**HANDSON WEEK-8**

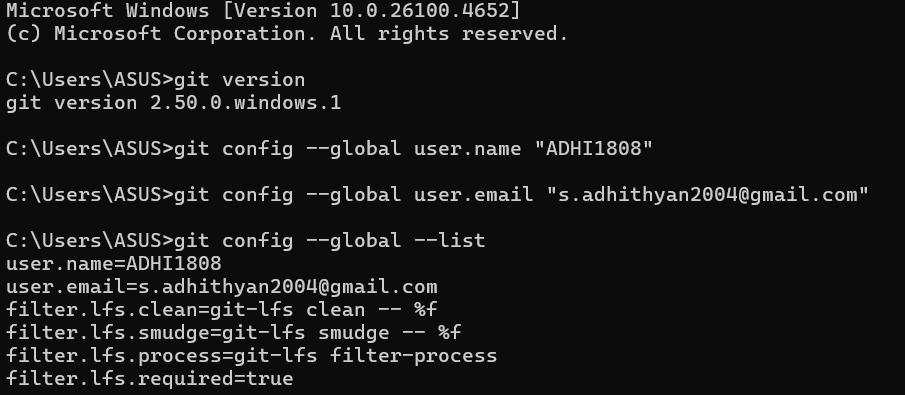
**Module- Git**

**Ex1: Git-HOL**

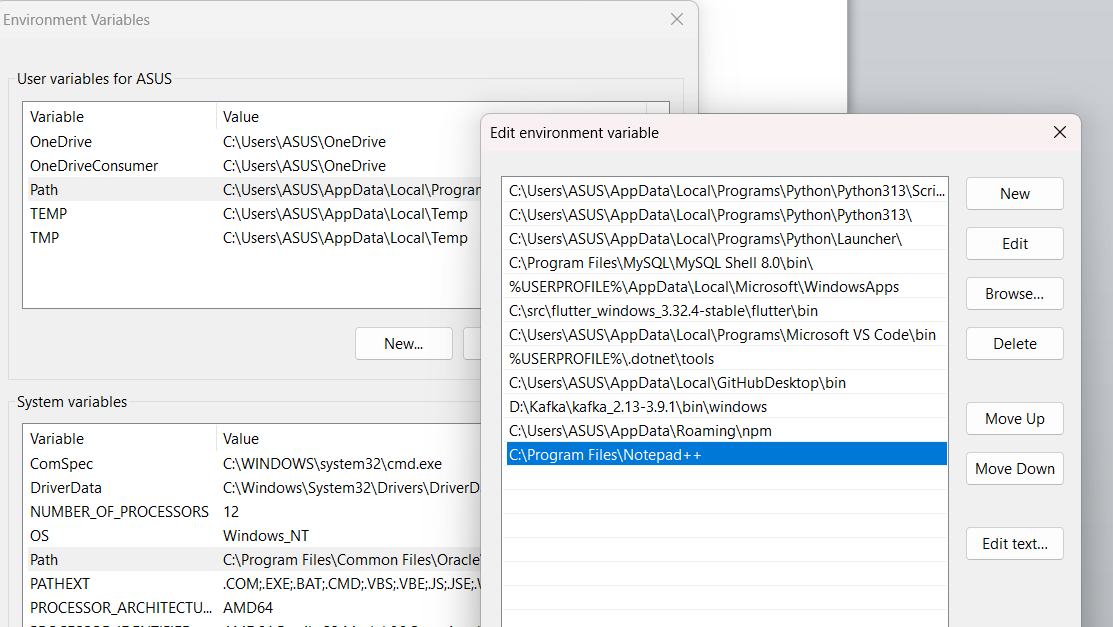
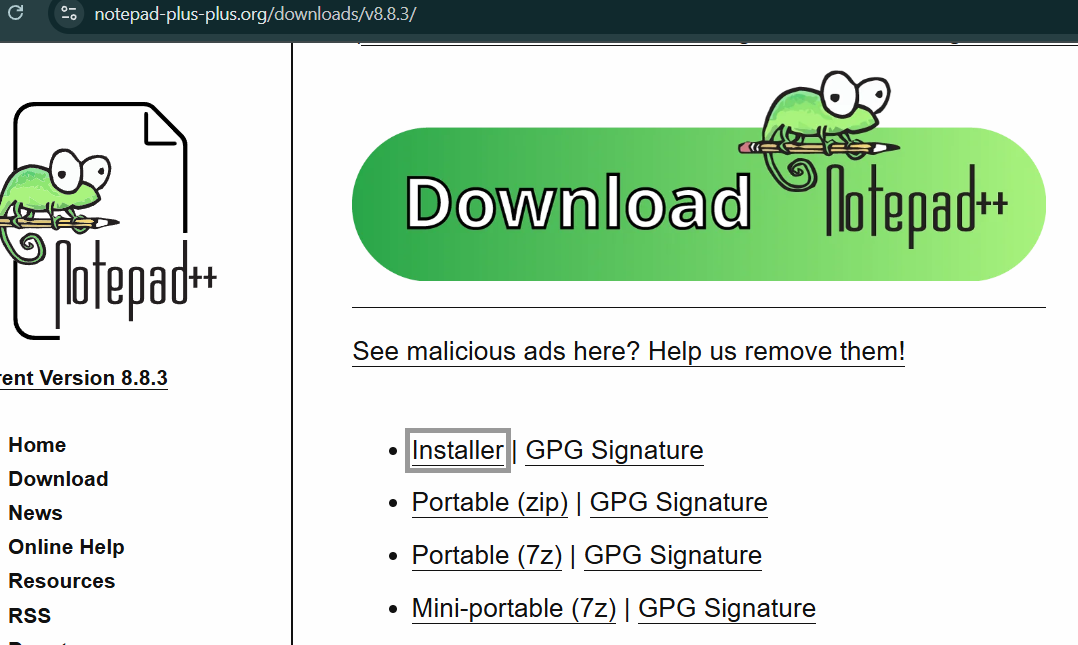
* Setup your machine with Git Configuration
* Integrate notepad++.exe to Git and make it a default editor
* Add a file to source code repository

Step1:Install Git in your system and complete setup

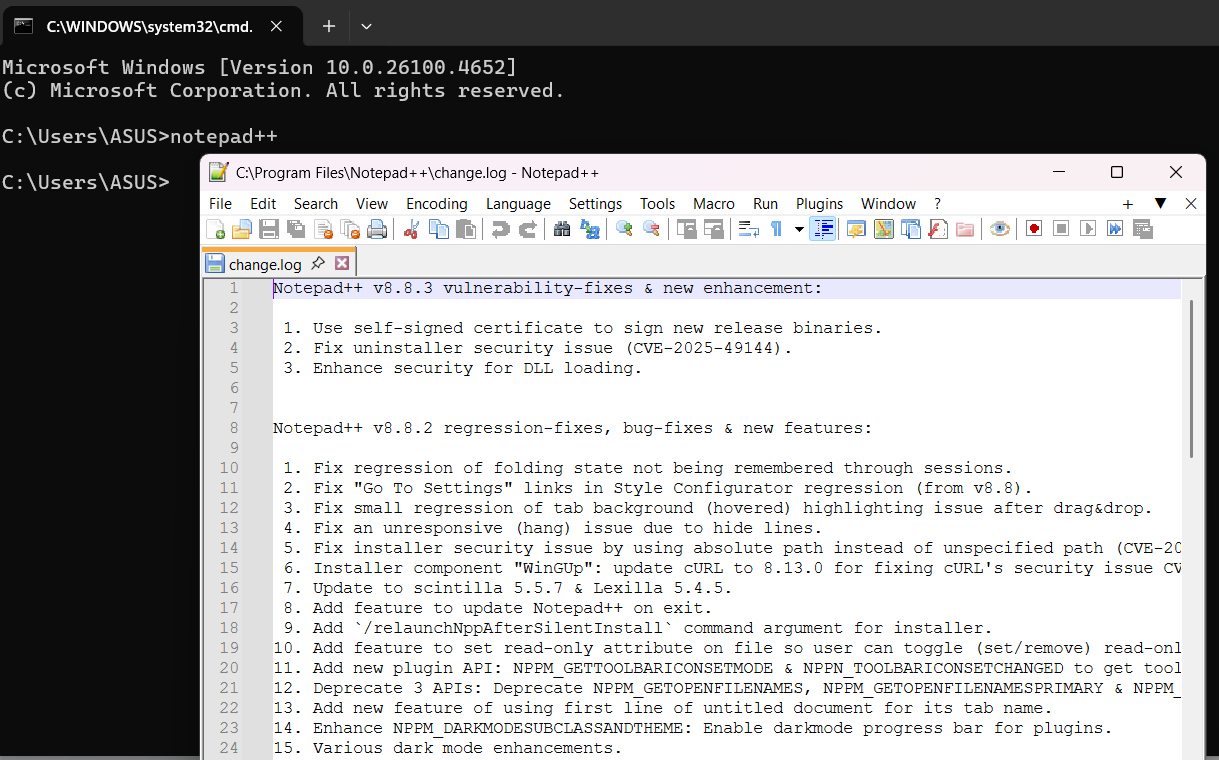
Step2:SignUp with your Git la using user Name and Email



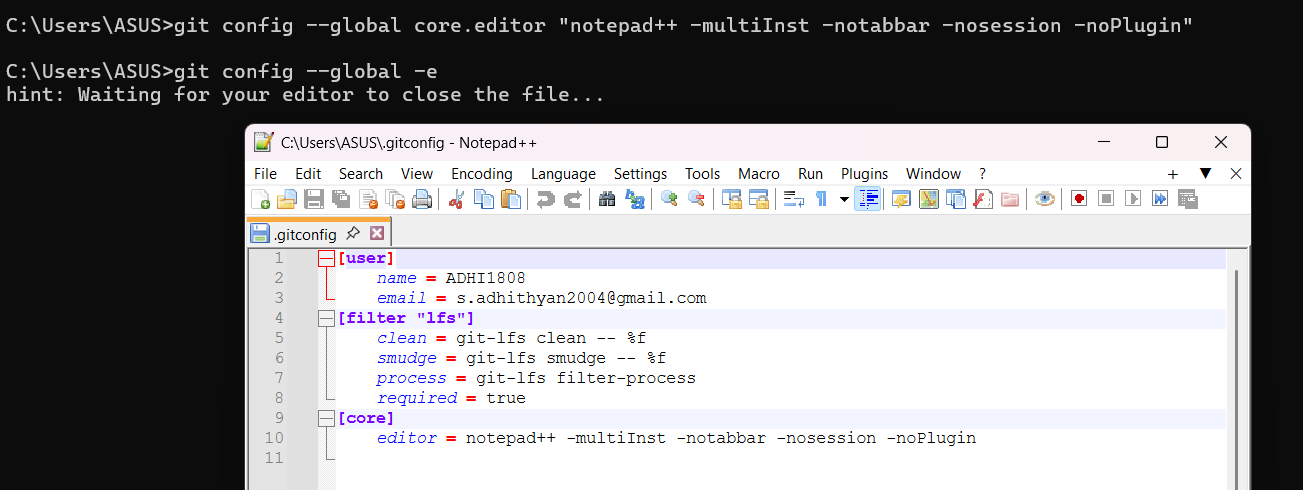
Step3: Install “Notepad++” and complete setup by adding the path to Enviroinment

****

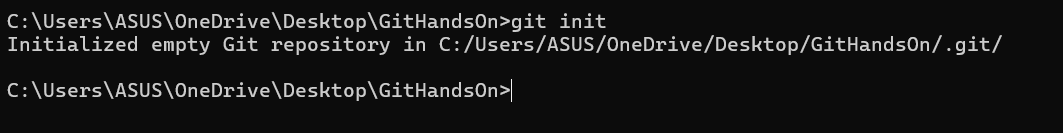
Step4:Chek the setup is working by running “Notepad++” in cmd to open the application

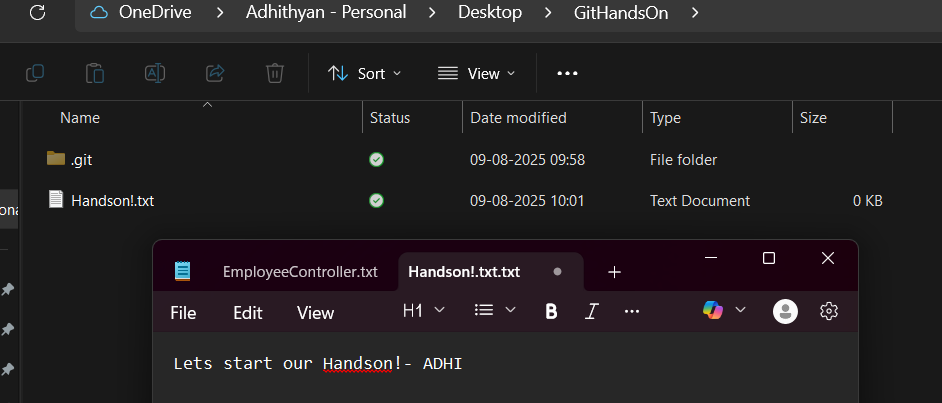


Step5: Set Notepad++ as default Git editor

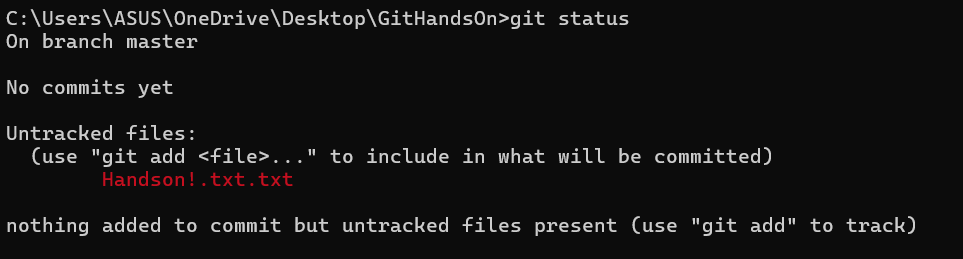


Step6: Create a empty folder “GitHandsOn” and initial git to it through its cmd

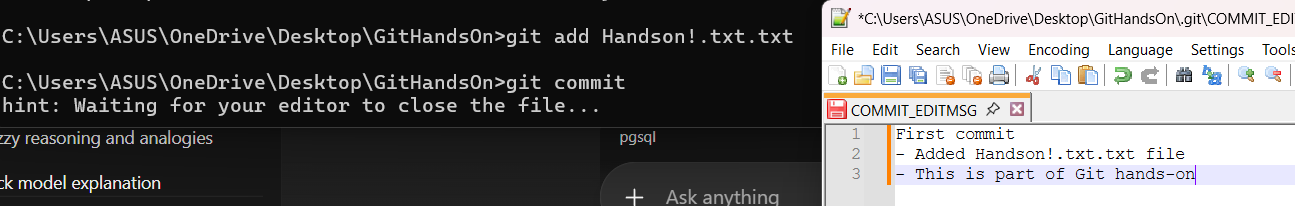


Step7: now create a text file “Handson1.txt” inside the folder

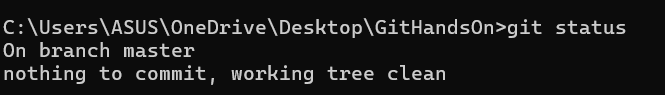
Step8: Now while checking Status of “GitHandsOn” folder we can see



Step9: Now commite the changes we made



Step10: now check status we can see no untrackedFiles



**Ex2: Git-HOL**

**Gitignore File:**

Simple text file that tells Git which files or folders to intentionally ignore in a project.This prevents unnecessary or sensitive files from being included in your project's history.

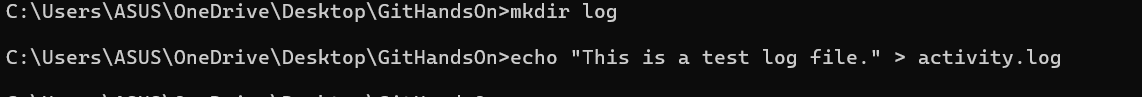
**How to Ignore Unwanted Files Using .gitignore**

Ignoring files with .gitignore is a straightforward process that involves creating the file and adding patterns to it.

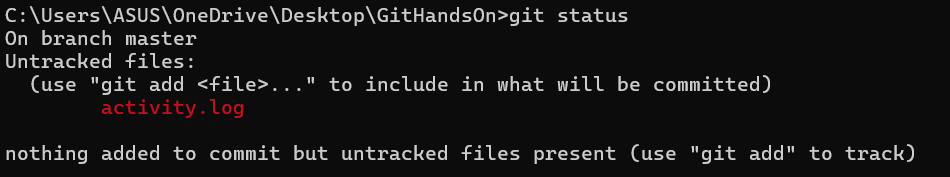
* **Create a .gitignore File:** In the root directory of your project, create a new file and name it .gitignore.
* **Add File or Folder Names:** Open the .gitignore file and add the names of the files or folders you want to ignore, with each entry on a new line. For example, to ignore a file named debug.log, you would simply add debug.log to the file. To ignore an entire directory called build/, you would add build/.
* **Use Patterns for Broader Matches:** You can use special characters called wildcards to ignore multiple files at once. For instance, to ignore all files that end with .log, you would add \*.log to your .gitignore file.

Implement git ignore command to ignore unwanted files and folders

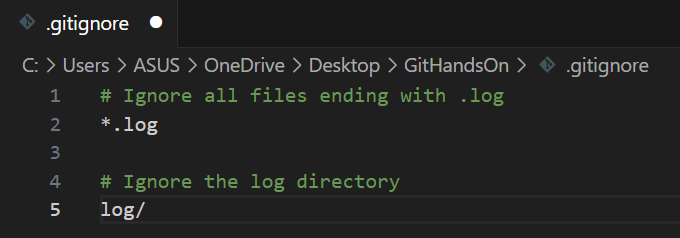
Step1: Create Required file and folder for testing. Log folder and sample log file



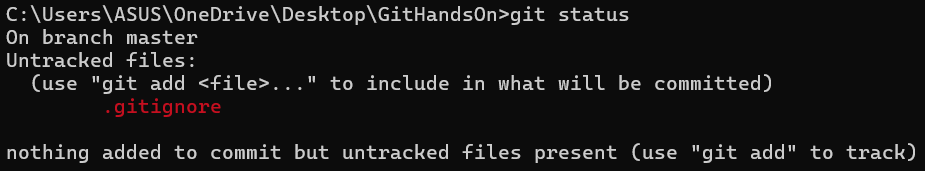
Step2: Check your repository's status to see that Git is tracking the new file and folder.



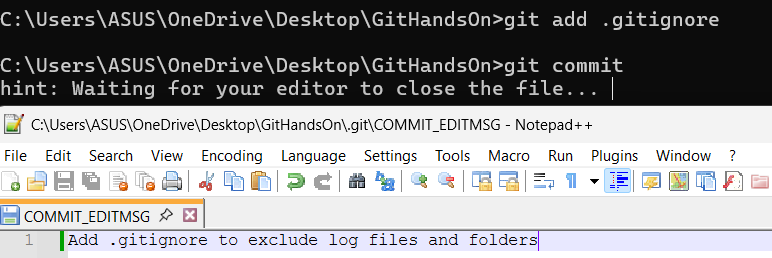
Step3: Create and Update the rule in .gitignore File

****

Step4: Now verify git status and see that You will see that activity.log and the log/ folder are no longer listed.



Step5: Add and commit the .gitignore file so that these rules are shared with anyone else who works on the project.



**Ex3: Git-HOL**

**Git Branching and Merging**

Branching and merging are fundamental Git features that allow developers to work on different things in parallel without interfering with each other.

* Branching = Create a separate copy of the project to develop features, fix bugs, or experiment without affecting the main branch (main/master).
* You work on the branch independently. After creating a branch, you can make commits to it without affecting the master branch. This allows you to work on a new feature over several days, while other developers can continue working on the main codebase without being disrupted by your unfinished work.
* Merging = Combine your branch changes back into the main branch after testing, making them part of the main codebase.

**Creating a Branch in GitLab**

Creating a branch in GitLab is how you start a new line of work within a GitLab-hosted repository.

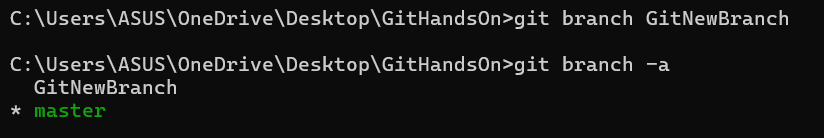
* In GitLab, you can create a branch from the repository page by typing a new branch name; it copies the current branch.
* Create branches for specific tasks, naming them descriptively (e.g., fix-login-bug).
* Branches made in GitLab are visible to everyone for collaboration.

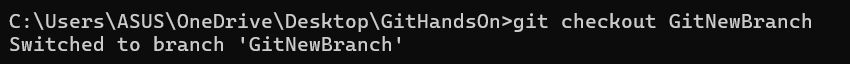
**Creating a Merge Request in GitLab**

A Merge Request (MR) in GitLab is a formal way to ask for your branch's changes to be merged into another branch (like master). This is also known as a "Pull Request" in other platforms like GitHub.

* Merge Request (MR) starts a review process where reviewers can comment and suggest changes.
* Shows a summary of commits and a detailed diff of changes.
* Allows automated tests and safe merging after approval.

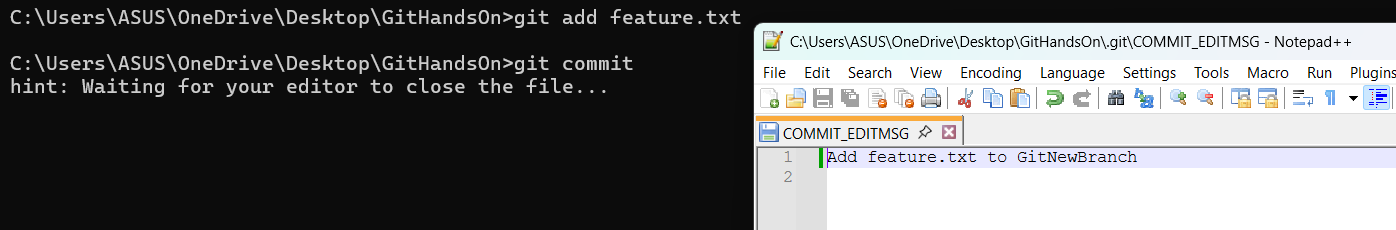
Construct a branch, do some changes in the branch, and merge it with master (or trunk)

Step1: Create new Branch “GitNewBranch” and switch to it

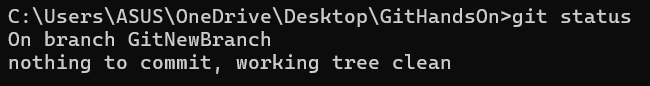


Step2: Create a new file with some text inside new Branch and commit it

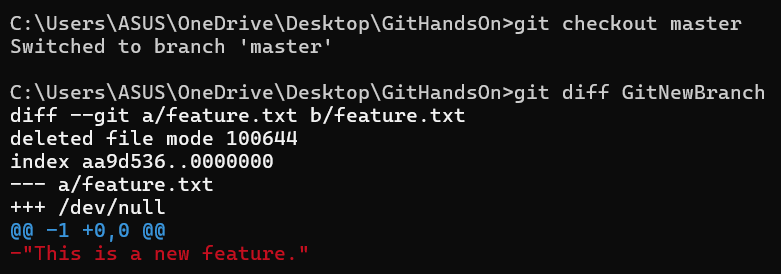




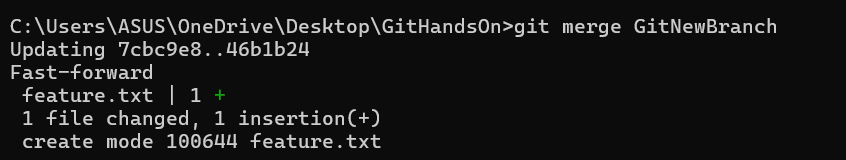
Step3: Noe check status of git and its clear



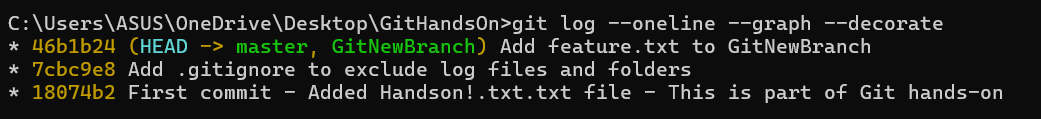
Step4: Before merging, you must be on the branch you want to merge *into* and List out all the differences between trunk and branch



Step5: Now merge them



Step6: Use the specified git log command to view the commit history in a compact, graphical format



Step7: Delete the branch after merging and observe the git status.





**Ex4: Git-HOL**

A **Merge conflict** occurs when two branches change the same part of a file and Git can’t decide which version to keep.

1.Identify the Conflict

First, Git will stop the merge process and tell you which file (or files) has a conflict. You can also use the git status command to see a list of "unmerged paths," which are the files with conflicts.

2.Manually Edit the File

Edit the file to fix the conflict. need to decide what the final version of the code should look like.

* Choose one version: might decide to keep only your changes and delete the other version.
* Keep the other version: might decide the other branch has the correct code and delete yours.
* Combine both: can manually edit the code to incorporate parts of both changes.

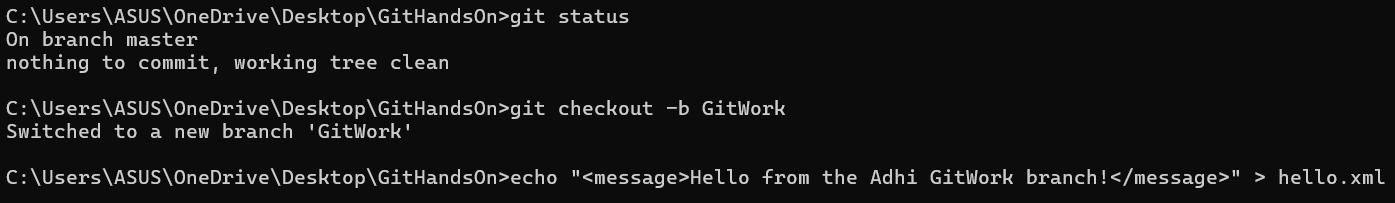
3. Commit the Fix

Once fixed all the conflicts in the file (or files), need to tell Git that the conflict is resolved.

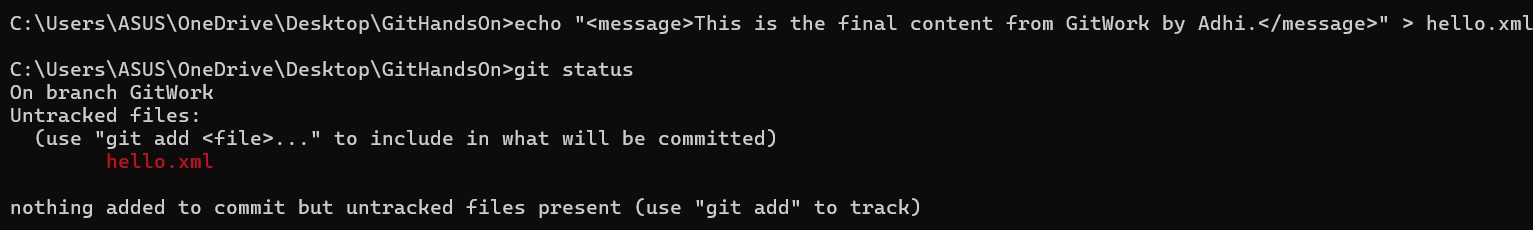
* Stage the fixed file: Use the git add <filename> command for each file fixed. This tells Git that have resolved the conflict in that file.
* Commit the merge: Once all conflicted files are staged, run git commit. Git will open a pre-populated commit message (e.g., "Merge branch 'new-feature'"). can just save this message to complete the merge.

Implement conflict resolution when multiple users are updating the trunk (or master) in such a way that it results into a conflict with the branch’s modification.

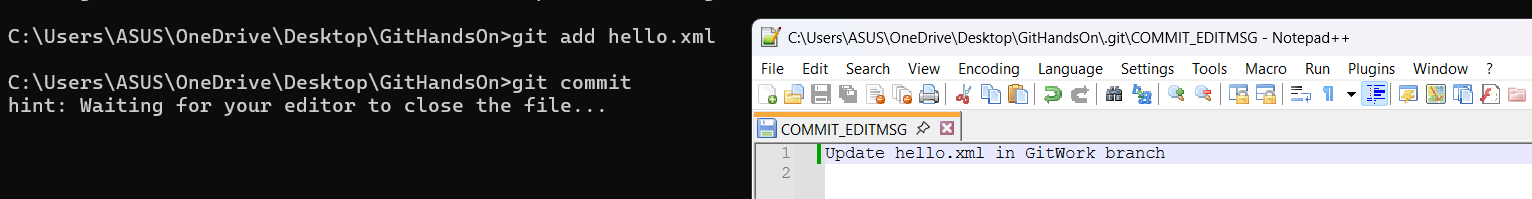
Step1: Check the status of main branch is clean, and create new branch “GitWork” and add a file “hello.xml”



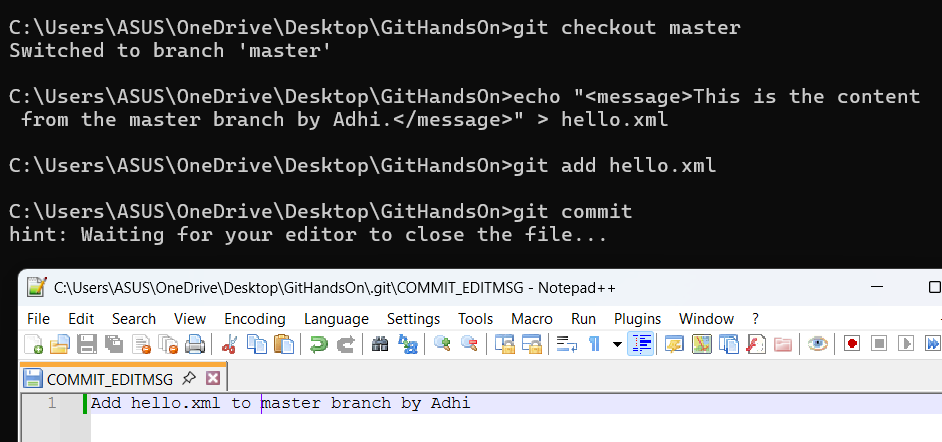
Step2: Update the content in hello.xml and check the status



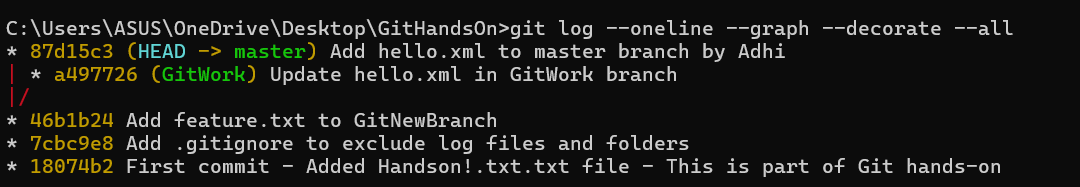
Step3: Commit the changes to reflect in the branch



Step4: Switch back to main brnch and add hello.xml and commit



Step5: Observe the log.The below command shows the history of both branches, illustrating how they have diverged.

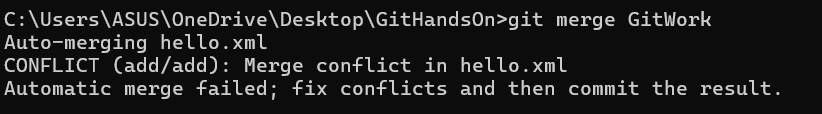


Step6: Check the differences with Git diff tool

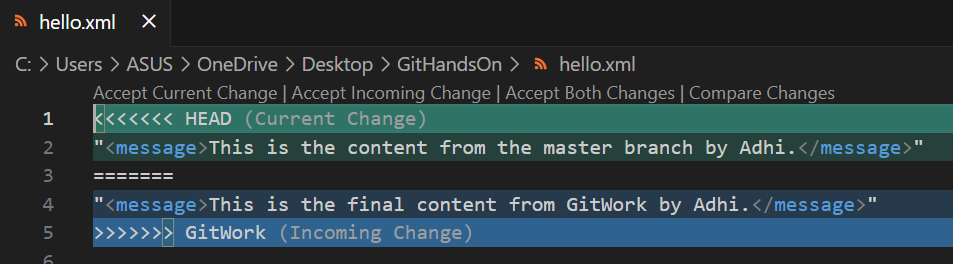
Run “git diff GitWork” and “git difftool GitWork”



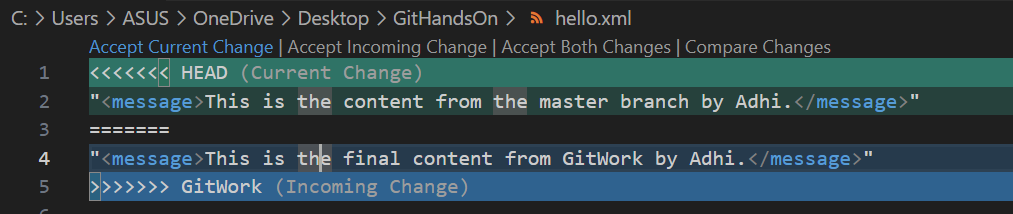
Step7: Merge the branch to the master and you can note the conflict



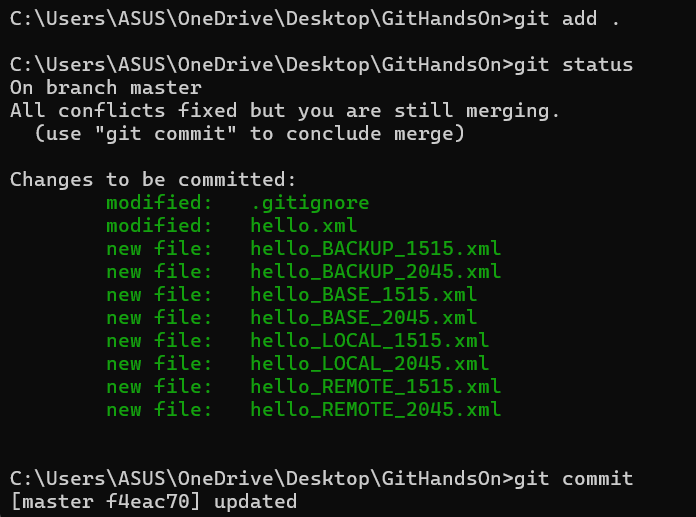
Step8: Open hello.xml in a text editor. You will see the conflict markers added by Git.



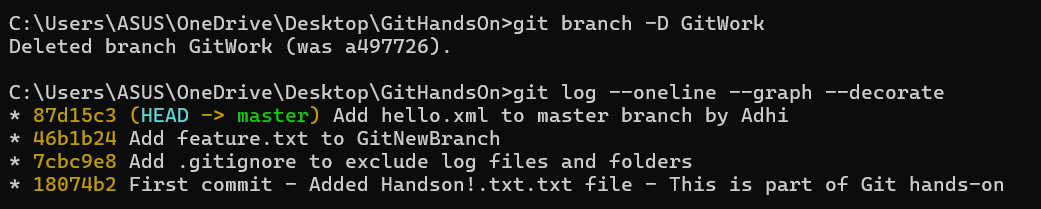
Step9: Select “Accept Current Change”







Step10: Now delete the branch



**Ex5: Git-HOL**

**Clean up and push back to remote Git**

1. Clean Up Your Local Repository

* First, you need to remove any unwanted files and folders from your project.
* To remove a file that is already tracked by Git, use the git rm <filename> command.
* To remove untracked files that Git doesn't know about, use the git clean -f command.
* To remove both untracked files and untracked folders, use the git clean -fd command.
* To check if there are any changes left, use the git status command.

2. Commit Your Cleaned Changes

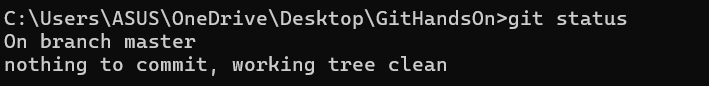
* Next,need to save the cleanup actions to your repository's history.
* To stage all your changes (like the file removals), use the git add . command.
* To commit the staged changes with a descriptive message, use the git commit -m "Your message" command.

3. Push Back to Remote

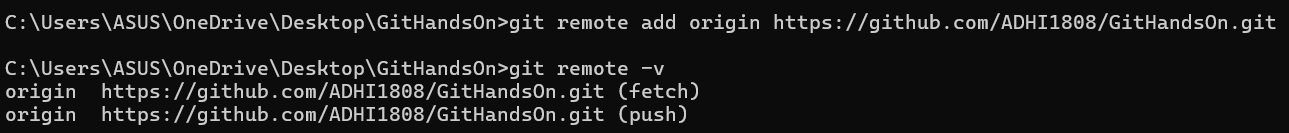
* Finally, update the remote repository with the cleaned version of your branch.
* To push your local commits to the remote repository, use the git push origin <branch-name> command.

Execute steps involving clean up and push back to remote Git.

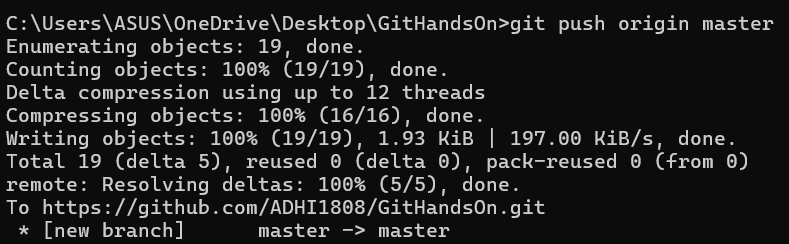
Step1: Check main status is clean



Step2: Add git remote origin using hhtps of repository



Step3: Pull the remote git repository to the master



Step4: Changes are reflected in the remote repository

