Topic: File Permissions

File Permissions describe the allowed operations by various users.

With respect to file permissions, all users are categorized into the following 4 types.

User Categories:

user/owner ---→ Represented by 'u'

group ----→ Represented by 'g'

others ----→ Represented by 'o'

all ---→Represented by 'a'

Use Case to understand Types of Users:

Project: PYTHONLIFE

This project divided into multiple modules. In each module multiple developers are working.

1) STUDENTS MODULE

A, B, C, D ARE WORKING

2) EMPLOYEES MODULE

X, Y, Z ARE WORKING

3) COURSES MODULE

M, N ARE WORKING

4) INFRASTRUCTURE MODULE

G, H ARE WORKING

DEVELOPER 'A' CREATED ONE FILE demo.txt

For demo.txt

User/owner: A (The person who created the file)

Group: B,C,D (The persons who are working in the same module)

Others: X,Y,Z,M,N,G,H (The persons who are working on other modules)

Permission Types:

For files and directories, there are 4 types of permissions.

1) r ---→ Read

2) w --→ Write

3) x --→ Execute

4) - ---→No Permission

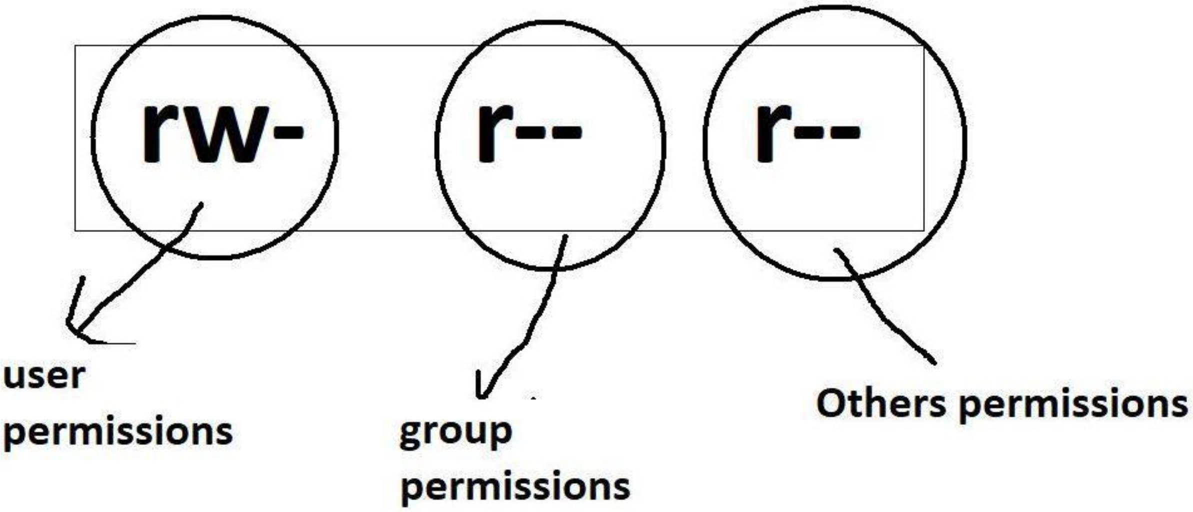
How to check Permissions of existing File:

By using **ls -l** command:

```
total 0

-rw-r--r-- 1 python python0 April 16 21:19 demo.txt
```

The file permissions are



Total 9 permissions. First 3 are user permissions, next 3 are group permissions and next 3 are others permissions.
user permissions: rw-
user can perform both read and write operations but not execute operation
group permissions: r--
group members can perform only read operation and cannot perform write and execute operations
others permissions: r--
other members can perform only read operation and cannot perform write and execute operations.
User Permissions + Group Permissions + Others Permissions
Order is important

Read permission + Write Permission + Execute Permission
order is important

Eg 1: $ chmod u+x demo.txt
adding execute permission to the user

Eg 2: $ chmod u+w,g+rw,o+r demo.txt
adding write permission to the user
adding read and write permissions to the group
adding read permission to the others
Eg 3: $chmod u+x,g-w,o+w demo.txt
adding execute permission to the user
removing write permission from the group
adding write permission to the others
Eg 4: $ chmod u=rw,g=rw,o=r demo.txt
Now user permissions: rw-
group permission: rw-
others permission: r—

Eg 5: $ chmod a=- demo.txt
Now user permissions: ---
group permission: ---
others permission: ---
Eg 6: $ chmod a=rwx demo.txt
Now user permissions: rwx
group permission: rwx
others permission: rwx

Numeric Permissions:

We can specify permissions by using octal number.
Octal means base-8 and allowed digits are 0 to 7.
0 → 000 ➔ No Permission
1 → 001 ➔ Execute Permission
2 → 010 ➔ Write Permission
3 → 011 ➔ Write and execute Permissions
4 → 100 ➔ Read Permission
5 → 101 ➔ Read and execute Permissions
6 → 110 ➔ Read and write Permission
7 → 111 ➔ Read, Write and execute Permissions

Note:

4 → Read Permission
2 → Write Permission
1 → Execute Permission
It is more easy to remember
5 → 4+1 → r-x
3 → 2+1 → -wx
6 → 4+2 → rw-
etc
1. Write command for the following permissions
For user → Read and write (6)
For group → Write and execute (3)

For others → Write (2)

umask Command:

umask means user mask. Hiding permissions.
Based on umask value,default permissions will be there for files and directories.
The default umask value:022

```
Desktop$ umask
0022
```

First 0 is sticky bit mostly used in admin related activities. We have to consider only last 3 digits as umask value.

Default permissions to the file: 666 - umask value
= 666 - 022
= 644 (user → r&w, group → read, others → read)

chown Command:

chown means change owner.
Only root user can perform this activity.

```
# chown root demo.txt
```

Now the owner of demo.txt is root.

chgrp Command:

chgrp means change group.
Only root user can perform this activity.

```
# chgrp root demo.txt
```

Now the demo.txt belongs to root group