Cognizant - DN 4.0 Deep Skilling Java FSE

Week 08 - Git

Superset ID: 6384852

Name: BHAVANA REDDY M

**Exercise 1: Git Commands**

**Step 1: Setup your machine with Git Configuration**

To create a new repository, signup with GitLab and register your credentials

Login to GitLab and create a “demo” project

1. To check if Git client is installed properly: Open Git bash shell and execute

If output shows Git with its version information that indicates, that Git Client installs properly.

1. To configure user level configuration of user ID and email ID execute
2. To check if the configuration is properly set, execute the following command.

**Step 2: Integrate notepad++.exe to Git and make it a default editor**

1. To check, if notepad++.exe execute from Git bash

If Git bash could not able to recognize notepad++ command that implies notepad++.exe is note added to the environment path variable.

To add path of notepad++.exe to environment variable, go to control panel -> System -> Advanced System settings. Go to Advanced tab -> Environment variables -> Add path of notepad++.exe to the path user variable by clicking on “Edit”

**Implementation:**

git version

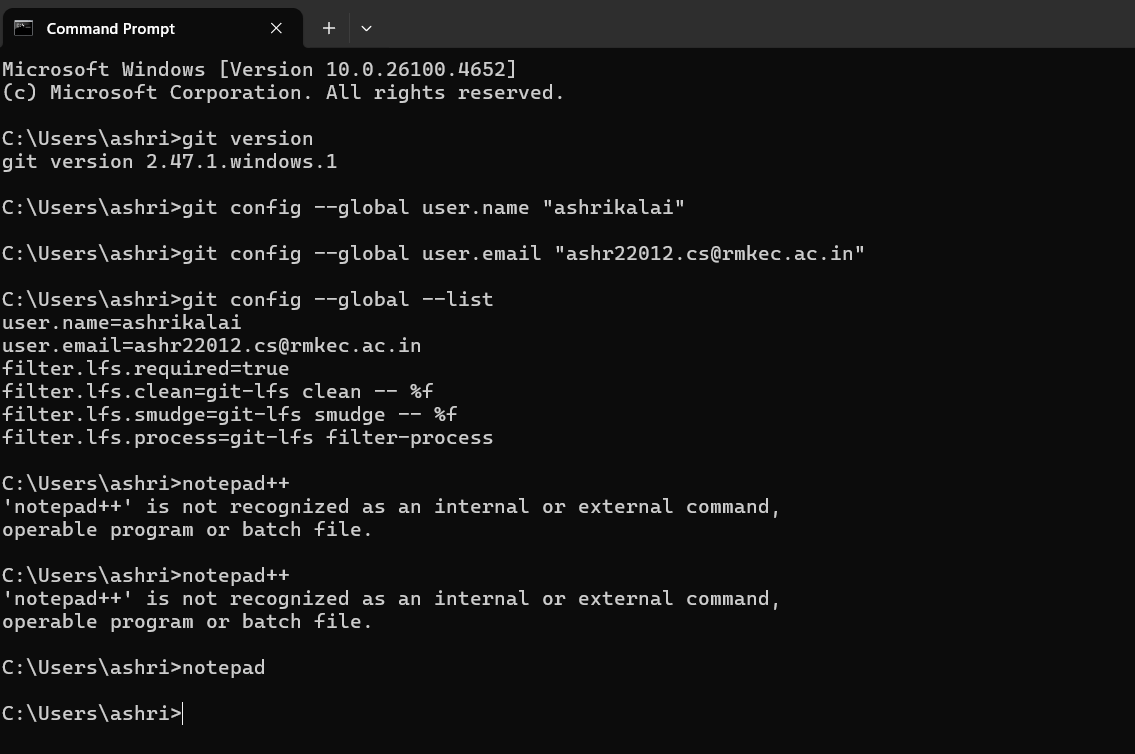
git config --global user.name "ashrikalai"

git config --global user.email [ashr22012.cs@rmkec.ac.in](mailto:ashr22012.cs@rmkec.ac.in)

git config --global –list

notepad

**Execution Window:**

****

**Exercise 2: Git Log Command**

**.gitignore:**

* .gitignore is a special file in a Git repository that tells Git which files or folders to **ignore**.
* Any files matching the patterns in .gitignore will **not** be tracked, staged, or committed to the repository.
* This helps keep your repository clean and free from unnecessary files such as temporary data, build outputs, log files, or sensitive information.

.gitignore

# Ignore all .log files

\*.log

# Ignore 'log' folder and its contents

log/

**Commands:**

git config --global core.editor "notepad.exe"

git init

notepad .gitignore

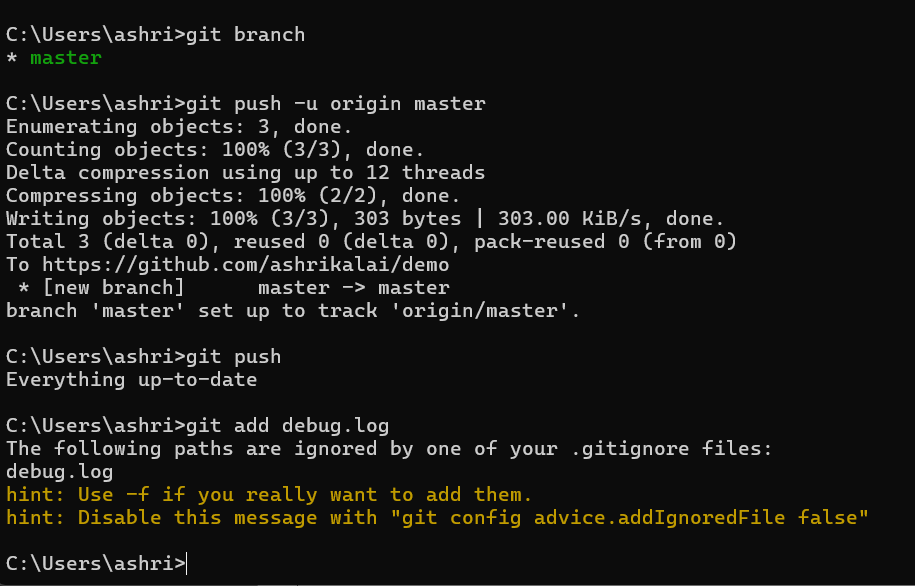
git add .gitignore

git push origin master

git push

git add debug.log

**Execution Window:**



**Exercise 3: Branch and Merge**

**Branching and Merging**

* **Branching** in Git allows you to create a separate line of development from the main project.  
  You can work on new features, bug fixes, or experiments without affecting the main codebase.
* **Merging** is the process of combining changes from one branch into another.  
  After work in a branch is complete, merging integrates it back into the main branch (often called master or main).

**Creating a Branch Request in GitLab**

* In GitLab, a **branch request** usually means creating a **new branch** in the remote repository so that other team members can collaborate on it.
* This can be done from:
  + **GitLab Web UI** → Repository → Branches → New branch.
  + **Git Commands**:

git checkout -b feature-branch

git push -u origin feature-branch

**Creating a Merge Request in GitLab**

* A **Merge Request (MR)** in GitLab is a formal request to merge changes from one branch into another (e.g., feature-branch → master).
* It allows code review, discussion, and approval before merging.
* Steps in GitLab:
  1. Push your branch to GitLab.
  2. Go to the repository in GitLab and click **Merge Requests** → **New Merge Request**.
  3. Select **source branch** and **target branch**.
  4. Add a title, description, and reviewers.
  5. Submit the merge request for review and approval.

**Git commands:**

git checkout -b GitNewBranch

git branch -a

echo "This is a new file in GitNewBranch" > branchfile.txt

git add branchfile.txt

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

git status

git checkout master

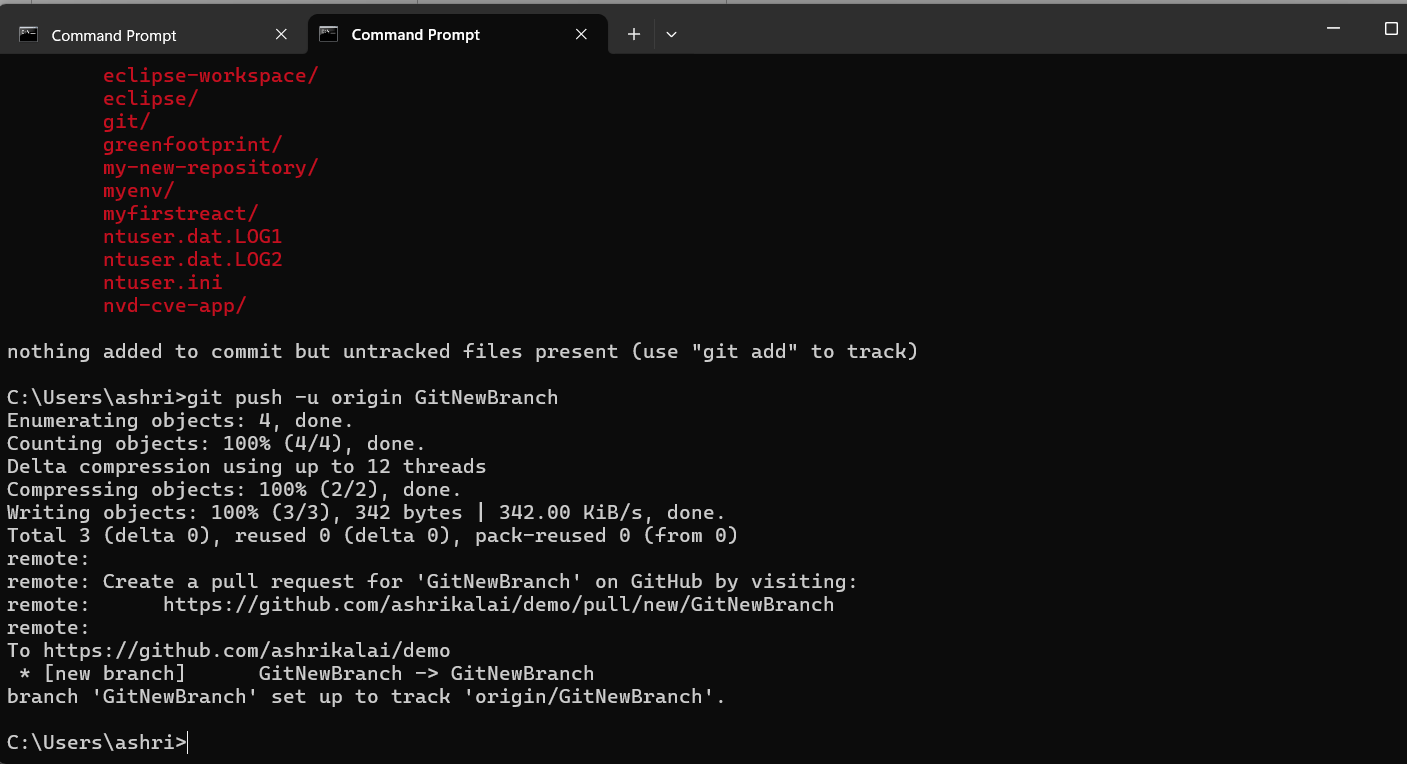
git diff master GitNewBranch

git merge GitNewBranch

git log --oneline --graph –decorate

git branch -d GitNewBranch

**Execution Window:**



**Exercise 4: Merge Conflict**

**Merge Conflict**

A merge conflict occurs when **two commits modify the same part of a file** and Git can’t automatically decide which change to keep.  
Example:

* Branch master changes line 5 of hello.xml.
* Branch GitWork also changes line 5 of hello.xml but with different content.

When merging, Git will ask you to resolve it manually.

**Git commands:**

git checkout master

git status

git checkout -b GitWork

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

git add hello.xml

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

git checkout master

echo "<message>Hello from master</message>" > hello.xml

git add hello.xml

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

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

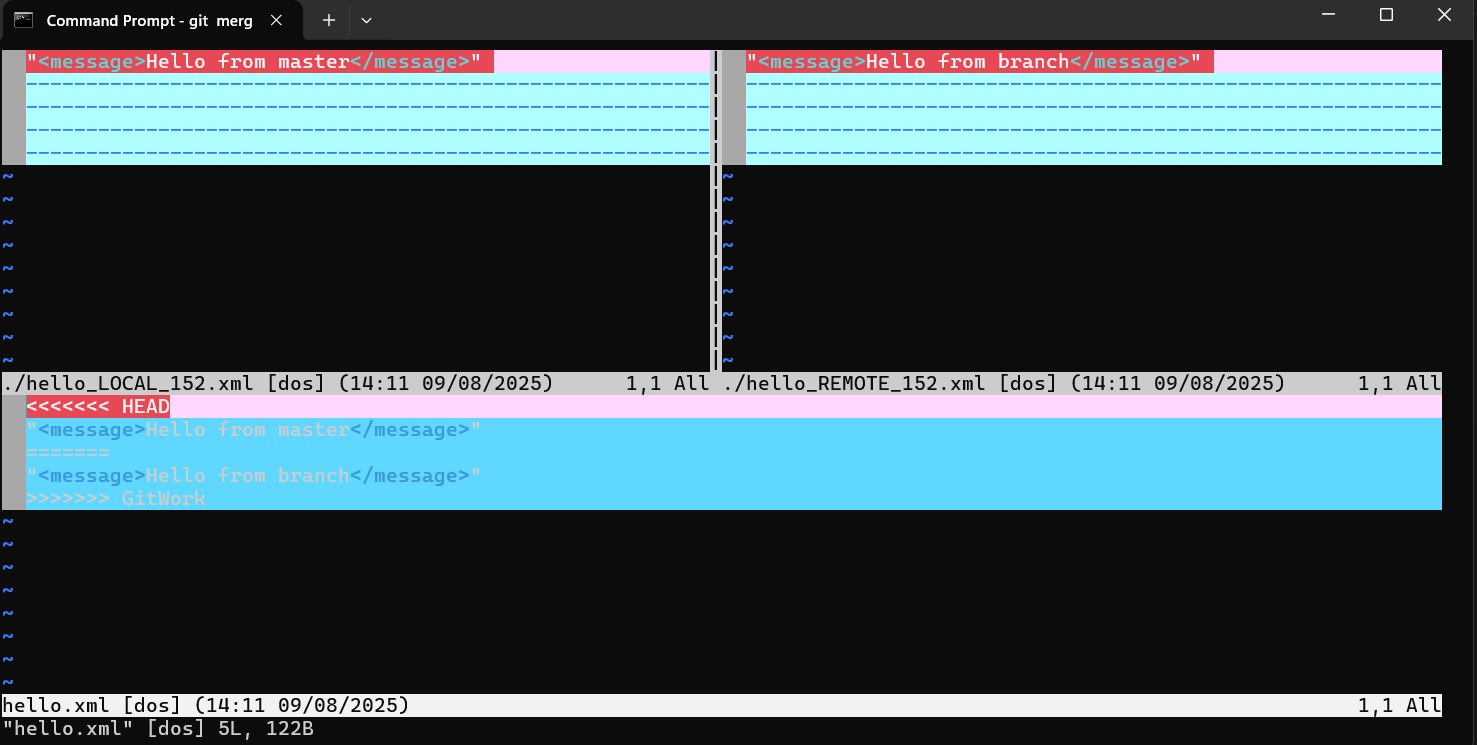
git merge GitWork

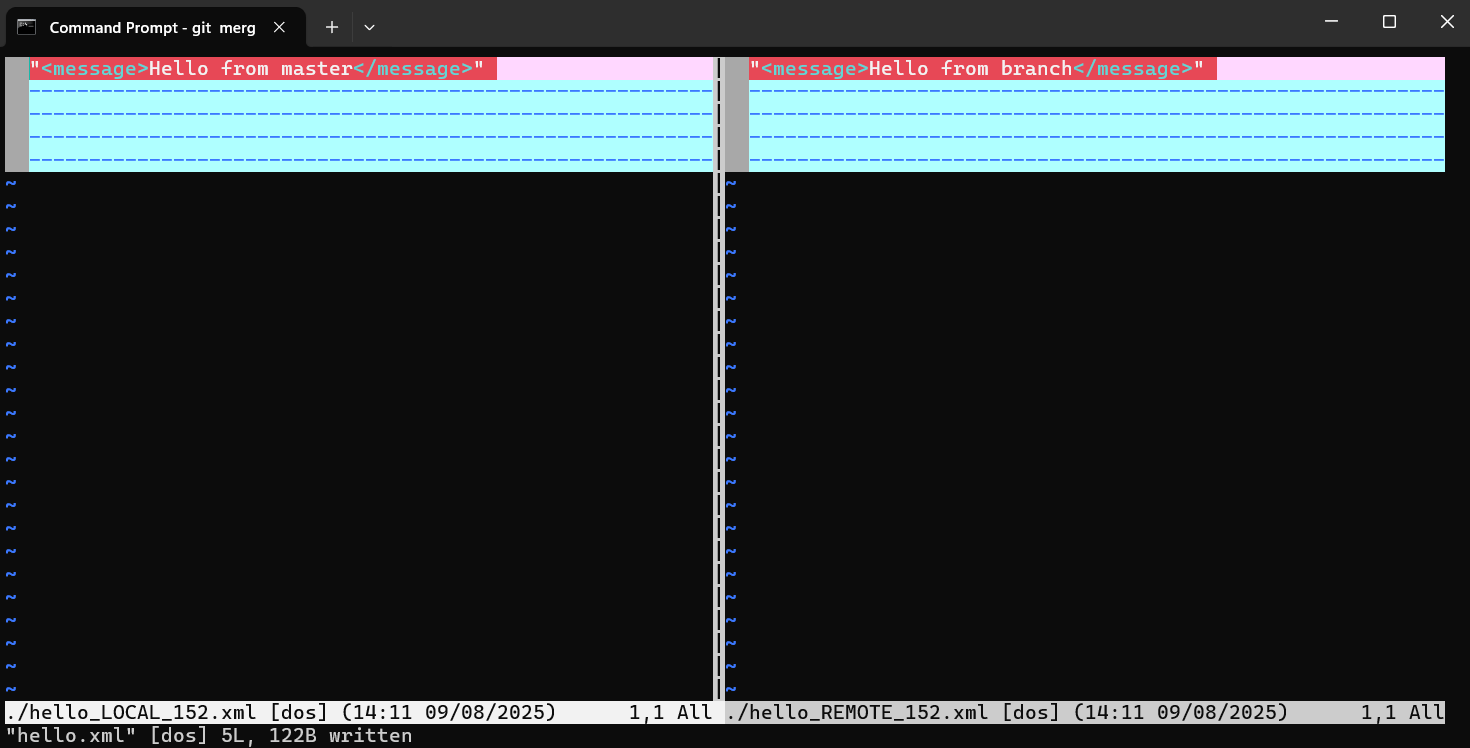
git mergetool

git add hello.xml

git commit -m "Resolved merge conflict between master and GitWork"

**Execution Window:**





**Exercise 5: Clean and Push**

In Git, “clean up” means ensuring that your local repository is in a stable and consistent state before sending changes to a remote repository.  
This typically involves:

* Making sure you are on the correct branch (e.g., master or main)
* Committing or stashing uncommitted changes
* Synchronizing with the remote repository to avoid conflicts

Once cleaned up, you “push back” changes to the remote repository so that others can access your updated work.

**Git commands:**

git checkout master

git status

git branch -a

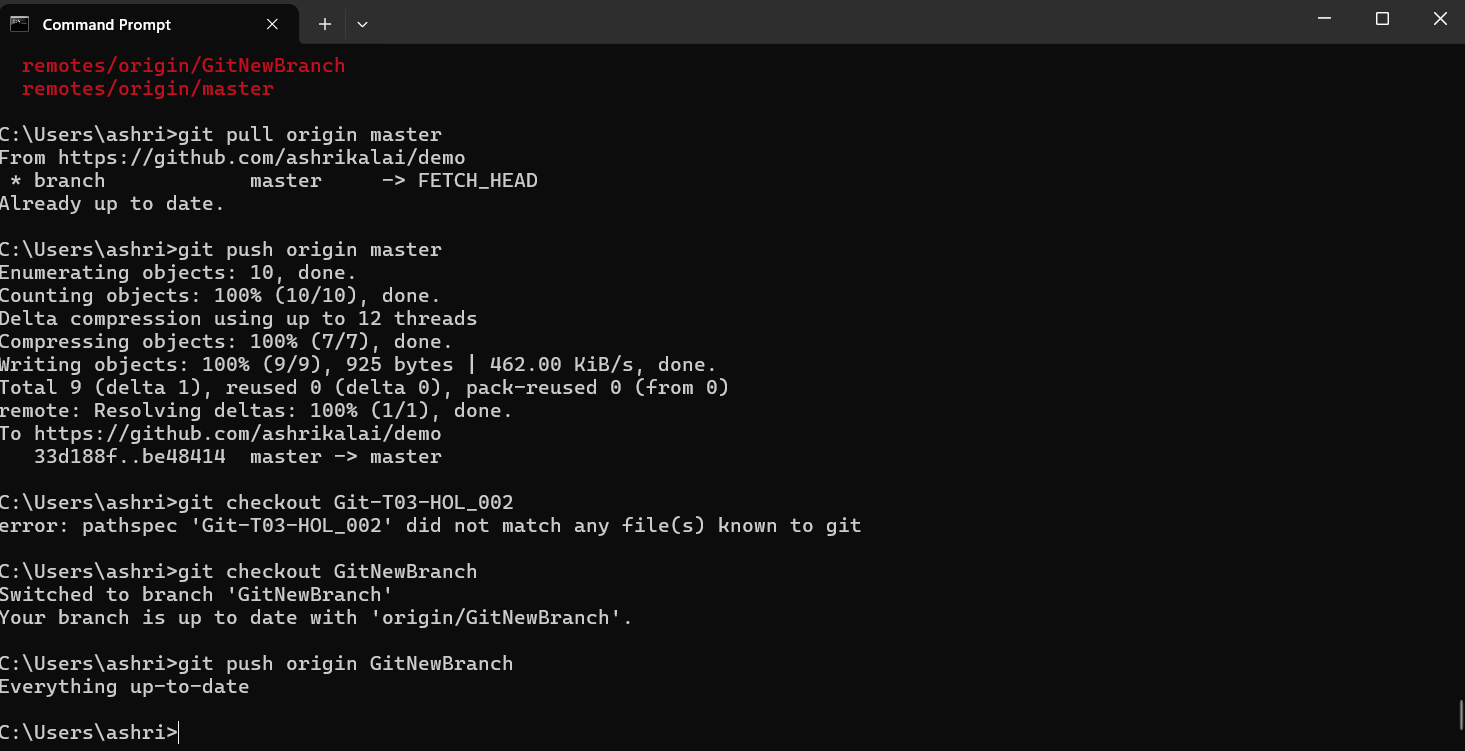
git pull origin master

git push origin master

git checkout GitNewBranch

git push origin GitNewBranch

**Execution Window:**

****