Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report 4
[Code No: COMP 307]

Submitted by:

Nischal Subedi (53)

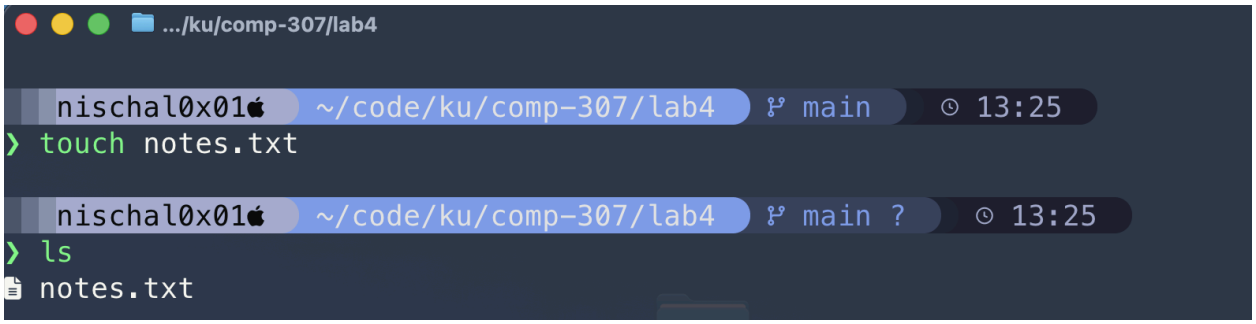
Group: CS60 (III/I)

.

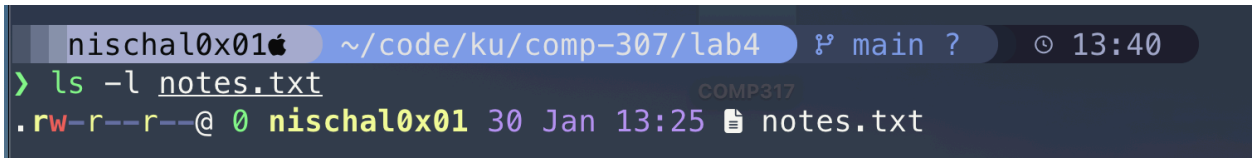
Submitted to:
Ms. Rabina Shrestha
Department of Computer Science and Engineering

Submission Date: 30/01/2026

Task-1 Create a file named notes.txt



a. Run ls -l notes.txt. What is the default permission string?



The default permission string is -rw-r–r–

b. Explain what each part of the permission string means(owner, group, others)

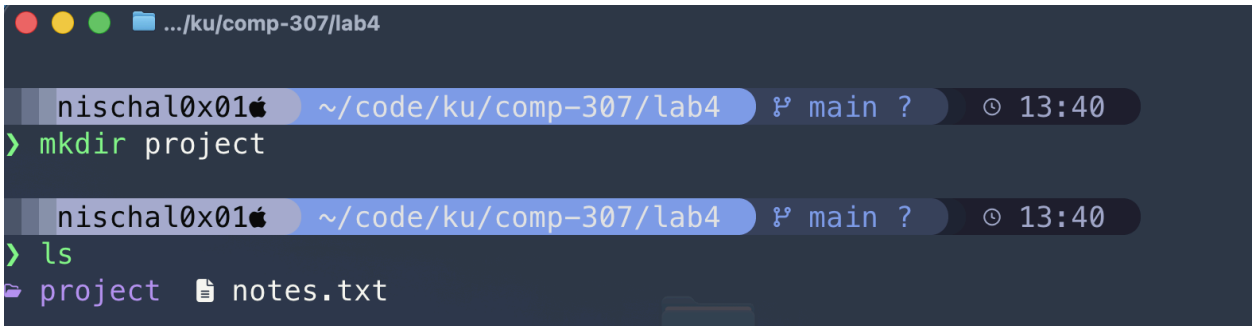
The permission string `-rw-r--r--` consists of 10 characters:

Position	Meaning
-	File type (- = regular file)
rw-	Owner permissions (read, write)
r--	Group permissions (read only)
r--	Others permissions (read only)

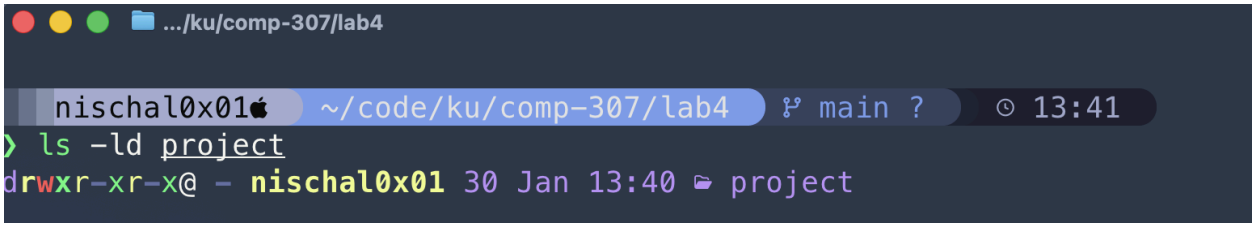
Permission Breakdown

- Owner: Can read and write the file.
- Group: Can only read the file.
- Others: Can only read the file.
- Execute permission is not set, which is typical for regular text files.

Task-2: Create a directory called project



a. Run ls -ld project



b. Compare the permission string of the directory with the file. Why does the directory show rwx for execution?

The permission string of a directory is different from that of a file because the execute (x) permission has a special meaning for directories. In a directory, execute permission allows a user to enter the directory and access its contents.

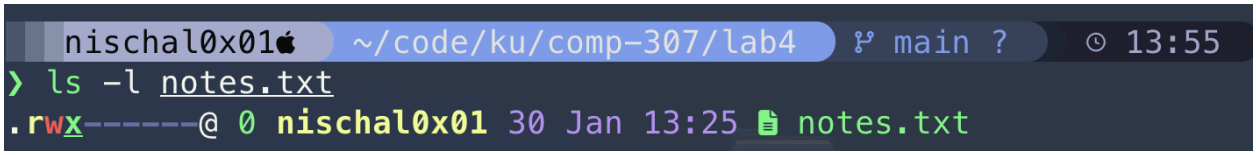
For example, a directory may have permissions like `drwxr-xr-x`, where:

- `r` allows listing the contents of the directory
- `w` allows creating, deleting, or renaming files inside the directory
- `x` allows entering the directory (`cd project`) and accessing files within it

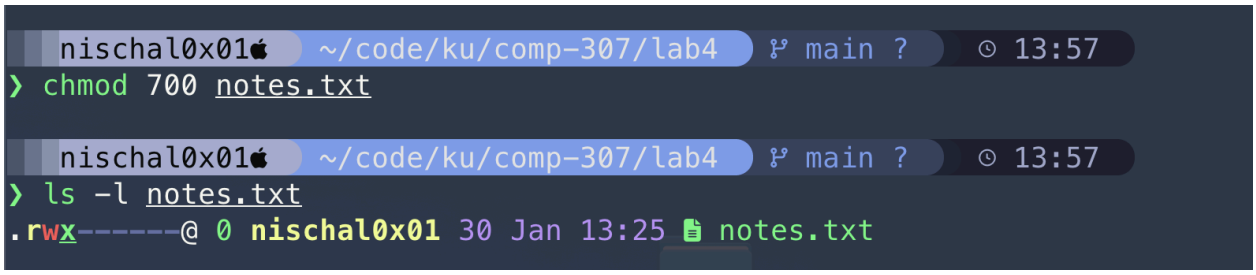
Without execute permission on a directory, a user cannot enter it or access the files inside, even if read permission is granted. This is why directories typically show `rwx` permissions, while regular files usually do not include execute permission unless they are meant to be run as programs.

Task-3: Using chmod (change permissions) on notes.txt file

a. **Use `chmod u+rwx,g-rw,o-rwx notes.txt` to change the permissions so only the owner can read, write, and execute, but group and others have no permissions. What does `ls -l` now show?**



b. **Try using the octal method to get the same result instead (e.g., `chmod 700 notes.txt`). Explain why 700 works.**



In Linux, file permissions can be set using octal (numeric) values, where each digit represents permissions for owner, group and others.

Each permission has a numeric value:

- Read (r) = 4
- Write (w) = 2
- Execute (x) = 1

Now break down 700:

- 7 (owner) → 4 + 2 + 1 = read, write, execute (rwx)
- 0 (group) → no permissions (---)
- 0 (others) → no permissions (---)

So, when we run chmod 700

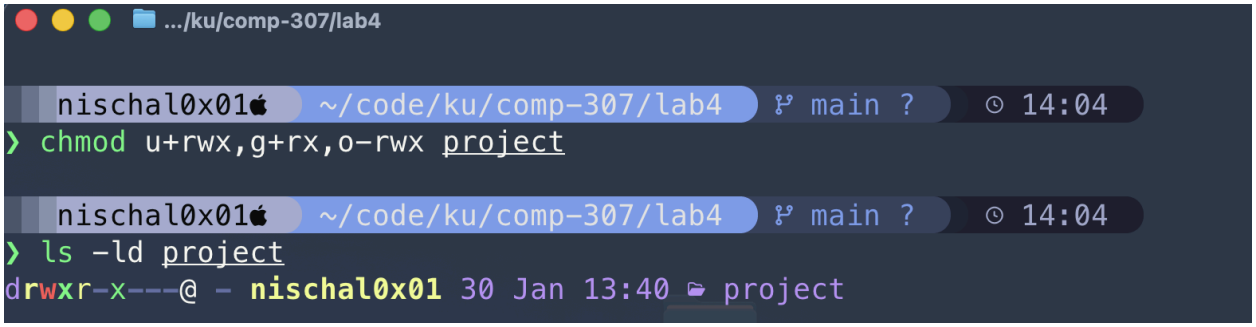
It gives:

- Full permissions to the owner
- No permissions to the group and others

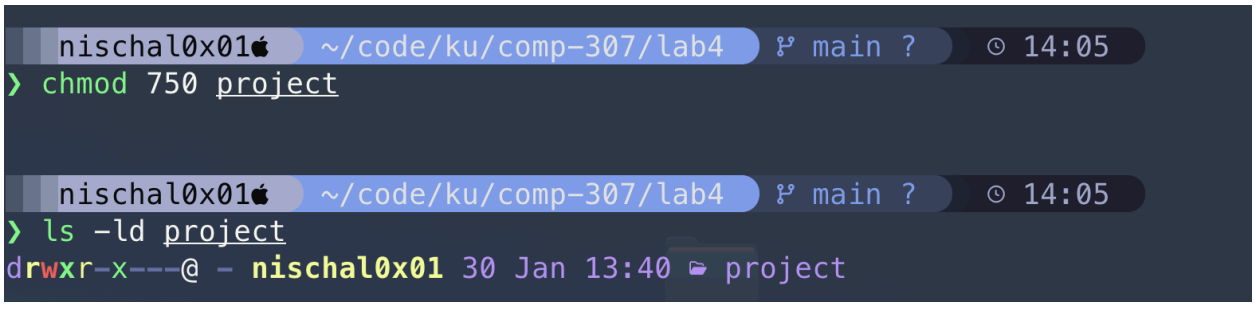
This matches exactly what *chmod u+rwx,g-rw,o-rwx notes.txt* does, which is why 700 works.

Task-4: Using chmod (change permissions) on project folder

a. Make it so the group can read and execute, but others have no rights.



b. Verify with `ls -ld project`. What numeric permission did you use?



This sets:

- Owner (7) → read, write, execute (rwx)
- Group (5) → read and execute (r-x)
- Others (0) → no permissions (---)

Task-5: Short Answer Questions

1. **Why is it not recommended to use `chmod 777` on important files or directories?**
   It is not recommended to use chmod 777 on important files or directories because `chmod 777` gives read, write, and execute permissions to everyone, which creates serious security vulnerabilities. Unauthorized users can alter or delete critical files, and malicious programs may exploit these permissions to damage the system. It also violates the principle of least privilege, making systems harder to secure and audit.

2. **When would changing the group of a file (with `chgrp`) be more appropriate than changing its owner (with `chown`)?**
   Changing the group is more appropriate when several users need shared access to the same files without changing ownership. This approach is commonly used in collaborative environments where team members belong to the same group. It helps maintain accountability while still allowing controlled access for multiple users.