1. Git-HOL

**Step1: Check Git version**  
git --version  
  
**Step2: Configure Git user name and email**  
git config --global user.name "Tiyasha"  
git config --global user.email "tiyasha1630@gmail.com"  
  
**Step3: Check Git configuration**  
git config --list  
git config --global --get user.name  
git config --global --get user.email  
  
**Step4: Clone GitLab project**cd /c/Users/Tiyasha  
git clone https://gitlab.com/tiyasha1630-group/GitDemo/-/learn\_gitlab  
cd GitDemo  
  
**Step5: Add, commit, and push file**  
echo "Hello GitDemo" > README.md  
git add README.md  
git commit -m "Initial commit"  
git push origin main  
 **Step6: Check if Notepad++ is in PATH**  
which notepad++  
notepad++ --help  
  
**Step7: Set Notepad++ as default Git editor (if in PATH)**  
git config --global core.editor "notepad++ -multiInst -nosession"  
  
**Step8: Set Notepad++ as default Git editor (using full path)**git config --global core.editor "\"C:\\Program Files\\Notepad++\\notepad++.exe\" -multiInst -nosession"

**Step9: Test editor setting**  
git config --global --get core.editor  
git init testeditor  
cd testeditor  
echo "x" > a.txt

git add a.txt  
git commit

2. Git-HOL

**Step 1: Create .log file and log folder**echo "This is a log file" > test.log  
mkdir log  
echo "Log folder file" > log/example.txt  
  
**Step 2: Open/Create .gitignore and add the following lines**  
 Ignore all .log files  
\*.log  
  
 Ignore the log folder  
log/  
  
**Step 3: Verify with git status**  
git status  
  
**Step 4: Commit the .gitignore file**  
git add .gitignore  
git commit -m "Update .gitignore to ignore .log files and log folder"  
  
**Step 5: Check status again**  
git status

3. Git-HOL

Branching:  
**Step 1. Create a new branch**git branch GitNewBranch  
  
**Step 2. List all local and remote branches**  
git branch -a  
  
**Step 3. Switch to the new branch**  
git checkout GitNewBranch  
  
**Step 4. Add some files**  
echo "File for GitNewBranch" > branchfile1.txt  
echo "Another file for GitNewBranch" > branchfile2.txt  
  
**Step 5. Commit the changes**  
git add branchfile1.txt branchfile2.txt  
git commit -m "Added files in GitNewBranch"  
  
**Step 6. Check status**  
git status

Merging:  
**Step 1. Switch to master**  
git checkout master  
  
**Step 2. List all differences between master and GitNewBranch**  
git diff GitNewBranch  
  
**Step 3. Visual differences using P4Merge (requires configuration)**  
git difftool -t p4merge master GitNewBranch  
  
**Step 4. Merge the branch into master**  
git merge GitNewBranch  
  
**Step 5. View the log after merging**  
git log --oneline --graph --decorate  
  
**Step 6. Delete the branch and check status**  
git branch -d GitNewBranch  
git status

4. Git-HOL

**Step 1. Verify if master is in clean state**  
git status  
  
**Step 2. Create a branch “GitWork” and switch to it, add hello.xml**git checkout -b GitWork  
echo "<message>Hello from GitWork branch</message>" > hello.xml  
git add hello.xml  
  
**Step 3. Update the content of “hello.xml” and observe the status**  
echo "<message>Updated content in GitWork branch</message>" > hello.xml  
git status  
  
**Step 4. Commit the changes to reflect in the branch**  
git commit -am "Updated hello.xml in GitWork branch"  
  
**Step 5. Switch to master**git checkout master  
  
**Step 6. Add a file “hello.xml” to master with different content**echo "<message>Hello from master branch</message>" > hello.xml  
git add hello.xml  
  
**Step 7. Commit the changes to the master**  
git commit -m "Added hello.xml in master branch"  
  
**Step 8. Observe the log**  
git log --oneline --graph --decorate --all  
  
**Step 9. Check differences**  
git diff GitWork master  
  
**Step 10. Use P4Merge to see differences (assuming p4merge is configured as mergetool)**git mergetool --tool=p4merge  
 **Step 11. Merge the branch into master**  
git merge GitWork  
  
**Step 12. Observe the git mark up (conflict markers will appear if conflicts exist)**cat hello.xml  
 **Step 13. Use 3-way merge tool to resolve the conflict**  
git mergetool  
  
**Step 14. Commit the changes after resolving conflicts**  
git commit -am "Merge GitWork into master with conflict resolved"  
  
**Step 15. Observe git status and add backup file to .gitignore**  
git status  
echo "\*.bak" >> .gitignore  
  
**Step 16. Commit the changes to .gitignore**  
git add .gitignore  
git commit -m "Added .gitignore entry for backup files"  
  
**Step 17. List all branches**  
git branch -a  
  
**Step 18. Delete the branch that was merged to master**  
git branch -d GitWork  
  
**Step 19. Observe the log**  
git log --oneline --graph --decorate

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**  
git checkout master  
git pull origin master  
  
**Step 4. Push the changes from “Git-T03-HOL\_002” to the remote repository**git checkout Git-T03-HOL\_002  
git push origin Git-T03-HOL\_002  
  
**Step 5. Observe if the changes are reflected in the remote repository**  
git log --oneline --graph --decorate --all