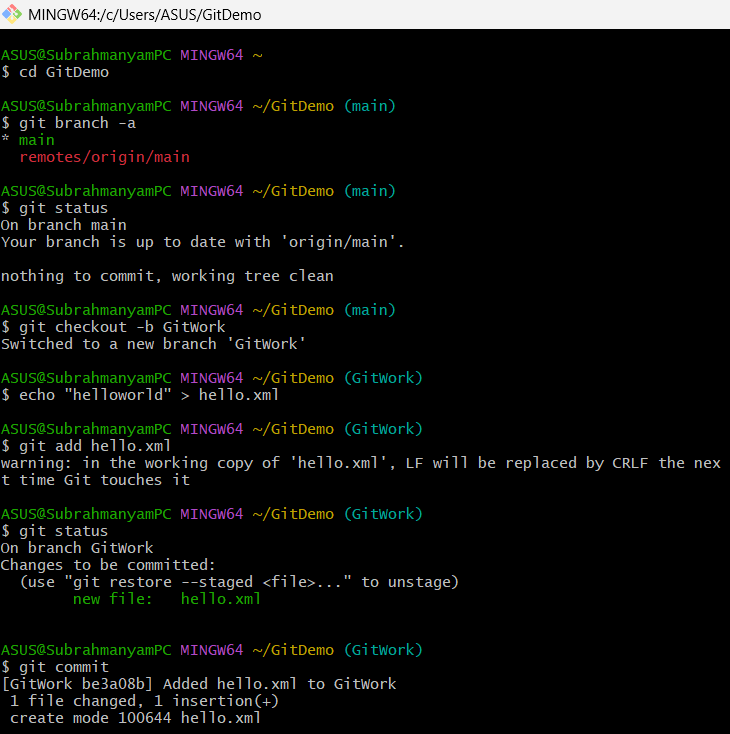
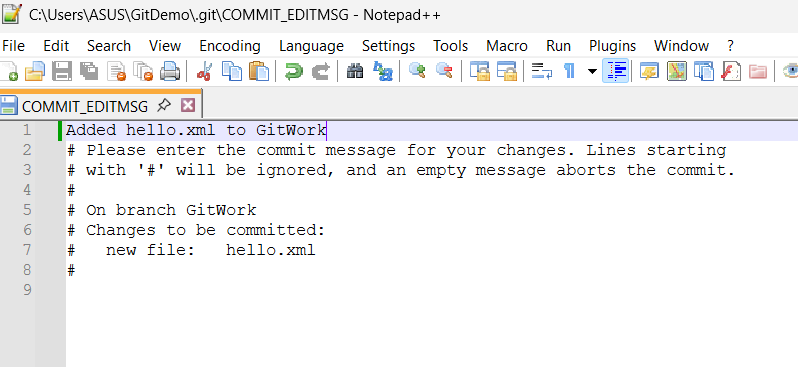
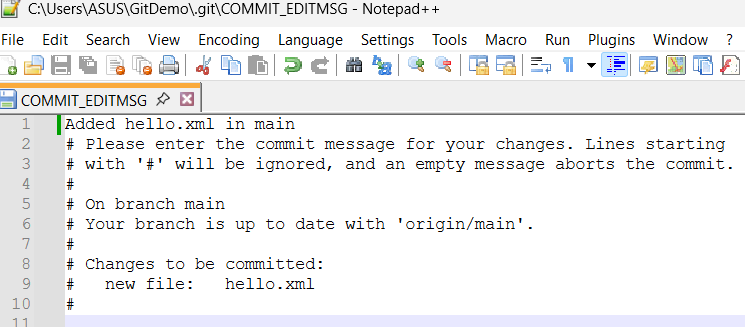
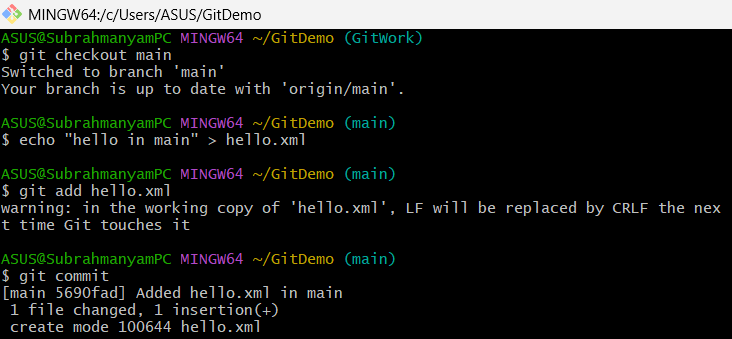
**Resolving merge conflicts**

Checked if main is clean, Created a hello.xml in GitWork (new branch) with content, “helloworld” and committed the changes to new branch.

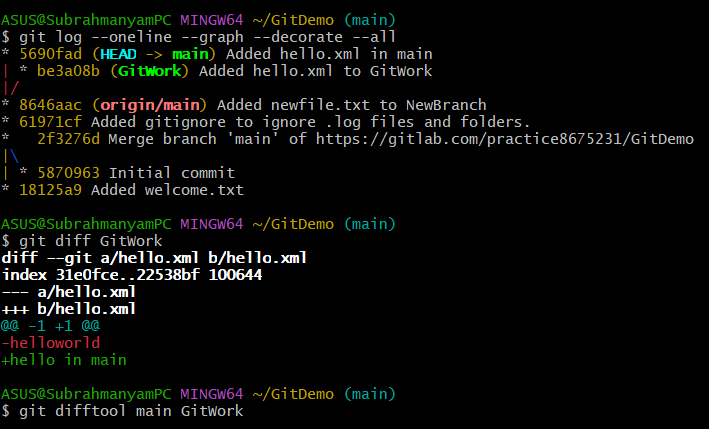




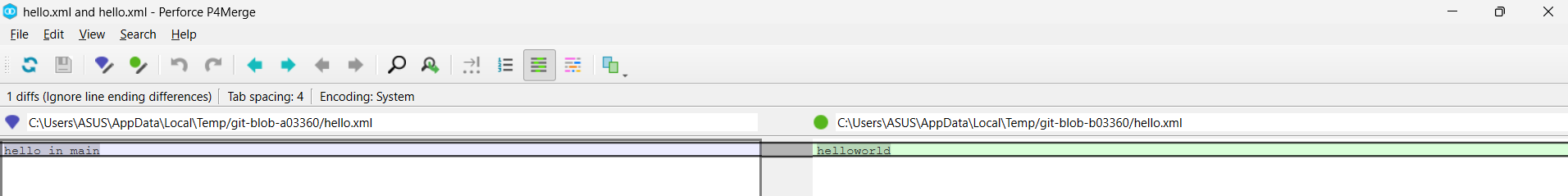
Switched to main and created another hello.xml with content “hello in main” and committed changes in main.



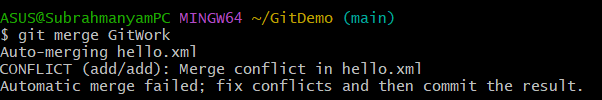
Log, Difference with branches using git diff and git diff tool p4merge tool (for better visualization)



P4 merge tool viz.



**THE CONFLICT**

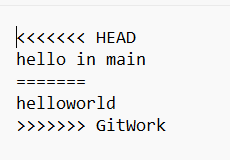


Git uses markers to show differences bw branches,

<<<<HEAD: refers to current branch (main)

===== : Divider between the two conflicting versions

>>>>: End of the content from the **incoming branch (Git Work)**



Used merge tool to resolve the conflict.

Conflict should be resolved manually

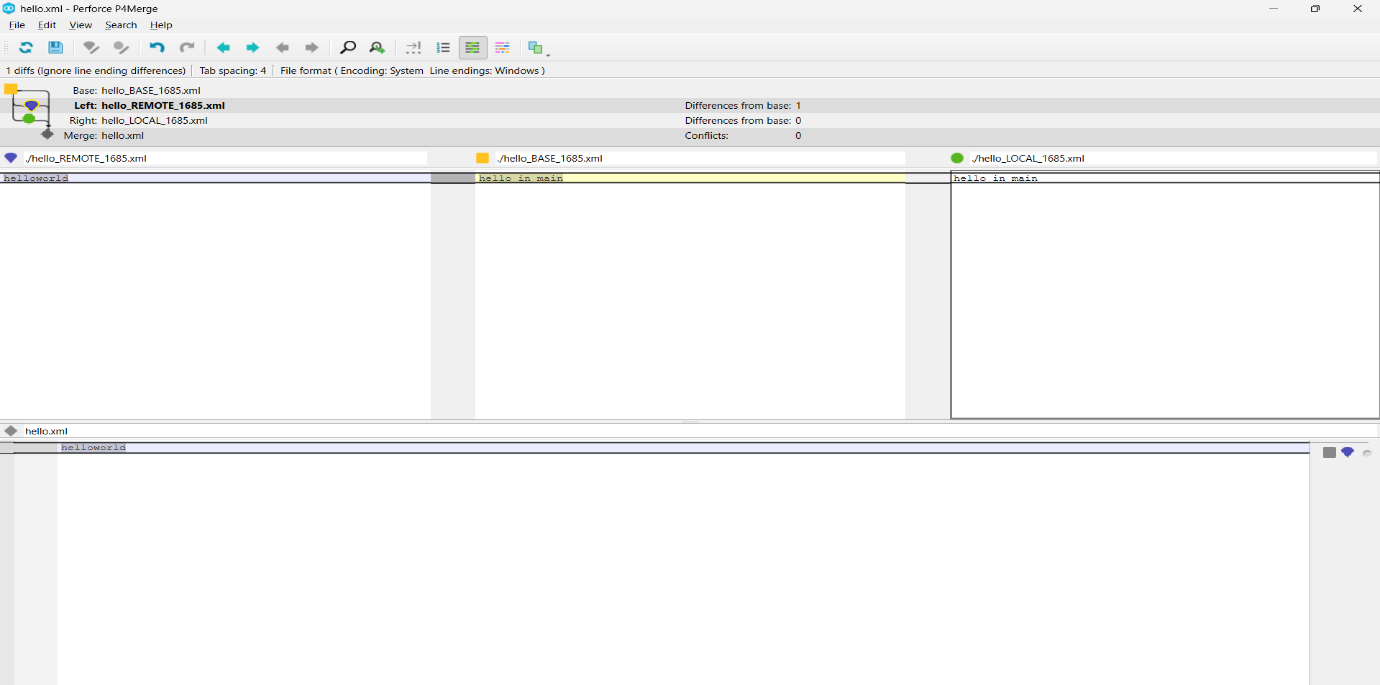
Merge tool helps us to do it visually, easily.



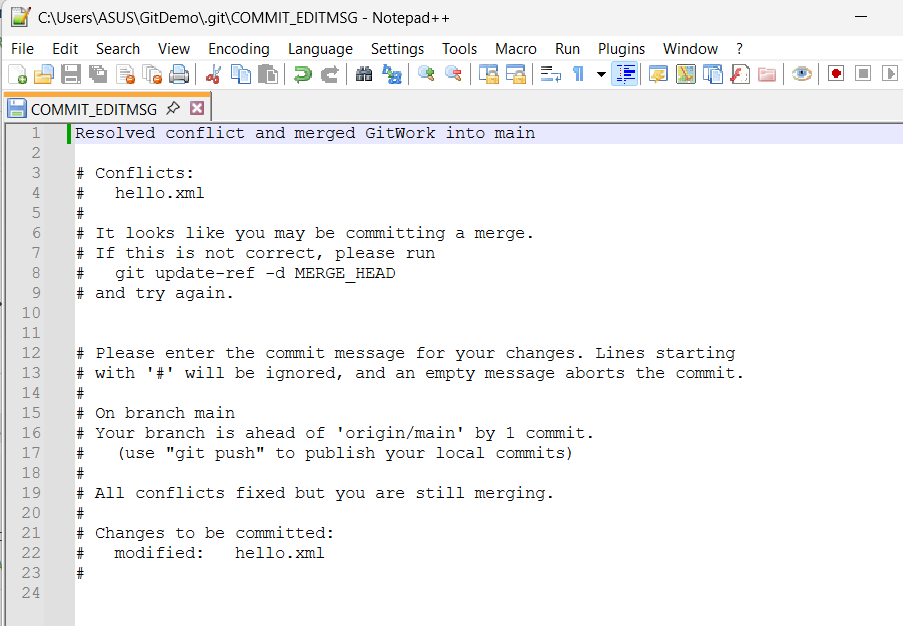
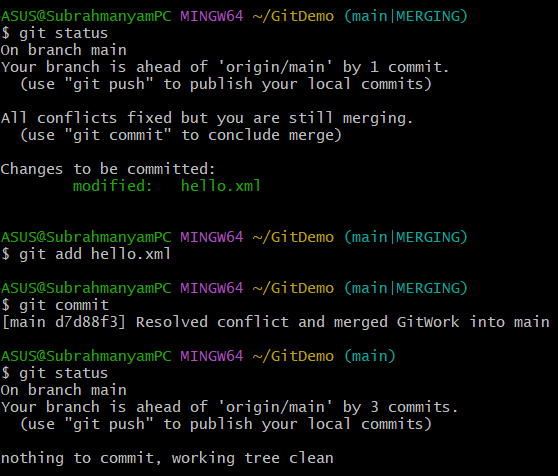
NEXT PAGE

Opens a window where,

* left pane is, main branch
* Right is GitWork
* Bottom pane is the final version
* Middle pane is the base version of both the files, since here both versions are created in their own branches, there is no base version
* Three symbols beside the, bottom pane allows us to select the pane for bottom pane( final version) , i.e which data to be in final version, also there is an none option as well
* Or we can edit the bottom pane manually
* Save and close the window
* Resolved conflict.



Git Status shows that, All the conflicts are resolved, hence staged the changes and committed

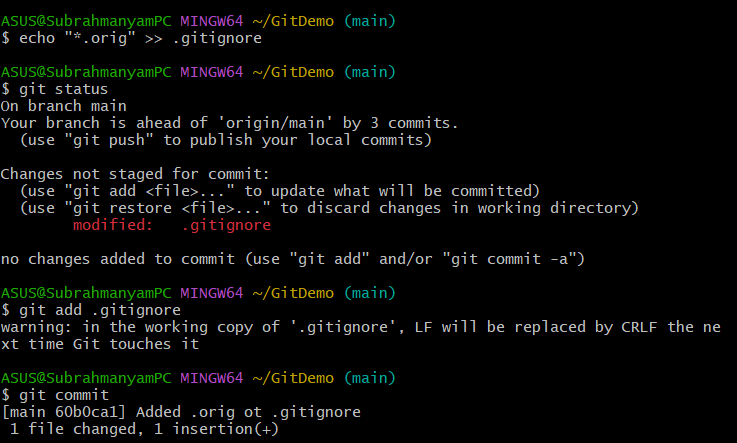


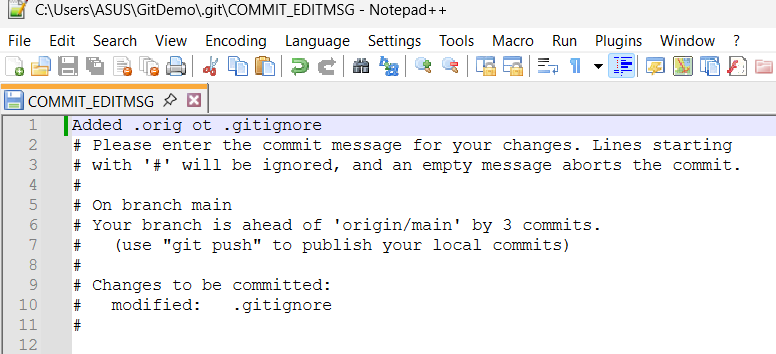
Git often **creates backup files** with the .orig extension before attempting to auto-merge or launch a merge tool. For example:

* hello.xml.orig

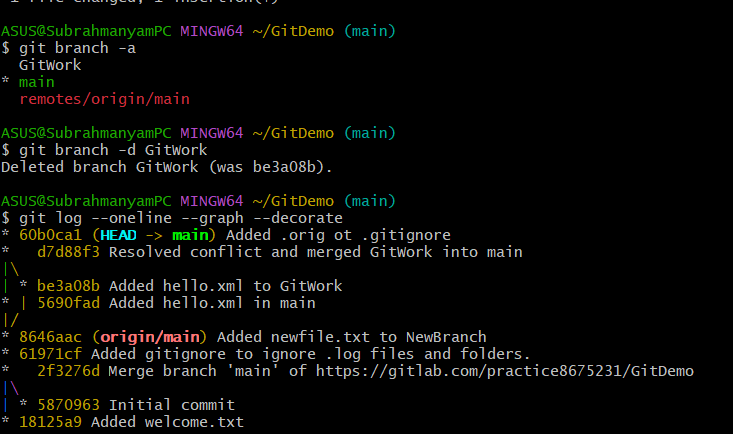
These files are not needed in our repo, and hence adding in .gitignore

Committing the .gitignore changes as well.





Deleting the GitWork branch and observing the logs



Finally pushing the changes to remote repo.

