo
2. 2. Create a sample log file:  
    echo "This is a log file" > debug.log
3. 3. Create a log folder and a log file inside it:  
    mkdir log  
    echo "Log Content" > log/system.log
4. 4. Create or edit the .gitignore file:  
    notepad .gitignore
5. 5. Add the following lines to ignore all .log files and the log folder:  
    \*.log  
    log/
6. 6. Save and close the .gitignore file.
7. 7. Verify the status to ensure the ignored files are not listed:  
    git status
8. Expected Output:  
   On branch main  
   nothing to commit, working tree clean
9. 8. Commit and push any other changes if needed:  
    git add .gitignore  
    git commit -m "Added .gitignore to ignore log files and folder"  
    git push origin main

**OUTPUT:**



**HandsOn3:**

**Branching:**

1. Create a new branch named 'GitNewBranch':  
 git branch GitNewBranch

2. List all the local and remote branches:  
 git branch -a  
 (Observe the '\*' mark indicating the current branch)

3. Switch to the newly created branch:  
 git checkout GitNewBranch

4. Add a new file to this branch:  
 echo "This is a file in GitNewBranch" > branchfile.txt

5. Stage the new file:  
 git add branchfile.txt

6. Commit the changes to the branch:  
 git commit -m "Added branchfile.txt in GitNewBranch"

7. Check the status of the branch:  
 git status

**Merging:**

1. Switch back to the main branch:  
 git checkout main

2. List out all the differences between main and GitNewBranch:  
 git diff main GitNewBranch

3. (Optional) View visual differences using P4Merge tool.

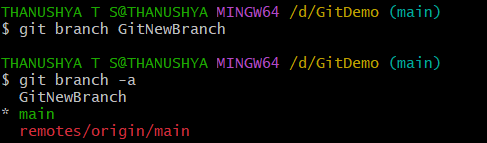
4. Merge the source branch into main:  
 git merge GitNewBranch

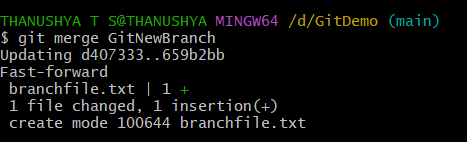
5. View the commit history after merging:  
 git log --oneline --graph --decorate

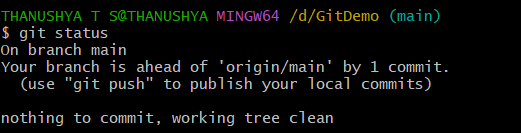
6. Delete the branch after merging:  
 git branch -d GitNewBranch

7. Check the final status:  
 git status

**OUTPUT:**







# HandsOn 4:

1. Verify if master is in a clean state:  
 git checkout main  
 git status

2. Create a new 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. Stage and commit the file:  
 git add hello.xml  
 git commit -m "Add hello.xml in GitWork"

5. Update the content of 'hello.xml':  
 echo '<message>Updated content in GitWork</message>' > hello.xml

6. Observe the status:  
 git status

7. Commit the changes:  
 git add hello.xml  
 git commit -m "Update hello.xml in GitWork"

8. Switch back to master:  
 git checkout main

9. Add a file 'hello.xml' with different content:  
 echo '<message>Hello from main branch</message>' > hello.xml

10. Stage and commit:  
 git add hello.xml  
 git commit -m "Add hello.xml in main"

11. View the commit history:  
 git log --oneline --graph --decorate --all

12. Check differences between branches:  
 git diff main GitWork

13. Use P4Merge for visual comparison.

14. Merge 'GitWork' into main:  
 git merge GitWork

15. If conflicts occur, use a 3-way merge tool to resolve them.

16. Stage resolved files:  
 git add hello.xml

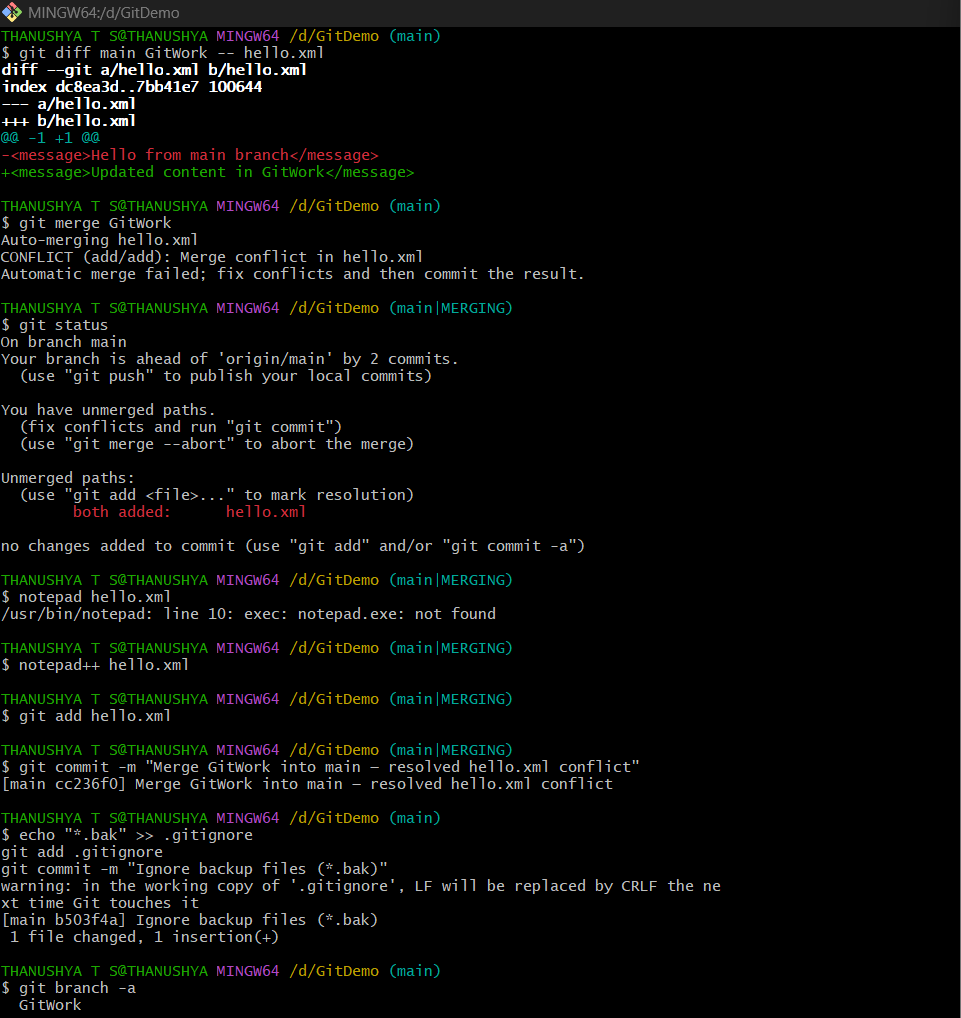
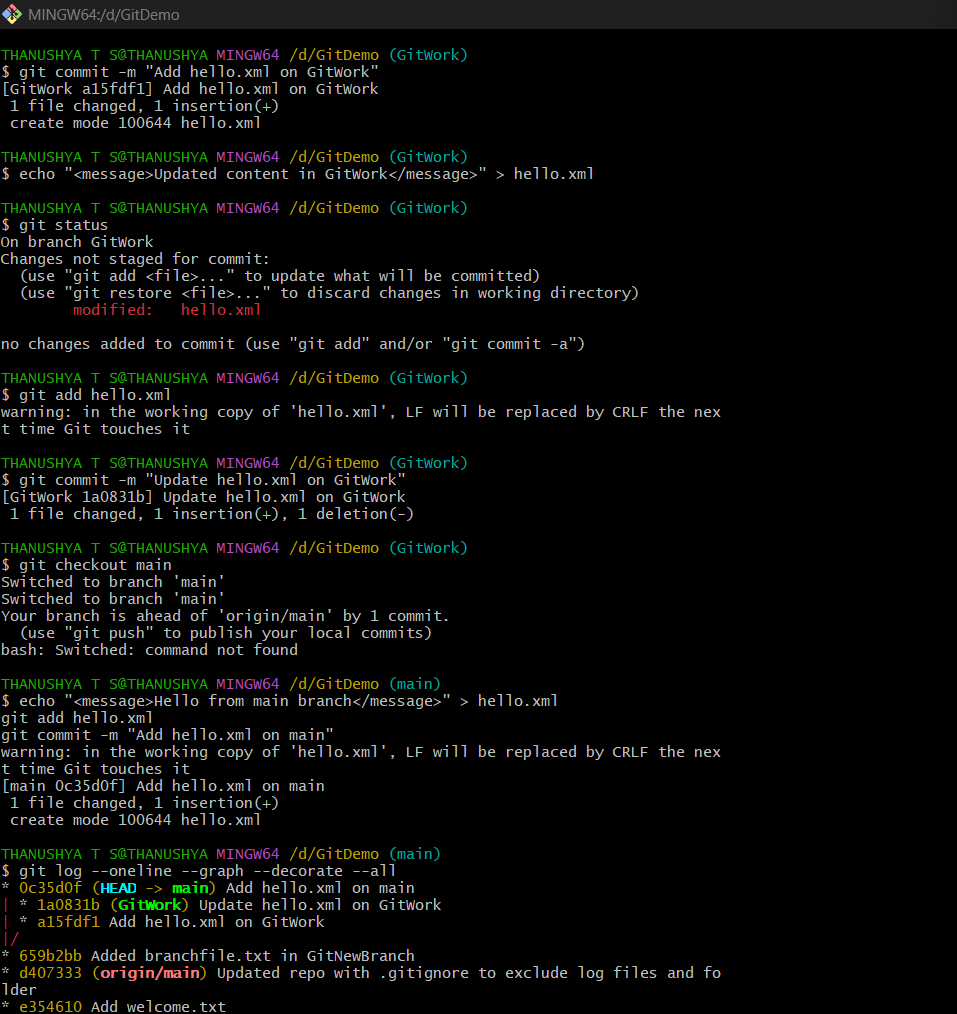
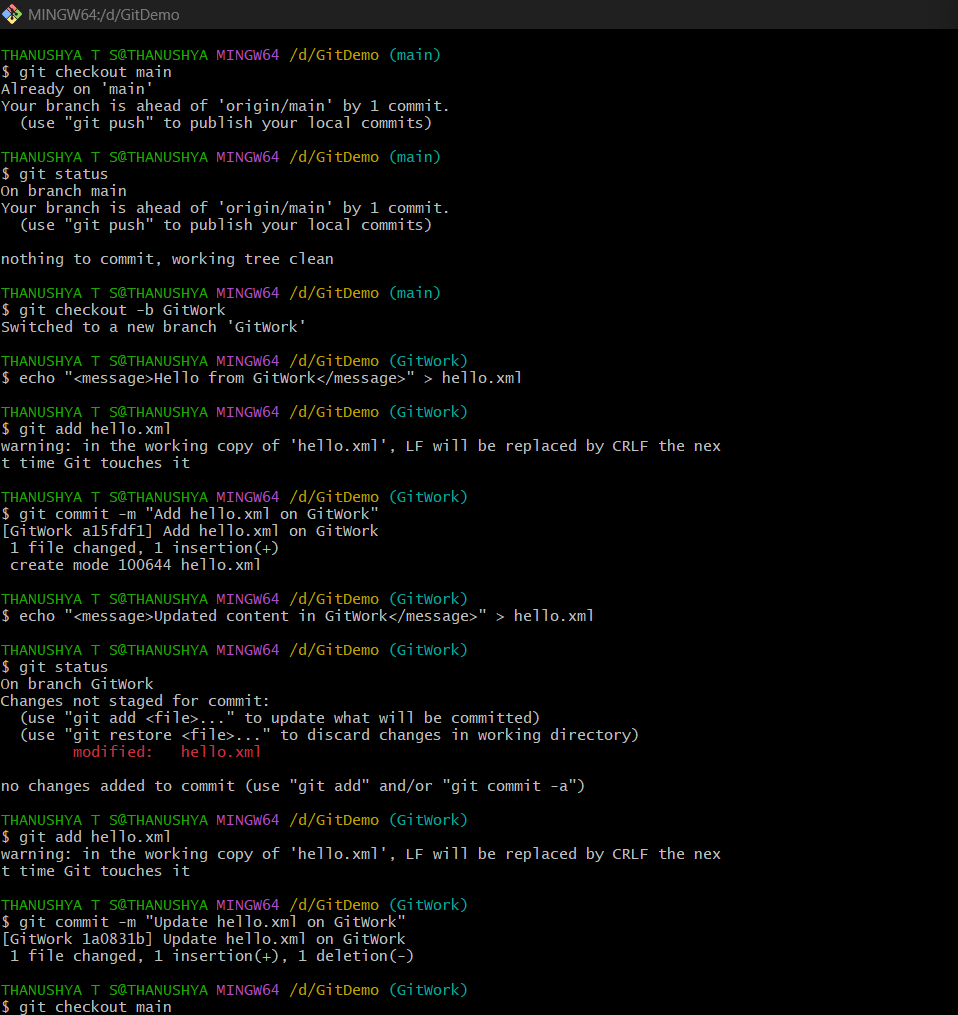
17. Commit after resolving conflicts:  
 git commit -m "Merge GitWork into main with conflict resolution"

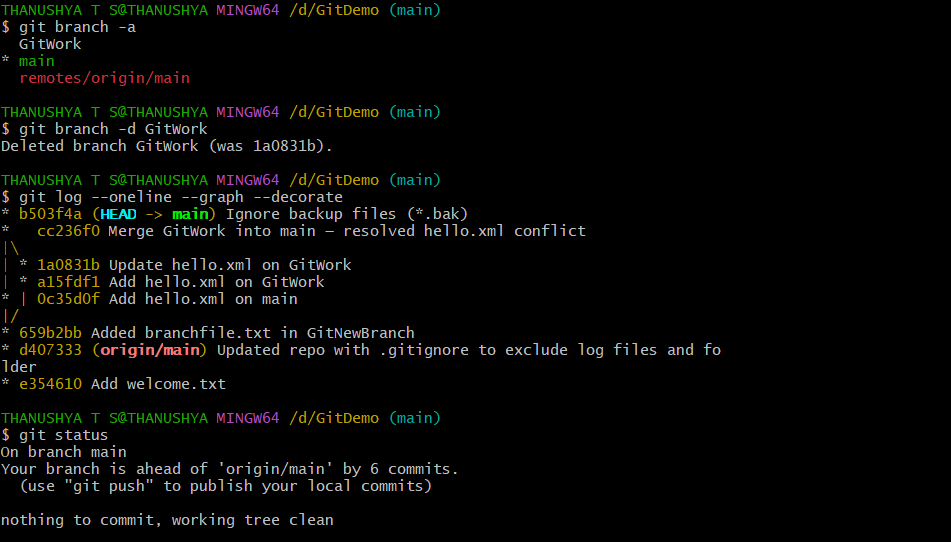
18. Add backup files to .gitignore (example: hello.xml.bak):  
 echo '\*.bak' >> .gitignore  
 git add .gitignore  
 git commit -m "Add .bak files to .gitignore"

19. List all branches:  
 git branch -a

20. Delete the merged branch:  
 git branch -d GitWork

21. View the log:  
 git log --oneline --graph –decorate

**OUTPUT:**



**HandsOn 5:**

**1. Ensure main branch is clean**

git checkout main

git status

**2. Create and switch to new branch**

git checkout -b GitWork

**3. Add hello.xml in GitWork**

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

git add hello.xml

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

**4. Update hello.xml in GitWork**

echo '<message>Updated content in GitWork</message>' > hello.xml

git add hello.xml

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

**5. Switch to main branch**

git checkout main

**6. Add hello.xml with different content in main**

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

git add hello.xml

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

**7. View commit history**

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

**8. Check differences between branches**

git diff main GitWork

**9. Merge GitWork into main**

git merge GitWork

**10. Resolve merge conflict**

* Open hello.xml
* Remove conflict markers (<<<<<<<, =======, >>>>>>>)
* Keep final desired content.

**11. Stage and commit the resolved file**

git add hello.xml

git commit -m "Merge GitWork into main with conflict resolution"

**12. Add .gitignore entry for backup files**

echo '\*.bak' >> .gitignore

git add .gitignore

git commit -m "Add .bak files to .gitignore"

**13. List all branches**

git branch -a

**14. Delete the merged branch**

git branch -d GitWork

**15. View final log**

git log --oneline --graph –decorate

**OUTPUT:**

