**Assignment: Using .gitignore to Ignore Unwanted Files in Git**

**Objective**

The goal of this hands-on lab is to understand how to use a .gitignore file effectively in a Git project. Specifically, I learned how to prevent certain unwanted files and folders—like logs or temporary files—from being tracked by Git.

**What I Learned**

* What .gitignore is and why it's used.
* How to ignore specific files and directories using .gitignore.
* How Git behaves before and after ignoring files.

**Step-by-Step Procedure**

**Step 1: Create the Required Files and Folder**

I navigated to my Git working directory (I used a folder named GitDemo) and created a .log file and a folder named log. I did this using the following Git Bash commands:

echo "This is a sample log file." > debug.log

mkdir log

echo "Log folder content." > log/logfile.txt

To verify that the files were created:

ls

ls log

This confirmed that both debug.log and log/logfile.txt were successfully created.

**Step 2: Check Git Status Before Adding .gitignore**

Next, I checked the current Git status to see how Git was tracking these files:

git status

**Output:**

Git showed both debug.log and the contents of the log/ folder as untracked files. This meant they were visible to Git and would be included in commits unless I told Git to ignore them.

**Step 3: Create and Configure .gitignore**

To tell Git which files to ignore, I created a .gitignore file:

notepad++ .gitignore

Inside the .gitignore file, I added the following lines:

\*.log

log/

Here’s what each line means:

* \*.log — This pattern ignores all files with a .log extension.
* log/ — This line ignores the entire log directory and everything inside it.

After saving and closing the file, I was ready to test if it worked.

**Step 4: Verify That Git is Ignoring the Files**

I checked Git’s status again:

git status

**Output:**

At this point, only the .gitignore file appeared in the list of changes to be staged. Git was no longer tracking debug.log or the contents of the log/ folder. This confirmed that the ignore rules were working exactly as intended.

**Step 5: Stage and Commit the .gitignore File**

Once I confirmed the ignore behavior was correct, I staged and committed the .gitignore file:

git add .gitignore

git commit -m "Added .gitignore to ignore .log files and log folder"

If I needed to push these changes to a remote repository, I used:

git push origin master

**Step 6: Confirm That the Files Are Permanently Ignored**

Finally, I verified once more:

git status

**Output:**

There were no .log files or log/ folders listed in the staging area. The working directory was clean, and Git confirmed there were no new files to be tracked. Even after I added more .log files or nested files inside the log/ folder, Git continued to ignore them as expected.

