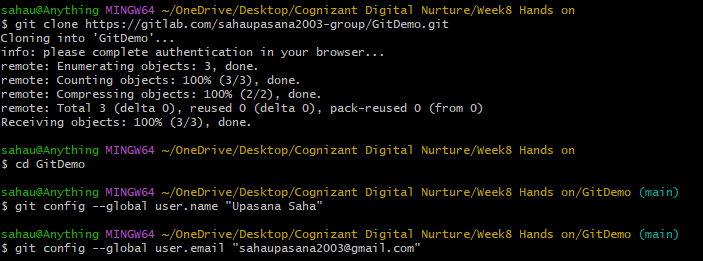
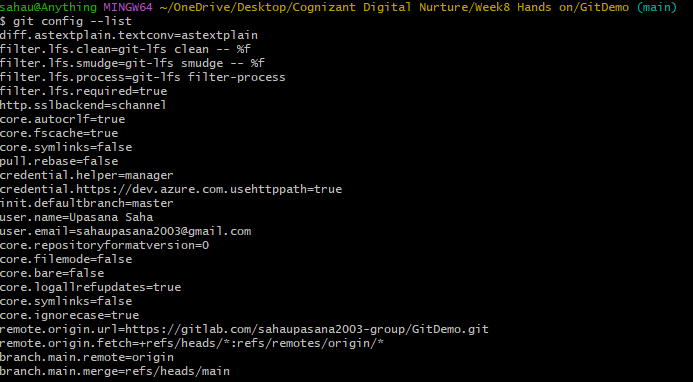
Week 8 Hands on – Module GIT

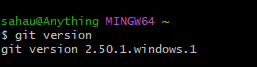
Hands on 1- Familiar with Git commands like git init, git status, git add, git commit, git push, and git pull.

1. To configure user level configuration of user ID and email ID execute



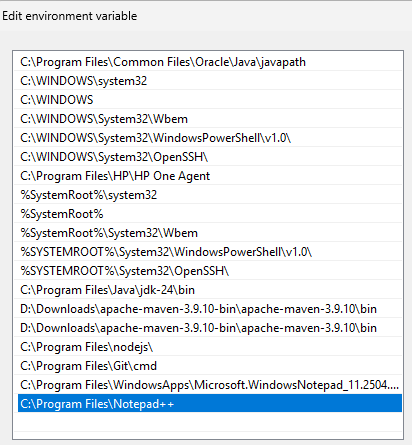


1. To check if the configuration is properly set, execute the following command.



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

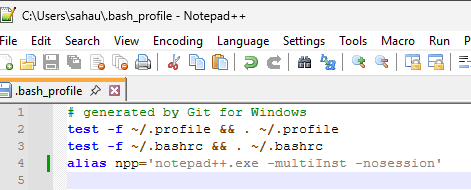
1. To check, if notepad++.exe execute from Git bash
2. Exit Git bash shell, open bash shell and execute

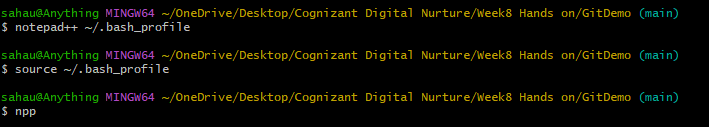


1. To create an alias command for notepad++.exe, execute



It will open notepad++ from bash shell, and create a user profile by adding the line in notepad++





npp is set as the alias name for notepad

1. To configure the editor, execute the command

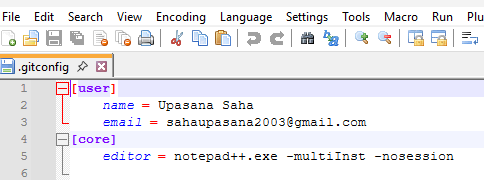


1. To verify if notepad++ is the default editor, execute the command



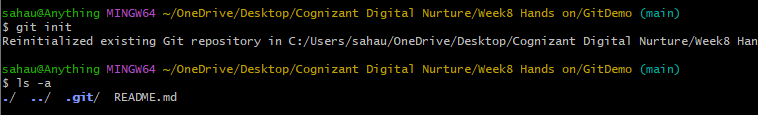
Here ‘-e’ option implies editor

It will show the entire global configuration

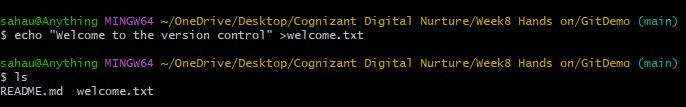


**Step 3: Add a file to source code repository**

1. Open Git bash shell and create a new project “**GitDemo**” by executing the command
2. Git bash initializes the “**GitDemo**” repository. To verify, execute the command

.

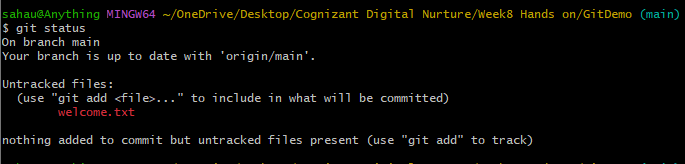
1. To create a file **“welcome.txt”** and add content to the file, execute the command
2. To verify if the file “welcome.txt” is created, execute



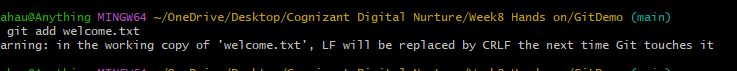
1. To verify the content, execute the command



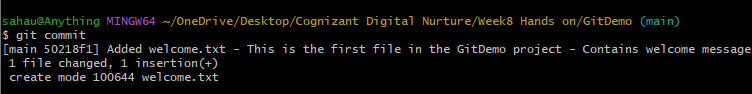
1. Check the status by executing



1. To make the file to be tracked by Git repository, execute the command

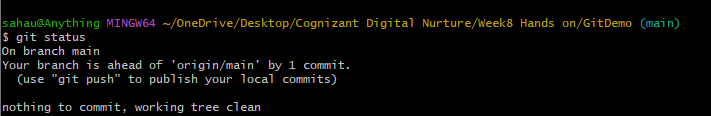


1. To add multi line comments, we are opening default editor to comment. Execute the command



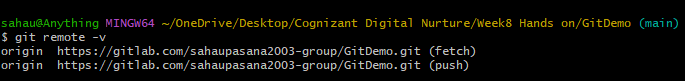
Notepad++ editor will open and to add multi-line comment with default editor

1. To check if local and “Working Directory” git repository are same, execute git status

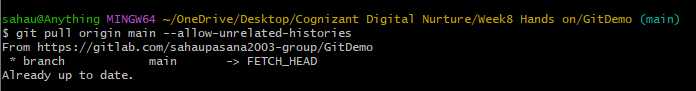


**welcome.txt** is added to the local repository.

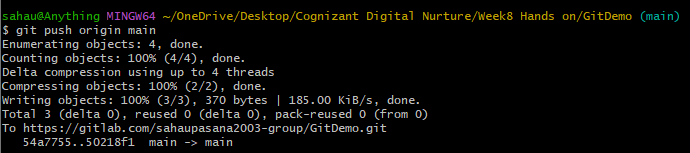
1. Signup with GitLab and create a remote repository **“GitDemo”**

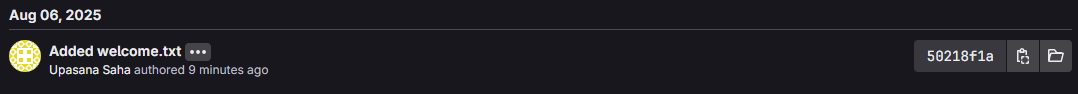
****

1. To pull the remote repository, execute



1. To push the local to remote repository, execute





Hands On 2 - Implement git ignore command to ignore unwanted files and folders

**What is .gitignore in Git?**

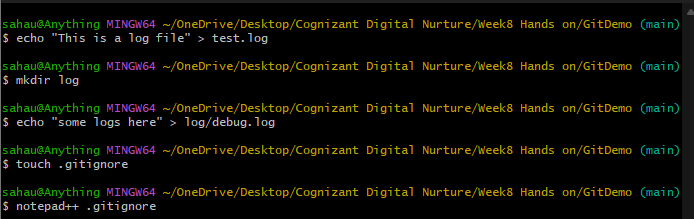
The .gitignore file tells Git which files or folders to ignore (i.e., not track).This is useful to avoid committing temporary files, logs, build files, IDE settings, etc.

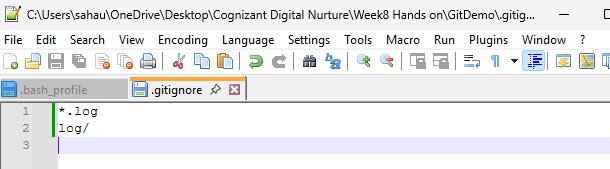
For example:

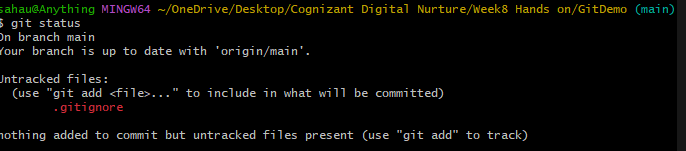
\*.log → Ignores all files ending with .log

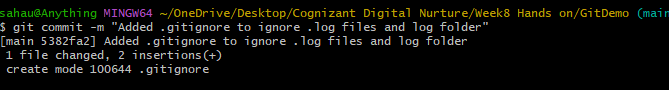
log/ → Ignores the log folder and its contents

Create a **“.log”** file and a **log folder** in the working directory of Git. Update the **.gitignore** file in such a way that on committing, these files (.log extensions and log folders) are ignored.Verify if the git status reflects the same about working directory, local repository and git repository









Hands On 3 - Construct a branch, do some changes in the branch, and merge it with master (or trunk)

Branching

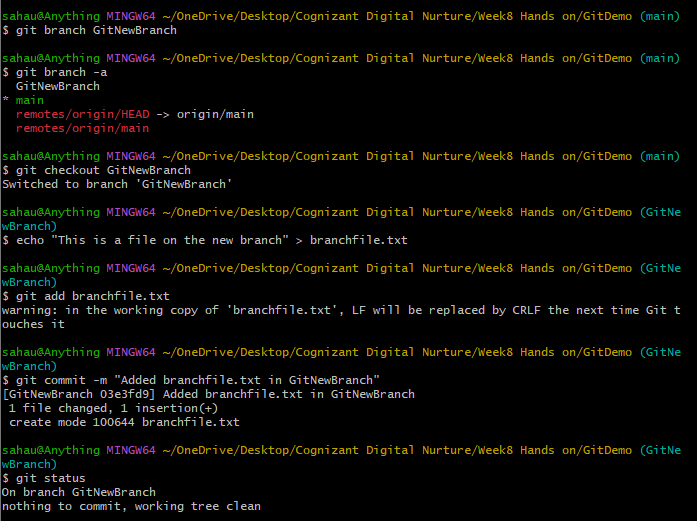
* A branch is a separate working line of development.
* Default branch is main (or master).
* Branching allows you to work on new features or fixes without affecting the main codebase.
* Example: main continues to work fine, while you test code on GitNewBranch.

Merging

* Merging combines changes from one branch (e.g., GitNewBranch) into another (e.g., main).
* Git compares commits and merges files together.

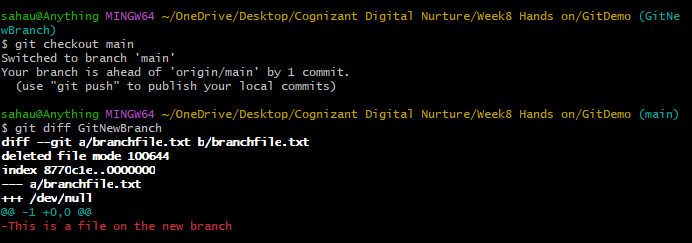
**Branching:**

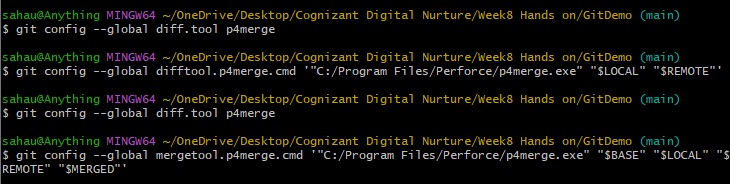
1. Create a new branch **“GitNewBranch”.**
2. List all the local and remote branches available in the current trunk. Observe the “\*” mark which denote the current pointing branch.
3. Switch to the newly created branch. Add some files to it with some contents.
4. Commit the changes to the branch.
5. Check the status with **“git status”** command.

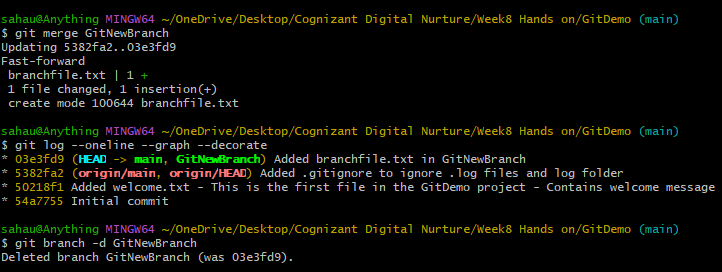


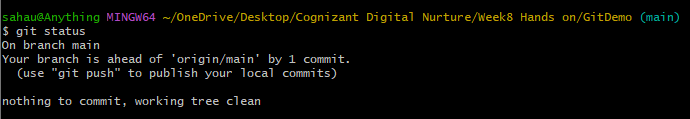
**Merging:**

1. Switch to the master
2. List out all the differences between trunk and branch. These provide the differences in command line interface.
3. Merge the source branch to the trunk.
4. Observe the logging after merging using **“git log –oneline –graph –decorate”**
5. Delete the branch after merging with the trunk and observe the git status.







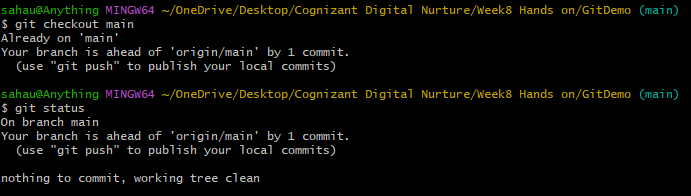


Hands On 4 - Implement conflict resolution when multiple users are updating the trunk (or master) in such a way that it results into a conflict with the branch’s modification.

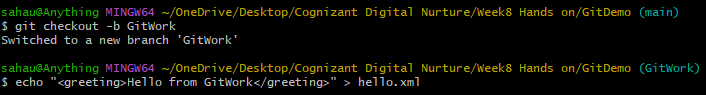
**How to resolve conflict during merge**

When two branches change the same part of a file differently, Git can't decide which one to keep and throws a merge conflict. You need to manually edit the file to resolve it.

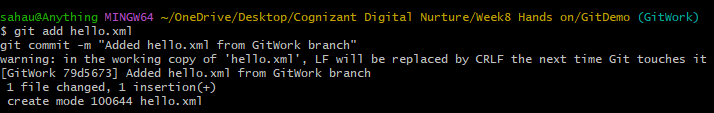
1. Verify if master is in clean state.

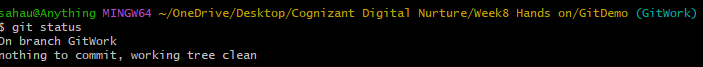


1. Create a branch “GitWork”. Add a file “hello.xml”.

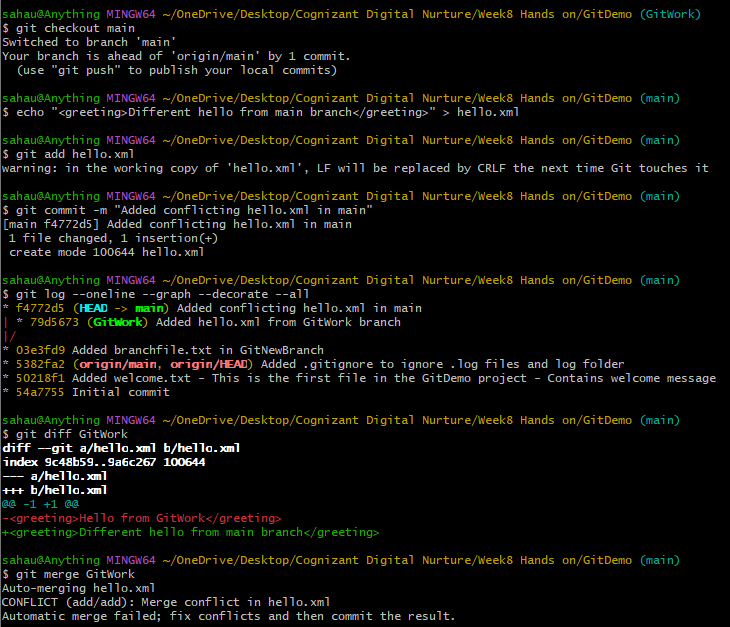


1. Update the content of “hello.xml” and observe the status
2. Commit the changes to reflect in the branch

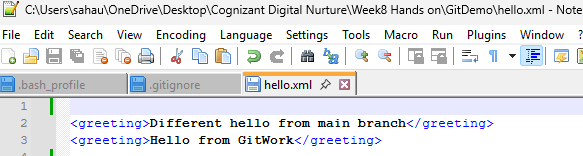




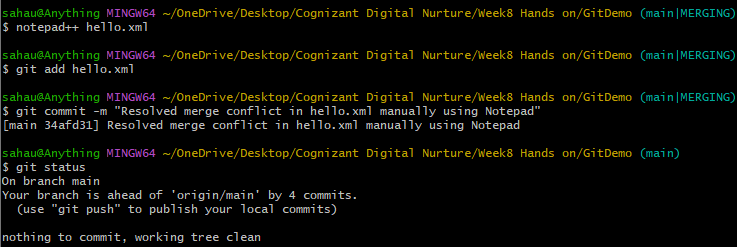
1. Switch to master.
2. Add a file “hello.xml” to the master and add some different content than previous.
3. Commit the changes to the master
4. Observe the log by executing “git log –oneline –graph –decorate –all”
5. Check the differences with Git diff tool
6. Merge the branch to the master
7. Observe the git mark up.



1. Use 3-way merge tool to resolve the conflict( P4merge was giving error s I used manually merge tool )



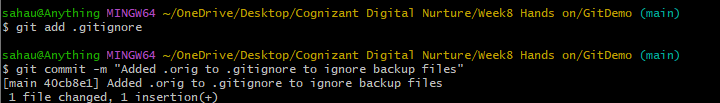
1. Commit the changes to the master, once done with conflict



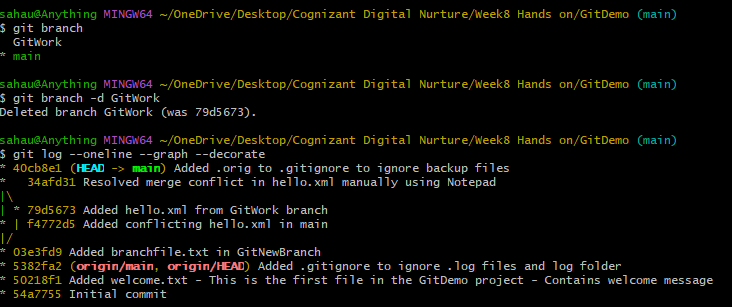
1. Observe the git status and add backup file to the .gitignore file.
2. Commit the changes to the .gitignore







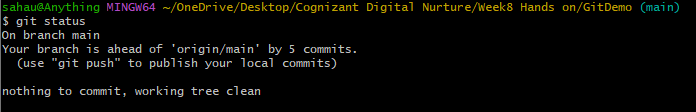
1. List out all the available branches
2. Delete the branch, which merge to master.
3. Observe the log by executing “git log –oneline –graph –decorate”



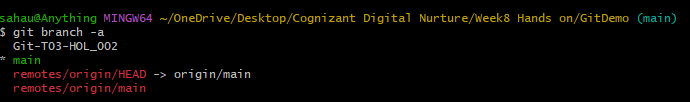
Hands On 5 - Execute steps involving clean up and push back to remote Git.

* **To clean up and push back to remote Git**, first delete any merged branches using git branch -d branch-name and ensure our working directory is clean with git status.
* Remove or ignore unnecessary files like backups by updating .gitignore and committing it.
* Then push our changes using git push origin main (or your branch name).
* If needed, delete remote branches with git push origin --delete branch-name. This keeps our local and remote repositories clean and in sync.

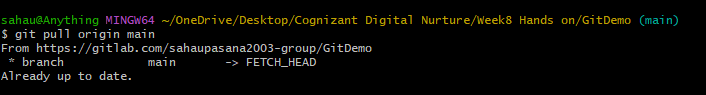
1. Verify if master is in clean state.



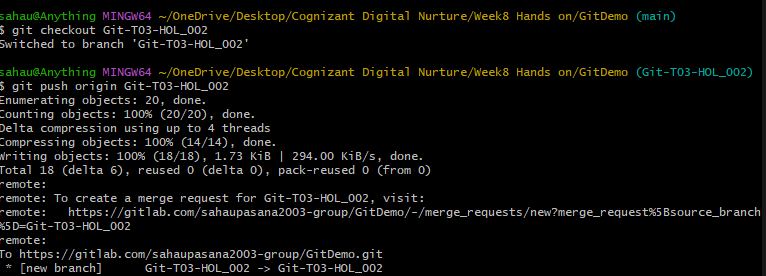
1. List out all the available branches.

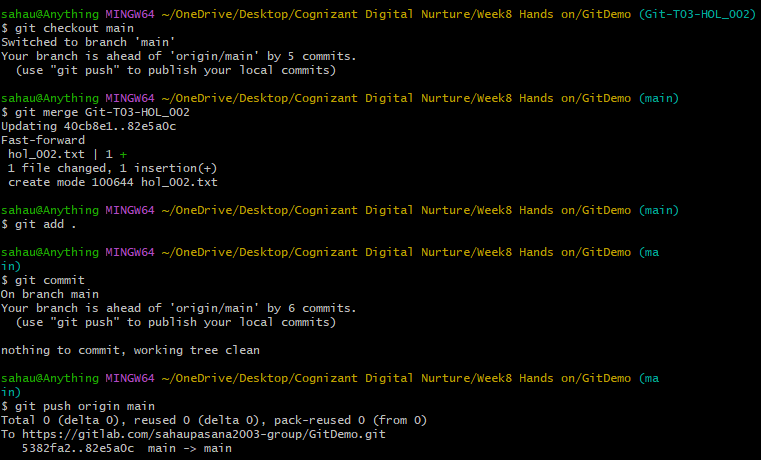


1. Pull the remote git repository to the master



1. Push the changes, which are pending from “Git-T03-HOL\_002” to the remote repository.





1. Observe if the changes are reflected in the remote repository.

